

NevProp3[®] USER MANUAL

Nevada backPropagation, version 3

ARTIFICIAL NEURAL NETWORK SIMULATOR for STATISTICAL INFERENCE

LICENSE AND NON-WARRANTY

This program is distributed for free under the terms of the GNU Public License (contained in the file COPYING), as is and with no warranty or obligation of support. Those wishing to obtain a warranty, support, or alternative licensing arrangements may contact the principal author.

Citation format for publication of studies using NevProp3 (to be modified according to journal specification):

Goodman, PH. *NevProp software, version 3*. Reno, NV: University of Nevada; 1996.

URL: <http://www.scs.unr.edu/nevprop/>

Software design supervised and manual written by Philip H. Goodman <goodman@unr.edu>, who assumes responsibility for any inaccuracies contained therein.

Programming Staff, 1992-1996, alphabetically

(please see also the file, Credits.doc)

Derek Carlson	Larry McKnight
Dwight Egbert	Ashish Purankar
Phil Goodman	David Rosen
Joyce Hallet	Fei Sha

Departments of Internal Medicine, Electrical Engineering, and Computer Science
University of Nevada, Reno

Development of NevProp was supported in part by:

U.S. Public Health Service Agency for Health Care Policy and Research
grant HS06830, "Outcomes by Neurocomputing"

•

U.S. Department of Defense grant DAMD17-94-4-4383,
"Developing the AJCC Prognostic System for Breast Cancer"

NevProp3® USER MANUAL



TABLE OF CONTENTS

(Note: Chapters and subchapter numbers are preceded by “&” for easy computer searching)

COVER PAGE / COPYRIGHT

TABLE OF CONTENTS (you are here)

&1. INTRODUCTION

- &1.1 Background
- &1.2 Features
- &1.3 Technical Support
- &1.4 Bug Reports

&2. AN OVERVIEW OF KNOWLEDGE, STATISTICS, & ARTIFICIAL NEURAL NETWORKS

- &2.1 Knowledge
- &2.2 Statistical Methods
- &2.3 Generalized Linear Models
- &2.4 General Nonlinear and Artificial Neural Network Models
- &2.5 Drawing Inference in Generalized Linear Models
- &2.6 Drawing Inference in Artificial Neural Network Models
- &2.7 Dealing with Missing Data

&3. OBTAINING & SETTING UP NevProp VERSION 3

- &3.1 Availability
 - 3.1.1 World Wide Web
 - 3.1.2 Direct Internet file transfer (“ftp”)
- &3.2 Compilation/Installation

&4. INTRODUCTORY TUTORIALS (Recommended for all levels of experience)

- &4.1 Continuous Prediction
- &4.2 Classification

&5. GENERAL OPERATION

- &5.1 Interface Options
 - 5.1.1 Fully-Interactive Mode
 - 5.1.2 Command-Line Argument Mode
 - 5.1.3 Interrupting Training
- &5.2 Formatting the Network (**.net**) File
- &5.3 Formatting & Reading Data
 - 5.3.1 Data appended within the **.net** File
 - 5.3.2 External data files (**.trn** & **.tst**)
 - 5.3.3 Handling Missing Data
- &5.4 Saving NevProp3 Files
 - 5.4.1 Results (**.res**) File

December 14, 1998 11:24 PM

- 5.4.2 Predictions (**.ptr**, **.pts**) Files
- 5.4.3 Weights (**.wts**) File
- 5.4.4 Imputation (**.itr**, **.its**) Files

(CONTENTS, continued)

- &5.5 General Approach to Network Design & Regularization
- &5.6 Interpreting the Results of NevProp3 Procedures
 - 5.6.1 Overview of results display
 - 5.6.2 Understanding NevProp3's measures of accuracy
- &5.7 Single Dichotomous Output (dependent) Variable (binary classification)
- &5.8 Multiple Dichotomous Output Variables (1-of-N or M-of-N classification)
- &5.9 Continuous Outputs
- &5.10 Mixed Dichotomous & Nondichotomous Outputs

&6. ADVANCED TOPICS

- &6.1 Bootstrapped-Corrected Model Performance & Standard Errors
- &6.2 Parametric Inference– Determining Mean Effects of Predictors **[IN**

PROGRESS]

- &6.3 Convergent and Non-Convergent Regularization **[IN PROGRESS]**
- &6.4 Variable Selection– Effects of Individual Predictors (ARD) **[IN PROGRESS]**
- &6.5 Prediction & Imputation Using ANN-~~kNN~~ mode **[IN PROGRESS]**
- &6.6 Ensemble (committee) of experts **[IN PROGRESS]**
- &6.7 Interactions Among Predictors (separate NevGA application) **[IN PROGRESS]**

&7. SETTINGS Reference

- &7.1 Defaults and Bare-Minimum Network (.net) file
- &7.2 NP File Settings
 - 7.2.1 **ResultsFile (.res)**
 - 7.2.2 **SaveTrainPrdFile (.ptr)**
 - 7.2.3 **SaveTestPrdFile (.pts)**
 - 7.2.4 **SaveTrainImputFile (.itr)**
 - 7.2.5 **SaveTestImputFile (.its)**
 - 7.2.6 **SaveWeightsFile (.wts)**
 - 7.2.7 **ReadWeightsFile (.wts)**
- &7.3 DATA File Settings
 - 7.3.1 **ReadTrainFile (.trn)**
 - 7.3.2 **ReadTestFile (.tst)**
 - 7.3.3 **NHeaders**
 - 7.3.4 **IDColumn**
 - 7.3.5 **StandardizeInputs**
 - 7.3.6 **ImputeMissing**
 - 7.3.7 **ShuffleData**
 - 7.3.8 **NVars**
 - 7.3.9 **InputColumns**
 - 7.3.10 **OutputColumns**
 - 7.3.11 **SaveStandWts**
- &7.4 CONFIGURATION Settings
 - 7.4.1 **Ninputs , Noutputs**
 - 7.4.2 **Nhidden**
 - 7.4.3 **HiddenUnitType**

- 7.4.4 **OutputUnitType**
- 7.4.5 **1ofN** option (for binary outputs)
- 7.4.6 **Connecting** the network

(CONTENTS, continued)

- 7.4.7 **WeightRange**
- 7.4.8 **kNN** mode (initiates ANN model of k-nearest neighbor)
- 7.4.9 **Expert** specification (initiates Ensemble mode to weight experts)
- 7.4.10 **EnsemblePrior**
- &7.5 TRAINING (optimization) Settings
 - 7.5.1 **TrainCriterion** (objective, or criterion function)
 - 7.5.2 **OptimizeMethod**
 - 7.5.3 **LearnRate**
 - 7.5.4 **SplitLearnRate**
 - 7.5.5 **QPModeSwitchThreshold** (quickprop setting)
 - 7.5.6 **QPMaxFactor** (quickprop setting)
 - 7.5.7 **Stochastic**
 - 7.5.8 **SigmoidPrimeOffset**
 - 7.5.9 **Momentum**
 - 7.5.10 **WeightDecay** (simple penalty)
 - 7.5.11 **BiasPenalty**
- &7.6 BEST-BY-HOLDOUT Settings
 - 7.6.1 **NHoldout, PercentHoldout**
 - 7.6.2 **MinEpochs**
 - 7.6.3 **BeyondBestEpoch**
 - 7.6.4 **AutoTrain**
 - 7.6.5 **NSplits**
 - 7.6.6 **SepBootXVal**
- &7.7 AUTOMATIC RELEVANCE DETERMINATION Settings
 - 7.7.1 **UseARD**
 - 7.7.2 **WhenARD**
 - 7.7.3 **ARDTolerance**
 - 7.7.4 **ARDFreq**
 - 7.7.5 **GroupSelection**
 - 7.7.6 **ARDFactor**
 - 7.7.7 **BiasRelevance**
- &7.8 REPORTING Settings
 - 7.8.1 **DescribeVars**
 - 7.8.2 **CalccIndex**
 - 7.8.3 **ScoreThreshold**
 - 7.8.4 **NBoots**
 - 7.8.5 **NEffectBoots**
 - 7.8.6 **OutputStatVars**

&8. ERROR MESSAGES

&APPENDICES

- Appendix I Cross-Reference of Common Statistical and ANN Jargon
- Appendix II Resources: FAQ Neural Networks

CHAPTER &1. INTRODUCTION

&1.1 Background

NevProp is a contraction of the words, “Nevada backPropagation.” NevProp, version 3, is a relatively easy-to-use, feedforward backpropagation multilayer perceptron simulator– that is, statistically speaking, a multivariate nonlinear regression program. NevProp3 is distributed for free under the terms of the GNU Public License (contained in the file COPYING). NevProp3 is controlled by either an interactive character-based interface, or a command-line argument. The program is distributed as C source code that should compile and run on most platforms. In addition, precompiled executables are available for Macintosh and DOS platforms.

NevProp3 was developed as a multidisciplinary research project at the University of Nevada Center for Biomedical Modeling Research. The development was funded, in part, by the University’s Graduate School and Departments of Internal Medicine, Electrical Engineering, and Computer Science, and by grants from the U.S. Public Health Service, Agency for Health Care Policy and Research (R01-HS06830, “Outcomes by Neurocomputing”), and the U.S. Department of Defense (DAMD17-94-4-4383, “Developing the AJCC Prognostic System for Breast Cancer”). As a spin-off of that research, we now share the software with other applied and theoretical researchers who might benefit from our efforts.

The development of NevProp was motivated by an interest in improving the prediction and classification accuracy of mathematical models derived from epidemiological healthcare databases (although it is applicable to databases in general). During a 1990 sabbatical as a Robert Wood Johnson health policy fellow in the U.S. Senate, the principal author appreciated the scientific and economic need to analyze large health care databases derived from the practice of medicine, rather than randomized clinical trials. Such databases usually have many biologically and socially informative variables, and are relatively inexpensive to collect (often as a by-product of quality review and retrospectively extracted from patient charts). On the other hand, many variables may be noninformative, partially redundant, subject to erroneous documentation and abstraction, and/or, replete with missing elements.

Needed, then, was the ability to extract maximal predictive and inferential information from many records and variables, while controlling the influence of bias, noise, and missingness. Trained as a physician and biostatistician, the principal author was familiar with the assumptions and limitations underlying linear classification and prediction models. In the late 1980’s, he and Dwight Egbert successfully used artificial neural networks (ANNs) to extract information from complex and noisy medical images. At that time, the ANN literature was generated predominantly by engineers, computer scientists, and mathematicians. The actual data analyzed in these papers were often synthetic and simple. Almost never were measures of statistical confidence or generalizability reported, nor were comparisons made with traditional linear models.

Were ANNs– simplified versions of distributed pattern recognition in the brain– a match for the health care database challenge?

NevProp reflects the effort of the faculty and students who accepted that challenge. Here are the three basic questions we set out to address:

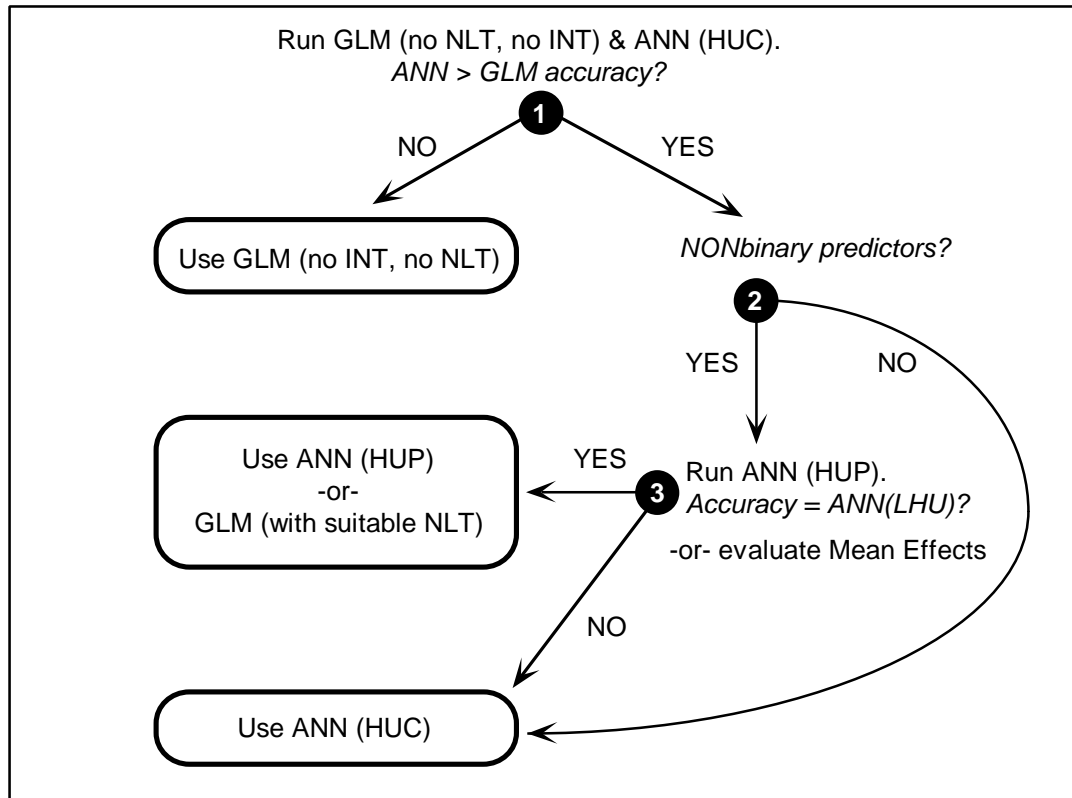
1. Statistically speaking, exactly what *is* an artificial neural network (ANN)?
2. Can ANN theory and architecture be refined sufficiently to be useful in the statistical analysis of large databases?
3. How would such ANNs fare in head-to-head competition with linear models?

Although this manual is not intended as a treatise on the subject, the first issue is addressed in Chapter 2. Chapters 3-6 reflect our (affirmative) answer to the second question.

Regarding the third question: perhaps most unexpected was its evolution. The most basic ANN *is* a generalized linear model, as described in Chapter 2. More complex ANNs, if properly configured, will not be any *less* accurate than the linear model, but can simultaneously and automatically capture important nonlinearities (in the predictors) and interactions (of the effects of the predictors). With this increment in performance comes computational intensity, and difficulty in drawing inference.

So the third question didn't really make sense. We rephrased it as:

3. When, and how, should nonlinear ANNs be used *alongside* traditional linear models to improve the *overall* information and knowledge gleaned from a database?



Abbreviations: GLM, generalized linear model (e.g., multiple linear regression, logistic regression):
 ...NLT, nonlinear transformations of predictors (e.g., spline, polynomial)
 ...INT, interactions terms involving predictors
[can also be accomplished using ANN without hidden units]
 ANN, artificial neural network with adaptive regularization:
 ...HUC, hidden units common to all predictors
 ...HUP, several hidden units from each predictor, none in common
 ...Mean Effects: provided in NevProp3 module

Figure 1. A three-step approach for incorporating artificial neural networks into biostatistical regression modeling on observational data sets.

Our answer is summarized in Figure 1. It is our experience, to date, that many clinical databases have *no* important nonlinearities and interactions. Therefore, comparing the predictive performance of a (properly configured) single nonlinear NevProp3 ANN to the corresponding linear model may save a tremendous amount of time: if there is no significant improvement in bias-corrected generalization accuracy, then not only is there no need for further ANN explorations, but there is no need for the investigator to perform the many arduous (and statistically hazardous) *multiple comparisons* of linear models coded with various transformations and interaction terms. That is, the marriage of the techniques will often result in a savings of time (equals money) and a better model of “truth.”

On the other hand, when the nonlinear NevProp3 ANN *does* significantly outperform the corresponding linear model, use of the ANN model may again save time through a hierarchical strategy:

First train a simplified NevProp3 ANN model that is linear in the parameters (no hidden units common to the predictors), but provides nonlinearity in the nondichotomous predictors— this

fits quickly. If the predictive performance is the same as that of the full ANN, the investigator should stay with a linear effects model, and focus on appropriate transformations of the predictors (e.g., geometric, logarithmic, piecewise, restricted cubic splines). If the form of the transformations are not meaningful, the simplified ANN itself can be used. No need to account for interactions.

On the other hand, if the full ANN *did* outperform the simplified ANN model above, the investigator should consider accepting the ANN over a modified linear model. This is especially the case if the main objective is just to create a reliable prediction tool (“black box”). Again, time and multiple comparisons of linear models are minimized. Additionally, with sufficient computational power, the NevProp3 ANN model can be used to select important variables, and statistically estimating their effects. This contrasts with the traditional approach of stepwise selection across many linear models, which is notoriously unreliable in extracting a subset of variables that will generalize on future data. Missing elements can even be allowed to remain in the NevProp3 ANN model as it is trained to predict.

Alternatively, the hierarchical strategy above can be replaced by the NevProp3 mean effects module. This module provides, among other things, information about the distribution of effects over the range of nondichotomous predictors.

ANNs can also be used to adaptively weight, or “gate” the predictions of multiple (human or statistical-model) experts. The gating ANN takes as its input the original training cases, and its output is a vector of weights that multiply the predictions made by each of the experts. Rather than simply averaging, or taking a fixed (linear regression) weighting of each expert’s predictions, the gating ANN assigns a vector of weights that differs *for each case*. The rationale is that the experts may differ in their expertise as a function of the kind of case presented (technically, as a function of the case’s location in data space). For example, consider a dataset comprised of information concerning patients undergoing open-heart surgery. One expert might be a cardiologist, another a heart surgeon. The cardiologist knows a lot about the diagnosis, management, and chronic course of heart disease. The surgeon knows a lot about acute decompensation and physiological recovery of the heart. Each physician is given a file containing the medical history, examination, and laboratory tests of patients admitted to the hospital for surgery. Each physician is asked to make a prediction about the probability that each patient will die during or soon after surgery. Let’s also obtain “expert” predictions from a linear regression model, an ANN, and a decision-tree program. So we now have 5 “experts” – 2 human, 3 computational. How shall we best combine their predictions? After all, future patients, health care providers, and insurers will want the best prospective estimate of prognosis. The gating ANN achieves this goal.

In summary, *properly configured* nonlinear ANNs can be recommended for use alongside linear methods in the statistical toolbox. At this time, we know of no ANN package other than NevProp3 that offers the features necessary for the statistical strategies described above. Use of many existing ANN packages (including a macro available for a popular commercial biostatistics program) will lead to disastrous results if applied in the strategies above, due to overfitting and the lack of effect and confidence estimates. In the future, we hope to see NevProp3-like algorithms incorporated as standard components of the most popular commercial statistical packages. We welcome your comments, suggestions, and word of progress with NevProp3.

&1.2 Features

NevProp3 is actually three programs sharing the same core modules. The basic mode of operation is a “generalized nonlinear (regression) model,” or GNLM. It is what most investigators are familiar with (or not so familiar with) as an “artificial neural network.” GNLM mode is used to make and validate predictions, estimate effects and confidence limits, and select important variables.

The second mode of operation might be called the “artificial neural network-version of a k-nearest neighbor classifier,” or ANN-kNN for short. This might seem contradictory, because the traditional kNN classifier operates solely in data space, whereas the ANN optimizes in parameter (weight) space. Whereas a kNN classifier stores all individual data points (cases) to make future predictions, the neural net stores only weights used to recreate a smooth prediction function. But if the density (crowding) of data varies smoothly across data space, there will be regions of relatively higher and lower densities. Theoretically, then, some shorthand notation could be used to describe the patterns of clustering rather than storing every data point in memory— this notation is provided by the nonlinear coding of the ANN-kNN in NevProp3. This has two applications: first, as a compact model of a full kNN classifier; and second, as a way to use cases with missing data elements (for developing a model, and/or for predicting on future, incomplete cases). This works because the final ANN-kNN model is an approximation of the data distribution. So, if you are missing some element(s) of a new case, you use the known elements to determine where the values of the unknown *would* have had their maximum probability of lying. The neat part of this model is that the output (say, the probability of death) can be included during the optimization of the ANN-kNN model, so that you never really need to know what values the program found for the missing elements. After all, you really just wanted to make a prediction. This concept is based on expectation-maximization theory, and is the first full ANN implementation of which this author is aware. And, at least in this author's opinion, may mirror the real mechanism of biological memory in the human neocortex.

The third mode of operation is the use of an ANN model to adaptively weight, or “gate” the predictions of multiple (human or statistical-model) experts. This ANN-Gated Ensemble (or AGE) model takes as its input the original training cases, and its output is a vector of weights that multiply the vector of predictions made by each of several experts. Rather than simply averaging, or taking a fixed (linear regression) weighting of each expert's predictions, the AGE assigns a different vector of weights *for each case*. The rationale is that the experts may differ in their expertise as a function of the kind of case presented (technically, as a function of the case's location in data space). From a Bayesian perspective, it facilitates predictive inference by marginalizing over multiple models— none of which can be assumed to be *the* correct model. AGE may be useful for maximizing prediction performance in critical operations in health care, engineering, finance, and politics— circumstances where many (human and statistical-model) “experts” have something to offer, and may even perform similarly on average— but the investigator doesn't know, a priori, which expert should be relied on for extrapolation to new data.

The following list of features is intended to be sufficiently detailed to permit experienced analysts to determine whether NevProp3 would be useful in their research environment.

MAJOR FEATURES OF NevProp3 OPERATION

(* indicates feature new in version 3)

1. Character-based interface common to the UNIX, DOS, and Macintosh platforms.

2. Command-line argument format to efficiently initiate NevProp3. For Generalized Nonlinear Modeling (GNLM) mode, beginners may opt to use an interactive interface. Other modes include ANN-kNN* (k-nearest neighbor) emulation and AGE* (ANN-Gated Ensemble/committee) modes.
3. Interactive (control-c) interrupt to change training settings on-the-fly, from either command-line or fully-interactive interfaces.
4. Flexible network control file, allowing arbitrary sequencing and multiple formats for responses, and #-- or /*--*/ user-commenting formats.
5. Automatic or general defaults are provide for all operational settings.* The most basic network file need only contain the data (or a pointer to an external data file), and the number of input (independent) and output (dependent) variables.
6. Automatic default configuration as a fully-interconnected 3-layer network.*
7. Options to save a file containing a log of time-stamped progress and results, weights (parameters), imputed datasets, and training and testing (validation) predictions.
8. Training data placed either within the network control file within an external file. Allowed in training and testing data sets are: an initial alpha-numeric column for case labeling; any number of user-specified headers (e.g., to describe the dataset and to label the variables); and #-- or paired /*--*/ user-commenting formats.
9. Selection of a subset of arbitrarily-sequenced variables in a dataset as inputs and outputs.*
10. Option to pre-randomize the training data.
11. Option to pre-standardize the training data (z-score or forced range*).
12. Option to pre-impute missing elements in training data (case-wise deletion, or imputation with mean, median, random selection, or k-nearest neighbor).*
13. Upon NevProp3 startup, an informative initial summary of active settings (which may be copied and pasted as the basis of a new network control file).
14. Option to display descriptive statistics on all input and output variables (mean, median, standard deviation, highest and lowest five values, and table of percentiles).*
15. User-specified interval reporting of common error measures on training (and optional held-out train subset).
16. Primary error (criterion) measures include mean square error, hyperbolic tangent error, and log likelihood (cross-entropy), as penalized an unpenalized values.
17. Secondary measures include ROC-curve area (c-index), thresholded classification, R-squared and Nagelkerke R-squared. Also reported at intervals are the weight configuration, and the sum of square weights.
18. Allows simultaneous use of logistic (for dichotomous outputs) and linear output activation functions (automatically detected to assign activation and error function).*
19. 1-of-N (Softmax)* and M-of-N options for binary classification.
20. Optimization options: flexible learning rate (fixed global adaptive, weight-specific, quickprop), split learn rate (inversely proportional to number of incoming connections), stochastic (case-wise updating), sigmoidprime offset (to prevent locking at logistic tails).
21. Regularization options: fixed weight decay, optional decay on bias weights, Bayesian hyperpenalty* (partial and full Automatic Relevance Determination– also used to select important predictors), automated early stopping (full dataset stopping based on multiple subset cross-validations) by error criterion.

22. Validation options: upload held-out validation test set; select subset of outputs for joint summary statistics;* select automated bootstrapped modeling to correct optimistically biased summary statistics (with standard deviations) without use of hold-out.
23. Saving predictions: for training data and uploaded validation test set, save file with identifiers, true targets, predictions, and (if bootstrapped models selected) lower and upper 95% confidence limits* for each prediction.
24. Inference options: determination of the mean predictor effects and level effects (for multilevel predictor variables); confidence limits within main model or across bootstrapped models.*
25. ANN-kNN (k-nearest neighbor) emulation mode options: impute missing data elements and save to new data file; classify test data (with or without missing elements) using ANN-kNN model trained on data with or without missing elements (complete ANN-based expectation maximization).*
26. AGE (ANN-Gated Ensemble) options: adaptively weight predictions (any scale of scores) obtained from multiple (human or computational) “experts”; validate on new prediction sets; optional internal prior-probability expert.*

&1.3 Technical Support

As volunteers, our resources for providing technical support are quite limited. In general, we can only hope to respond to inquiries placed via Internet E-mail.

If the program will not compile or initialize, please check with a local computer resource person before emailing the author (to be sure that the problem resides with NevProp3 per se). To report a bug, please see section 1.4, below.

We cannot help with the design or analysis of models. In the Appendices are listed useful published and on-line resources.

Questions and comments regarding the **quickprop** algorithm itself should be directed to Scott Fahlman <sef+@cs.cmu.edu>. Those regarding the theory and development of Bayesian regularization (**ARD**) should contact David MacKay <mackay@ras.phy.cam.ac.uk>.

Thank you. *Phil Goodman* <goodman@unr.edu>

&1.4 Bug Reports

Before reporting bugs to the author (above), please check the UNR machine (section 3.1) for the file “nevprop.bug” for the existence of known bugs, fixes, and upgrade plans. Thank you :)

CHAPTER 2. PREDICTION & KNOWLEDGE:

The role of NEURAL NETS as STATISTICAL TOOLS

&2.1 What is Knowledge?

*“Problems may be solved in the study which have baffled all those who have sought a solution by the aid of their senses. To carry the art, however, to its highest pitch, it is necessary that the reasoner should be able to utilise all the facts which have come to his **knowledge**, and this in itself implies, as you will readily see, a possession of all **knowledge**, which, even in these days of free education and encyclopedias, is a somewhat rare accomplishment.”*

Sherlock Holmes– THE FIVE ORANGE PIPS
Doyle, Sir Arthur Conan (1859-1930), *The Strand Magazine*, 1891.

As so aptly phrased by Sir Arthur Conan Doyle, it seems obvious that facts must derive from nature, both to generate and refute hypotheses. But it was really not until the seventeenth century that Francis Bacon introduced what we now call the “scientific method.” During the preceding 1600 years, “knowledge” had been confined to metaphysical argument and purely logical reflection on Greek and Roman classics. It was, therefore, a major paradigm shift for Bacon to propose, in his *Advancement of Learning*, “This is the foundation of all, for we are not to imagine or suppose, but to discover, what nature does or may be made to do.”

Presumably, you are interested in gleaning knowledge about the real world by applying advanced statistical methods to observed data. But what exactly *is* knowledge? Here is this author’s attempt at its definition and classification:

Knowledge is information derived from the combination of experience and reasoning. The scientific method requires that the infrastructure of our knowledge be aggregated through direct observation of nature (also known as *a posteriori* knowledge). We adorn this infrastructure with interpolations and extensions by the process of reasoning (*a priori* knowledge). Ideally, we attempt to refute the resultant hypothetical adornments by further empiric observations. And so the cycle continues. (2.1a)

Knowledge can also be classified from another perspective– according to *why* we seek it:

To accomplish a defined task (practical, or immediate predictive knowledge). (2.1b)

To know principles for the sake of knowing (generalizable knowledge). (2.1c)

Practical knowledge (2.1b) is used to achieve an observable outcome under very specific circumstances. We might say that practical knowledge leads to a model that can be confirmed by observation. Detailed measurements are input to the model, from which certain testable predictions are made. In this way, *practical knowledge leads to technological innovation and engineering accomplishments.*

On the other hand, generalizable knowledge (2.1c) is extracted in the form of general principles, laws, rules, etc. Generalizable knowledge may be very concrete– such as the discovery of an

underlying statistical distribution that “generalizes” successfully on future data (i.e., using the same set of predictor variables). But generalizable knowledge is usually more compressed or abstract, stripped of the less relevant predictors. In other words, an attempt is made to retain only the essential “effects” in the rules. Generalizable knowledge may be useful to reason about underlying physical processes, or for hypothesis-formulation in the design of new experiments. Additionally, discovering generalizable knowledge makes us “happy.” For some biological reason, we humans continually strive to understand our history, our world, and ourselves. It is the accumulation of *generalizable knowledge that is most central to the advancement of science*.

Of course, generalizable knowledge may ultimately find application in future, albeit presently unanticipated circumstances (that is, wherein some predictors have changed). *This is the translation of science into technology* (or, equivalently, generalizable into practical knowledge). Indeed, experienced investigators are acutely aware of the need to convince funding organizations of the (potential) practical applications of scientific research they propose.

In summary, knowledge may exist as generalizable *a priori* laws of science, or as practical *a posteriori* prediction & technology. The process of transforming data into knowledge is the drawing of statistical *inference*., reviewed in the next chapter.

&2.2 Statistical Methods

2.2.1 What are “Statistical Methods”?

It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts.

Sherlock Holmes– A SCANDAL IN BOHEMIA
Doyle, Sir Arthur Conan (1859-1930), *The Strand Magazine*, 1891.

“Statistics” refers both to the actual collection, analysis, and presentation of data, and to the general discipline concerned with such methods. Most generally, then, “statistical methods” are those techniques used to collect data sets, to extract relationships, and extrapolate (*draw inference*) from information contained in the data. What establishes statistics as a discipline is that similar conceptual techniques are needed for data sets in every field of application—wherever knowledge is sought and applied.

Underlying statistical thinking is the concept that a given data set is only one of many possible such sets that *could* be collected (presently or in the future). Measurements on a given sample will vary depending upon what, when, and how the data is collected. Recognizing that repeated sampling from the total eligible, or *target population*, will demonstrate variability, these measured features are referred to as *variables*. Therefore, the analysis and interpretation of a variables depends not only on the given sample, but on what is assumed about the possible alternative observations that could have been obtained instead.

A statistical *model* is used to describe the relationship among the variables, attributing any remaining variation to random fluctuations, or “noise” (which we hope is not relevant to the problem at hand). The model is usually expressed as a mathematical function, whose actual “shape” is controlled by parameters whose values must be determined (fit). Parameters are not directly observed, but serve to summarize the relationships among observed variables. Because

there are generally fewer parameters than variables and cases, parameters restrict the ways in which variables may interrelate. The reason for such constraint will be discussed later.

Statistical *inference*, then, is the process of using these models to extrapolate to a larger universe of data, of which we observe only the dataset at hand. Two global levels of inference may be addressed, depending on real-world problem to be solved:

COMPARATIVE INFERENCE. How well do competing models explain the observations?
This is especially clear when the models conform to physical scientific models. (2.2.1.1)

PARAMETRIC & PREDICTIVE INFERENCE. Given an accepted model formulation, what are reasonable values that the parameters should take (*parametric inference*)? What predictions may safely be drawn when this model is presented with new observations (*predictive inference*)? (2.2.1.2)

In terms of the framework presented in section 2.1, generalizable knowledge is gleaned from both model comparison and parametric inference. Practical knowledge results from the application of predictive inference.

In the way of a concrete example, say we collect data on surgical patients to generate a statistical model about the relationship of pre-operatively measured variables to the post-operative probability of death (mortality). We'd like to apply this model to predict mortality on new future patients admitted to the hospital— clearly a meritorious use of predictive inference. Although this type of extrapolation is commonly referred to as “generalization,” it must be taken in the narrow, statistical-distributional-sampling sense of the word. A purely predictive model is a black box, presenting no true “generalizable” knowledge to the scientist.

Disease, diagnosis and treatment are dynamic entities. For instance, it is unlikely that patients studied today will be representative of patients undergoing surgery in 10 years. Statistically speaking, the distribution of data will change— some features will be the same, some modified, and some related to new or unanticipated effects of changing disease patterns or new drugs and technology. Can we make parametric inference to glean generalizable knowledge from our current model, that might facilitate future needs?

Because statistical models describe relationships among variables, the most basic parametric inference we can squeeze out is *important* pairwise association: when one variable is altered, what is the *effect* on the other? (In doing so, we may be ignoring important interactions among the effects of other explanatory variables.) Here, “important” means that the association truly reflects a relationship in the target population. That is, we'd like to infer *effects* in the target population using measurements from a finite sample. Clearly, this type of knowledge— knowledge of effects— corresponds to the concept of generalizable knowledge. Note that the concept of statistical “effect” is mathematical way to characterize an association, and does not necessarily connote a cause-effect relationship among entities in the real world.

Returning to our example, assume that our statistical model included 50 predictive pre-operative variables, and that the parameters fit to the model show a statistical effect of diabetes significantly increasing surgical mortality. In ten years, we would be very hesitant to plug new patient profiles the same 50-variable model. But if our new patient is diabetic, we will still consider the main effect, or rule: such patients are at higher risk until proven otherwise. Therefore, in future patient care, and in the design of a future studies, we would likely consider

the role of diabetes. Additionally, characterizing the bivariate diabetes -> death effect makes us feel comfortable that we understand something more about the underlying pathophysiology.

2.2.2 Frequentist vs Bayesian Statistical Analysis

As described above, important interrelationships among real-world variables may be summarized by the specification of statistical models. The actual process of fitting parameters to a statistical model using existing data may be approached using either “frequentist” analysis or “Bayesian” analysis.

Frequentist analysis seeks the single, most likely set of parameters that satisfy the constraints of a model (maximum likelihood fit). This process is most generally referred to as *regression* (section 2.2.3). While regression may be computationally frugal, the need to determine the statistical significance of these results involves a retrospective evaluation of the estimation procedure over the distribution of possible observations that could have been observed assuming a “true” but unobserved set of parameters. The interpretation of resulting p values or confidence intervals is, therefore, not straightforward.

In particular, for flexible regression models, like nonlinear artificial neural networks (ANNs), it is the model’s *predictions* and the *net effects of predictors* that are of intrinsic interest, not the architecturally- and initialization-sensitive values of the many individual parameters. Therefore, frequentist assumptions are suspect, and the generation of p values or confidence intervals for individual parameters is problematic.

An alternative, “Bayesian” approach is to base inference on parameters and predictions directly on the observed data— no single correct solution to the model is assumed. Models are formulated entirely in probabilistic terms, which facilitates the interpretation of confidence boundaries. The objective is to properly weight competing models, and, within each model, to integrate over a *distribution* of parameters (rather than a single most-likely solution). Weaknesses of Bayesian analysis include (often extreme) computational intensity, and a need to make assumptions (subjectively but explicitly) about the forms of parametric distributions.

In some knowledge domains, scientists can formulate very specific mathematical models for the processes they wish to better understand. For instance, drug pharmacokinetics and distribution in multiple body compartments based on well-characterized tissue affinities and metabolic properties. Or, nerve cell signal processing based on accepted approximations of cable theory and synaptic physiology. Although both of these examples comprise nonlinear models, the relationships among their predictors are explicit, and inference follows readily after maximum likelihood optimization. Such inference can be used to test the scientific hypotheses that motivated the models and data collection. Frequentist methods are most applicable here.

In other knowledge domains (especially with complex, but poorly understood relationships among variables) explicit or analytic formulations of parametrized models are simply not available. Modern computation power has promoted the emergence of several approaches to the description of these relationships, including ANNs (especially multilayer perceptrons) and artificial evolutionary programs (especially genetic algorithms). Both methods emulate “tricks” nature uses to search for good solutions.

ANNs are discussed later in this chapter and in the rest of this manual. It should be pointed out that while ANNs are usually implemented as regression models (i.e., using maximum likelihood

optimization to fit parameters), the sensitivity of parametric fit to initialization and connectivity is reminiscent of Bayesian concerns that distributions of parameters be considered. While direct application of Bayesian methods to ANNs is theoretically and computationally complex, such methods can be hybridized with maximum likelihood methods to (potentially) improve ANN inference (see Chapter 6 of this manual).

2.2.3 What is “Regression”?

From a drop of water a logician could predict an Atlantic or a Niagara.

Sherlock Holmes— A STUDY IN SCARLET
Doyle, Sir Arthur Conan (1859-1930)

A regression model is a function that describes a dependent relation (say, R) among measurements of quantitative variables. From a frequentist perspective (section 2.2.2), if we applied the “correct” (but unknown) regression relation to a new set of cases, we could make precise predictions about the response variables, without having to actually measure them. Let $\mathbf{Y} \equiv \{y_1, y_2, \text{etc}\}$ be a vector list of such predicted variables (*synonyms*: response, dependent, or output variables), and $\mathbf{X} \equiv \{1, x_1, x_2, \text{etc.}\}$ be a vector list of the predictors (*synonyms*: covariates, covariables, or the independent, input, explanatory or descriptor variables)¹ used in a regression function, $g(\mathbf{X})$. Mathematically, then, we may describe a regression relation as:

$$R(\mathbf{Y}|\mathbf{X}) = g(\mathbf{X}). \quad (2.2.3.1)$$

As discussed earlier, knowing the relationships among a set of variables is useful in at least 2 ways: making predictions (predictive inference or estimation), and gaining insight (parametric inference) about the natural relation. Scientists increase the latter (generalizable knowledge) by modeling and drawing inference about the physical world. When a scientific hypothesis about a physical process is expressed mathematically as a conditional prediction, it can be modeled as a regression. If that formula involves a relatively simple weighted combination of the predictors, and the model generalizes well, we are inclined to interpret the weights (parameters) to have physical meaning (see section 3, below). If the model does not generalize, then we may use this as evidence to refute our scientific hypothesis. In short, we wish to make valid statistical inferences about the regression function’s parameters, in order to support or reject scientific inferences about physical processes. We’d also like to make practical use of the predictive inference.

For example, let \mathbf{X} be results of medical tests performed on a patient presenting to an emergency room with chest pain, and \mathbf{Y} represent the presence of a heart attack, which cannot otherwise be confirmed for many hours. If a heart attack is actually occurring now, we want to immediately administer medication to reduce damage to the heart. However, this medication may carry major risks, so we don’t want to use it on all patients with chest pain. If we’ve made measurements of \mathbf{X} and \mathbf{Y} on many such patients in the past, and had some way to create the “true” regression relation, R , we should now be able to estimate the probability of \mathbf{Y} (heart attack) given knowledge limited to \mathbf{X} (clinical findings). Here, making accurate predictions may save lives. Of course, in the real world, we know there is always variability in our measurements, so we can’t generate the “true” relation; but we can estimate the accuracy of our predictions to determine if R was, in fact, a good approximation of the true relation. That is, does R accurately generalize, in the statistical sense, to patient data not used to construct R ?

Simple regression models often suffice to characterize basic physical processes. But even basic hypotheses may require complicated mathematical descriptions to truthfully reflect nature. For example, the linear Newtonian relation between velocity and time breaks down at high relative velocities— although it doesn't matter in the activities of daily human living. However, in some scientific applications, Einstein's model of special relativity better approximates the correct relation.

Complex phenomenon comprise multiple underlying physical influences. In general, making accurate predictions and inferences about complex processes requires that the corresponding regression model be complex. The mathematical form of these complexities may include nonlinear transformations of the variables (predictor nonlinearity), and/or nonlinearity in the way the parameters affect the predictors (effect nonlinearity).

Below, we review the family of linear models, followed by a family of very flexible nonlinear models called artificial neural networks.

&2.3 Generalized Linear Models

If the expected response of a regression function involves only a function of the weighted sum of the components of \mathbf{X} , that is,

$$R^{GLM}(\mathbf{Y}|\mathbf{X}) = g(\mathbf{X}\mathbf{b}), \quad (2.3a)$$

where $\mathbf{b} = \{\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2, \text{etc.}\}$ is a vector list of weights (parameters), we call this the *generalized linear model* (GLM)². A more precise name would be the *linearizable* model, but that's a bit more difficult to pronounce. The problem then reduces to finding (estimating, fitting) the appropriate values for the weights, \mathbf{b} , after linearizing the function g . The class of GLMs includes linear regression ($g(\mathbf{X}\mathbf{b}) = \mathbf{X}\mathbf{b}$), analysis of variance, logit, probit, loglinear, multinomial response, and commonly used survival and time series models. For each, a corresponding assumption is made concerning the distribution of error about the expectation.

In GLM regression, we select a linearizing function from the exponential family, according to the distribution of data and theory about how the model would explain the random component (i.e., the part not explained by \mathbf{Y} .) The exponential family includes the normal, logit, probit, multinomial, Poisson, and other distributions.

For example, in ordinary linear regression, we are interested in a relation,

$$\mathbf{Y} = \mathbf{X}\mathbf{b} + \mathbf{e} = \sum_i \mathbf{b}_i x_i + \mathbf{e} \quad (2.3b)$$

with a mean, or expectation,

$$R^{Linear}(\mathbf{Y} | \mathbf{X}) \equiv E(\mathbf{Y} | \mathbf{X}) = \mathbf{X} \quad , \quad (2.3c)$$

assuming that, given \mathbf{X} , \mathbf{Y} has a normally distributed error, \mathbf{e} , with mean \mathbf{Xb} and constant variance s^2 . The parameters of an ordinary linear regression model may be estimated using the closed-form method of least square minimization.

In the binary response *logistic regression model* (LRM), we are interested in the probability of the occurrence of a binary {0,1} response event, Y — that is, the relation $R(\mathbf{Y}|\mathbf{X}) \equiv \text{Prob}\{Y = 1 | \mathbf{X}\}$. LRM uses the *logistic function* (Figure 1) for $g(\mathbf{X})$, so that we have

$$R^{LRM}(\mathbf{Y}|\mathbf{X}) \equiv \text{Prob}\{Y = 1 | \mathbf{X}\} = \frac{1}{1 + \exp(-\mathbf{Xb})}. \quad (2.3d)$$

The logistic function is a simple sigmoid centered at a value of 0.5, with a lower asymptotic limit of zero and an upper limit of one. This function is just one of many that could be used to constrain probabilities to lie within the 0-1 interval.

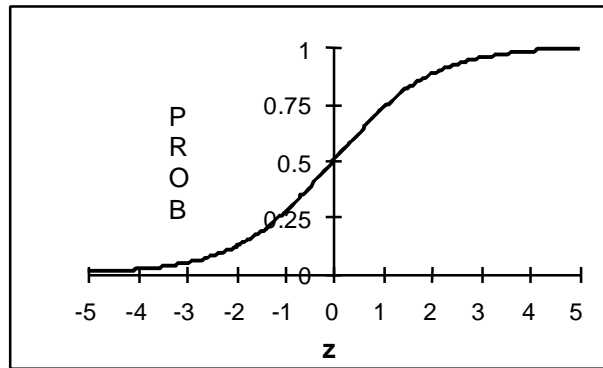


Figure 2.3.1. Probability of z expressed as the logistic.

To simplify the estimation of the weights, we work directly with the linear predictor, \mathbf{Xb} . In general, a function that restores the linear predictor is called a *link function*. In our notation, g is the *inverse* of the link function. In LRM, the link function is the log odds, or “logit,” where

$$\text{Odds} \equiv \frac{\text{Prob}}{1 - \text{Prob}}, \quad (2.3d)$$

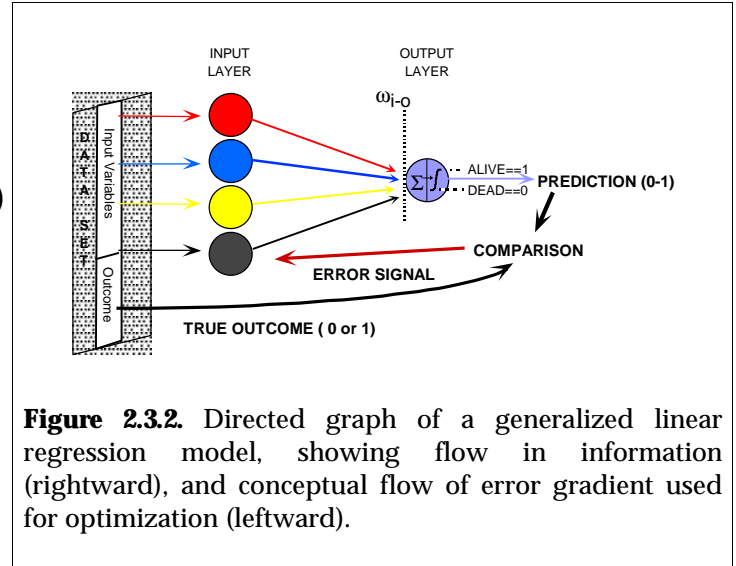
and the natural log of the odds, or *logit*, is,

$$\text{logit} \equiv \log \left(\frac{\text{Prob}\{Y = 1 \mid \mathbf{X}\}}{1 - \text{Prob}\{Y = 1 \mid \mathbf{X}\}} \right) \quad (2.3e)$$

$$= \log \left\{ \frac{\frac{1}{1 + \exp(-\mathbf{X}\mathbf{b})}}{1 - \left(\frac{1}{1 + \exp(-\mathbf{X}\mathbf{b})} \right)} \right\} = \log(\exp(\mathbf{X}\mathbf{b}))$$

$$= \mathbf{X}\mathbf{b} \equiv \mathbf{b}_0 + \sum_i^{\text{\# input units}} \mathbf{b}_i^{IF(\text{Path}^i)} x_i \quad (2.3f)$$

where b_0 is the usual regression constant, or bias, weight multiplying an assumed fixed value of $x_0 = 1$, and the summation is the b_i -weighted average over all predictors x_i .



Having recovered the linear portion using the logit link, we can now perform linear regression to estimate the weights, \mathbf{b} . Unlike ordinary linear regression, however, logit link function residuals have no closed-form solution. The parameters of these models must be estimated iteratively, usually under maximum likelihood assumptions.

It will be shown in later sections that using the logit link for a binary response regression function not only simplifies estimation of the weights, but also facilitates the interpretation of the effects of the predictors.

&2.4 General Nonlinear and Artificial Neural Network Models

As in GLMs, the functional form of the response of a nonlinear model should be consistent with the type of prediction, and the distribution of errors about predictions. For a binary probability model, the same considerations hold as in the previous discussion of the LRM. Therefore a logistic inverse link is also appropriate in the nonlinear case. Computation of the argument to g will be more complex, and, by definition, not linearizable.

In some fields, the mathematical formulation of the nonlinear model follows underlying theory or previously observed knowledge. Examples include the kinetics of drug distribution between body compartments, and the conduction of an action potential along segments of a neuron. The parameters estimated by this “analytic” nonlinear regression are directly interpretable by the scientist.

However, when modeling a poorly understood or otherwise complex system the scientist does not have explicit knowledge about the (usually multiple) factors affecting a response. It is desirable, then, to have a self-organizing mathematical formulation that could take on a range of multivariate functional forms. Provided that the model is not overfit, the parametric inference

could provide insight into underlying processes (section 6, below), and the predictive inference may have technologic application.

We now limit our focus to probabilistic regression, most often used for classification. Although we restrict our attention to a single binary response, as in the LRM (3.4), extension follows naturally in both GLMs and ANNs to ordered and multinomial polytomous classification.

In binary classification, we desire a binomial error distribution. The expected response for a binary-response ANN is therefore similar to the logistic GLM (3.4), but g is more complex. We introduce first the concept of a “layer” of functional units, implying that units from one layer connect to units in the next, hierarchically, without shortcuts. The most simple logistic ANN (LANN) is the fully interconnected, feed forward single hidden layer (SHL) model (figure 2.4.1), with relation

$$R^{LANN}(Y|X) \equiv \text{Prob}\{Y = 1 \mid X\}$$

$$= \frac{1}{1 + \exp \left[- \left[\mathbf{b}_0 + \sum_{j=1}^{\text{\#hidden units}} \mathbf{b}_j \frac{1}{1 + \exp \left[- \left[\sum_i^{\text{\#input units}} \mathbf{b}_i^{IF(Pathij)} x_i \right] \right]} \right] \right]} \quad (2.4a)$$

where the exponential argument is the \mathbf{b}_j -weighted average over all intervening logistic units. The input to each intervening unit is the sum of \mathbf{b} -weighted predictors, x_i . We can see that the GLM's “ \mathbf{Xb} ” term is replaced by a sum of LRM terms, each with its own \mathbf{Xb} . We call the intermediate logistic functions the “hidden units,” in analogy to interneurons of the brain.

More compactly, we can express the SHL-ANN relation by its logit transformation (2.3f),

$$\text{Logit}^{\text{SHL-LANN}} = \mathbf{b}_0 + \sum_{j=1}^{\text{\#hidden units}} \mathbf{b}_j \frac{1}{1 + \exp \left[- \left[\sum_i^{\text{\#input units}} \mathbf{b}_i x_i \right] \right]} \quad (2.4b)$$

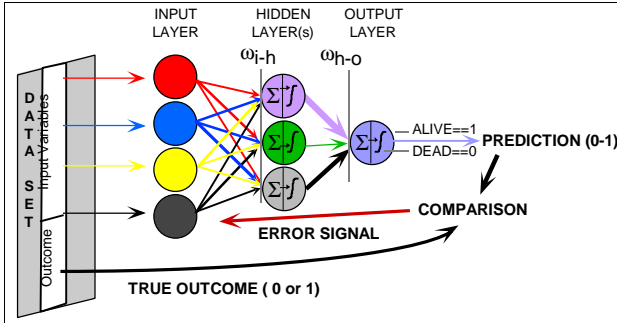


Figure 2.4.1. Directed graph of feed forward single hidden layer artificial neural network model, showing flow in information (rightward), and conceptual flow of error gradient used for optimization (leftward).

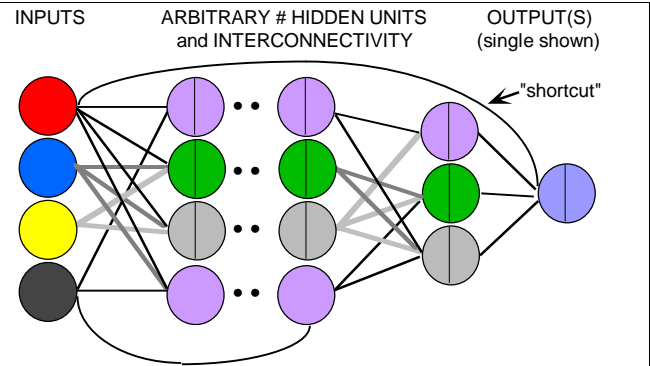


Figure 2.4.2. Directed graph of an arbitrarily interconnected network regression model. There is no requirement that hierarchical “layers” exist.

In general, there is no reason to require that connectionism to be complete, nor that only a single hidden unit intervene along each path from input to output. Most generally, we have an “unlayered,” arbitrarily interconnected ANN (figure 2.4.2). At each hidden unit, a logistic transformation is carried out on the sum of its inputs. This transformed value may then be carried forward to other hidden units until the final output (response) function is computed. If the output is to be a logistic probability response, we can again express it as the logit,

$$\text{Logit}_j^{\text{Output}} = \left(b_0 + \sum_{i=1}^{\# \text{input units}} b_i^{IF(\text{Path}_{ik})} x_i + \sum_{j=1}^{\# \text{hidden units}} b_j^{IF(\text{Path}_{jk})} \frac{1}{1 + \exp(\text{Logit}_j^{\text{Hidden}})} \right)$$

$$= \frac{1}{1 + \exp \left[- \left(b_0 + \sum_{i=1}^{\# \text{input units}} b_i^{IF(\text{Path}_{ik})} x_i + \sum_{j=1}^{\# \text{hidden units}} b_j^{IF(\text{Path}_{jk})} \frac{1}{1 + \exp(\text{RECURSION})} \right) \right]}, \quad (2.4c)$$

where the second term summates the b_i -weighted values from input units directly connected (“shortcutting”) to the output, and the third term summates the b_j -weighted values from any hidden unit connected to the output unit. The superscript “IF” tests for the existence of a connection. Notice that this formula is recursive— the exponential argument is the same formula applied to each hidden unit, as if it were the output unit of an intervening ANN. Although figure 2.3.2 is a more intuitive portrayal of the unlayered ANN, we need further mathematical machinery to compute the overall effect of each predictor (below).

&2.5 Drawing Inference in GLMs

First, consider a GLM with a single continuously-valued response variable, y , and 2 predictors, $\mathbf{X} \equiv \{1, x_1, x_2\}$. Then the predicted value of y , from the final model fit with $\mathbf{b} = \{\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2\}$, may be expressed as

$$R_{GLM}(\mathbf{Y}|\mathbf{X}) \equiv g(\mathbf{X}\mathbf{b}) = g(\mathbf{b}_0 + \mathbf{b}_1 x_1 + \mathbf{b}_2 x_2) . \quad (2.5a)$$

After we estimate the weights, the linearity makes it easy to describe the predictive relationship (draw parametric inference).

In regression modeling, we call the change in y with respect to x the *effect* of x_1 ; here,

$$\begin{aligned} \text{effect of } x_1 &\equiv \frac{\partial y}{\partial x_1} \\ &= \frac{\partial (\mathbf{b}_0 + \mathbf{b}_1 x_1 + \mathbf{b}_2 x_2)}{\partial x_1} \\ &= \mathbf{b}_1 . \end{aligned} \quad (2.5b)$$

Assuming linear weighting and no interaction among the predictors, we see that the change in y resulting from a unit change in x_1 , for any case, is just the fitted weight for that predictor, adjusting for the other predictors. For example, using a simple linear model, we might wish to predict the length of a patient's hospital stay (in days), conditioned on age (in years), and gender (coded, say, as male=0 and female=1). Based on (5.2), the effect of age is \mathbf{b}_1 , the change in length of stay for each year increase in age, and the effect of gender is just \mathbf{b}_2 , the change in length of stay for females versus males. Under additional statistical assumptions, we could estimate confidence intervals or test hypotheses about the significance of these effects.

The preceding formulation of the effect is, technically correct only for a continuous predictor, because the derivative of a noncontinuous variable is undefined. But the regression framework implicitly assumes all variables are differentiable, and we usually encounter no problems with interpretation of the effects themselves or transformations of the effects (like the odds ratio). However, inaccuracy can be introduced in considering marginal effects on predicted probabilities, as discussed below. In a GLM, we can avoid this inaccuracy for noncontinuous variables, like gender, by “flipping” its levels, assuming the other predictors levels do not change. In our example, for a female case “f”, we have

$$E(y^f|\mathbf{X}) = \mathbf{b}_0 + \mathbf{b}_1 \times 1 + \mathbf{b}_2 x_2^f = \mathbf{b}_0 + \mathbf{b}_1 + \mathbf{b}_2 x_2^f , \quad (2.5c)$$

and for any male case “m”, we have

$$E(y^m|\mathbf{X}) = \mathbf{b}_0 + \mathbf{b}_1 \times 0 + \mathbf{b}_2 x_2^m = \mathbf{b}_0 + \mathbf{b}_2 x_2^m , \quad (2.5d)$$

so that the flip from femaleness to maleness produces an adjusted partial effect of

$$(2.5c)-(2.5d) = \beta_1(x_2^f - x_2^m).$$

If gender is coded with unit difference between the levels, we again have effect = \mathbf{b}_2 . This assumes linearity in the weights, and the absence of important interactions.

If we have good reason to believe that the linearity assumption holds (e.g., prior research, exploring data analysis), we may now wish draw inference on the underlying connections between predictors and response. In the example above, say we found that being female versus male increased the length of hospital stay by a statistically significant (and a clinically substantive) amount, and that this discrepancy was not accounted for by age differences. We may postulate underlying biological or sociodemographic reasons for this, and take actions depending on the purpose of our original inquiry (e.g., designing further studies with additional predictors, focusing hospital quality improvement projects based on sex, approving different insurance reimbursements for men and women, etc.).

We may extent the concept above to all GLMs. An estimate of a parameter provides the partial effect of a predictor controlling for the other predictors.

We do not have to abandon the GLM if we have good reason to suspect interaction among the predictive effects (that is, when effect of one predictor depends on another). We may attempt to model an interaction in a multiplicative way by adding synthetic “interaction terms,” shown here for all possible 2-way interaction:

$$\mathbf{Y} = \mathbf{X}\mathbf{b} + \mathbf{e} = \sum_i \mathbf{b}_i x_i + \sum_i \sum_j \mathbf{b}_{ij} x_i x_j + \mathbf{e} \quad (2.5e)$$

Ideally, we would include only interactions considered plausible from earlier work. But in exploring complex systems, interactions are usually not known in advance. So we consider many two-way, and even higher order (like $\mathbf{b}_{x_i x_j x_k}$) terms. Of course, we get into statistical problems by testing hypotheses on a (combinatorially) large number of models. More important is the difficulty in drawing inference in the presence of interaction, because there is no single parameter that represents the effect of a predictor with interactions. If only a few interactions are modeled, we may use graphic displays, or create prediction rules and compute effects for specific cases. Flexible models, like ANNs, inherently consider interaction effects. Interpreting the net effects of individual predictors in ANN models is discussed in section 2.6.

Lao³ suggested five methods of interpreting the effects of parameter estimates in GLMs (that do not involve interaction among the effects). Methods 1, 2, and 3 apply to all GLMs, while methods 4 and 5 apply only to responses modeled as probabilities:

GLM METHOD 1. SIGN and SIGNIFICANCE of the parameter estimate. The simple sign (direction) of the effect is useful because it does not depend upon the GLM link function. Given a significant p value (or a CI that does not include 0), a positive sign suggests the

probability of the response event increases with the level or presence of the predictor, controlling for the effects of the other predictors.

GLM METHOD 2. PREDICTED VALUES of the model, \mathbf{Xb} , or transformed \mathbf{Xb} , given a set of values of the predictors. This is most useful in simple linear regression, where \mathbf{Xb} is the actual quantity of interest, and in Poisson regression, because the transformed \mathbf{Xb} is the predicted count. For logistic regression, the logit itself does not have intuitive meaning.

GLM METHOD 3. MARGINAL EFFECTS of the parameter values: the impact (and statistical significance) of one unit increase in one of the components of \mathbf{X} is insightful in linear regression without interaction terms. In probability models, a suitable transformation, such as the odds ratio in logistic regression, can provide meaningful information.

GLM METHOD 4. PREDICTED PROBABILITIES given a set of values of the predictors. We often want to know how the predicted probability will differ for different subsets of cases in a dataset, controlling for the effects of other predictors. For example, “adjusting for age, being male vs. female increases the probability of heart attack by 0.35.” In the case of a logistically-modeled probability response, we supply values or levels of each predictor of interest, compute \mathbf{Xb} (the logit), exponentiate to get the odds, and compute the probability. In the case of the logistic GLM response, we simply supply values or levels of a predictor of interest, set the values of the other predictors to one meaningful level at time (or to the mean, if continuous), compute the logit score, exponentiate to get the odds, and compute the probability. This can be problematic, because the putative joint means or levels may not exist in the real distribution of data (the model is uncalibrated in those regions of data space). And with m predictors, there are potentially 2^m comparisons to examine in the case of binary predictors alone (and exponentially more for multilevel variables). But if a limited number of specific subset comparisons are justified *a priori*, this process can provide absolute probability. Finally, absolute predictive probabilities make sense only if prior probabilities are definable; i.e., if the data was drawn from a population wherein rates of the predictive levels are initially defined. For example, this criterion is met by observations drawn from randomized or epidemiological cohort studies, because the prevalence of risk factors is available. But this criterion is not met in case control studies, because discrete outcome populations are assembled, resulting in an arbitrary mix of risk factor rates.

GLM METHOD 5. MARGINAL EFFECT on the PROBABILITY of an event. We can combine features of the last 2 methods to determine the effect of one unit increase on predicted probabilities. But the derivative (5.2) now includes the nonlinear logistic transformation, so that the effect parameter must be multiplied by (probability)*(1-probability). Because this will vary for each case, the other variables are only partially controlled by the GLM. Therefore, such comparisons are only approximate.

In the logistic regression model, methods 1-4 are useful, for the reasons described with each. Most statistical packages display results of methods 1 and 3. For the latter, we use the odds ratio (OR) formulation from to simply exponentiate the fitted weight to obtain the odds ratio of the binary response event occurring given a unit increment in the predictor from an arbitrary value x_0 to $x_0 + 1$ (that is, the marginal effect of the predictor):

$$\begin{aligned}
 OR &= \frac{ODDS\{Y=1 \mid (x_0+1)\mathbf{b}\}}{ODDS\{Y=1 \mid x_0\mathbf{b}\}} = \frac{\exp((x_0+1)\mathbf{b})}{\exp((x_0)\mathbf{b})} \\
 &= \exp((x_0+1)\mathbf{b} - (x_0)\mathbf{b}) = \exp(\mathbf{b}).
 \end{aligned}
 \tag{2.5f}$$

For example, a (statistically significant) estimate, $\mathbf{b}_1 = 2.38$, from a logistic regression model means that the probability of the response event increases linearly with a logit (log odds) of 2.38. The direction is helpful, but the value itself does not provide insight. But the OR formulation tells us that the odds of the event is $\exp(2.38)$, or 10.8, times greater for each unit increment in x_1 . If x_1 is a binary predictor, this means that the event is 10.8 times more likely to occur, than not to occur, in the presence of x_1 . Notice that the OR is a *multiplicative*, rather than additive, interpretation of the effect of a predictor.

&2.6 Drawing Inference in ANNs

Like the case of a GLM, we usually want to inquire about predictive effects after estimating the weights of an ANN model. Unfortunately, there no longer exists a *unique* identifiable relationship between a specific weight and the effect of a predictor (figures 2.3.1 and 2.3.2), because the effect depends upon multiple weights acting along multiple pathways that connecting it to the output.

For instance, in figure 2.6.1, predictor i will exert an effect (weighted by a parameter b_{ij}) upon hidden unit j . How this hidden unit responds, however, depends also on the weighted inputs from other predictor values for that case. This response is, in turn, passed along to the output unit (in a SHL-ANN) or to other hidden units. There will generally be many paths from a predictor to the prediction.

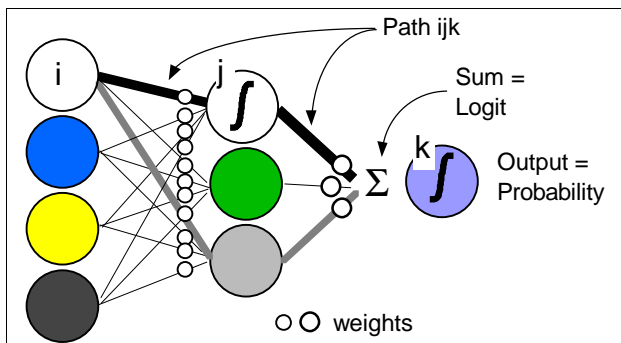


Figure 2.6.1. Directed graph of a arbitrarily interconnected network regression model. There is no requirement that hierarchical “layers” exist.

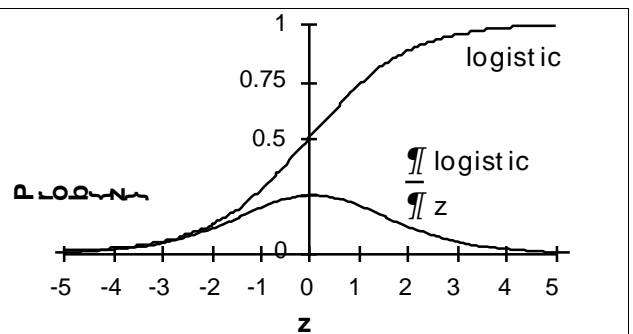


Figure 2.6.2. Simultaneous plots of logistic activation function and it's derivative.

We start with the same partial differential equation as for the GLM (5.2) to compute the effect of a predictor x_1 on the output of ANN. For compactness, we will use the logit formulation, which

is weighted input to the final logistic output unit k (figure 2.6.1). We will later convert the logit to an odds ratio expression, as we did for the GLM. In the absence of ambiguity, we may occasionally abuse notation for the sake of clarity.

First, consider the SHL-ANN (4.2). Taking the partial derivative, we have

$$\text{Logit effect of } x_i \equiv \frac{\eta_{\text{Logit}}}{\eta_{x_i}}, \quad (2.6a)$$

which can be expanded by the chain rule of partial derivatives as

$$= \frac{\eta_{\text{Logit}}}{\eta_{HO}} \frac{\eta_{HO}}{\eta_{x_i}} \quad (2.6b)$$

where HO is the output from the hidden units. The derivative of the logit with respect to a HU is simply the weight between them. Because each predictor x_i transmits effects via *all* hidden units, (2.6b) necessitates a summation over them,

$$= \sum_j^{\text{\#hidden units}} b_j \frac{\eta_{HO_j}}{\eta_{x_i}}. \quad (2.6c)$$

Again expanding the partial derivatives, we have

$$= \sum_j^{\text{\#hidden units}} b_j \frac{\eta_{HO_j}}{\eta_{HI_j}} \frac{\eta_{HI_j}}{\eta_{x_i}}. \quad (2.6d)$$

The partial derivative of HI with respect to x_i is simply the weight between them (the derivative of the inputs w.r.t. x_i for the other predictors is zero). So finally, for the SHL-ANN, we have

$$\text{Logit Mean effect of } x_i = \sum_j^{\text{\#hidden units}} b_j \frac{\eta_{HO_j}}{\eta_{HI_j}} b_i. \quad (2.6e)$$

The partial derivative of HO w.r.t. the total input (HI) is the derivative of the activation, or inverse logistic link, function, which depends on HI ,

$$\frac{\eta_{HO}}{\eta_{HI}} = \frac{\eta\left(\frac{1}{1+e^{-HI}}\right)}{\eta_{HI}} = \frac{-e^{-HI}}{(1+e^{-HI})^2}. \quad (2.6f)$$

Figure 2.6.2 shows a plot of the logistic and its derivative. At the center of the logistic (probability of 0.5), its derivative is maximal at 0.25, from which it symmetrical rounds a shoulder and asymptotically approaches 0 in either direction. With the understanding that it depends on HI, we will refer to the activation derivative w.r.t. its input as

$$Act'_j \equiv \frac{\mathcal{H}O_j}{\mathcal{H}H_j}. \quad (2.6g)$$

Notice that if the activation function of the hidden units was *linear* rather than logistic, (2.6e) becomes a product of constant values -- that is, a GLM:

$$effect\ of\ x_i | linear\ activation = \sum_j^{\#hidden\ units} \mathbf{b}_j (\text{constant}) \mathbf{b}_i = \mathbf{b}_i^{\text{composite}}. \quad (2.6h)$$

By (2.6e), there is a different effect of a predictor for each case, because the effect depends on the values of the other predictors, which jointly determine HI and thereby the derivative of the activation for each hidden unit. That is, we have a total interaction of effects. We can, however, *average* over all cases in a dataset to get the “mean effect” of a predictor. Reordering the multiplicative terms,

$$Mean\ Effect^{SHL-ANN}\ of\ x_i = \frac{1}{N} \sum_1^{N\ cases} \left(\sum_j^{\#hidden\ units} \mathbf{b}_i \mathbf{b}_j Act'_j \right). \quad (2.6i)$$

This formulation was proposed by Intrator and Intrator⁴ as a “mean weight” for a SHL-ANN.

We now derive the formula for the mean effect of an *arbitrarily* connected logistic ANN (figure 2.4.2). Applying the partial derivative chain rule to (2.4c), we have

$$effect^{ANN}\ of\ x_i \equiv \frac{\mathcal{H}Logit}{\mathcal{H}x_i} = \frac{\left(\mathcal{H} \mathbf{b}_0 + \sum_{i=1}^{\#input\ units} \mathbf{b}_i^{IF(Path^k)} x_i + \sum_{j=1}^{\#hidden\ units} \mathbf{b}_j^{IF(Path^k)} \frac{1}{1 + \exp(\text{Logit}_j^{\text{Hidden}})} \right)}{\mathcal{H}x_i} \quad (2.6j)$$

$$= \mathbf{b}_i^{IF(Path^k)} + \frac{\mathcal{H} \left(\sum_{j=1}^{\#hidden\ units} \mathbf{b}_j^{IF(Path^k)} \frac{1}{1 + \exp(\text{Logit}_j^{\text{Hidden}})} \right)}{\mathcal{H}x_i} \quad (2.6k)$$

$$= \mathbf{b}_i^{IF(Path_{ik})} + \sum_{j=1}^{\# \text{hidden units}} \mathbf{b}_j^{IF(Path_{jk})} \frac{\mathcal{H}O_j}{\mathcal{H}Logit_j^{\text{Hidden}}} \frac{\mathcal{H}Logit_j^{\text{Hidden}}}{\mathcal{H}x_i}. \quad (2.6l)$$

The \mathbf{b}_i term is the contribution from any existing shortcut connections (these were not allowed in the definition of a SHL-ANN). The partial derivative of HO w.r.t. the Logit is the same activation derivative (2.6g). The parenthetic term looks very similar to (2.6d), except that the input to the hidden unit is generically referred to as its incoming Logit. Unlike (2.6e), however, the logit can accumulate not only from (a subset of) the weighted predictors, but from the weighted output of earlier hidden units in the network. Differentiating (2.6l) recursively we obtain a sum of terms that can be expressed very simply if grouped according to paths from input to output:

$$= \mathbf{b}_i^{IF(Path_{ik})} + \sum_1^{\# \text{paths}} \left[\mathbf{b}_{\text{path}}^{\text{H-O}} \prod_{j \in \text{path}} \left(\mathbf{b}_j^{\text{H-H}} Act'_j{}^{\text{H}} \right) \right]. \quad (2.6m)$$

This conveys an intuitive sense to the predictive effect: referring to figure 2.4.2, start from the predictor and delineate all possible paths a signal could travel to reach the output. The first term represents “shortcut” connections directly from inputs to output; there are no intervening activation functions. The $\mathbf{b}^{\text{H-O}}$ are the weights from the final hidden unit to the output; there is no activation derivative multiplying these weights because we summate to obtain the logit, but do not go on to compute the output probability at this stage. Finally, we take the product of all hidden-to-hidden weights, $\mathbf{b}^{\text{H-H}}$, with activation derivatives along their paths (there will be a 1:1 correspondence for each hidden unit). Note that the summation will generally include replicates of \mathbf{b} ’s, because of overlapping path segments.

We now add these path effects together to estimate the total effect for a single case. As in (6.9), we recognize that each case contributes a different effect, and so we compute the mean effect of an arbitrarily configured logistic ANN as follows:

$$\text{Mean Effect}^{LANN} \text{ of } x_i = \mathbf{b}_i^{IF(Path_{ik})} + \frac{1}{N} \sum_1^{\text{N cases}} \left(\sum_1^{\# \text{paths}} \left[\mathbf{b}_{\text{path}}^{\text{H-O}} \prod_{j \in \text{path}} \left(\mathbf{b}_j^{\text{H-H}} Act'_j{}^{\text{H}} \right) \right] \right) \quad (2.6n)$$

It is readily seen that applying (2.6n) to a simple SHL-ANN reduces exactly to (2.6i), noting that the first term is not allowed in that model.

And for linear rather than logistic hidden unit activation functions, (2.6n) reduces to the effect of a predictor in a GLM:

$$\text{Mean Effect}^{Linear-ANN} \text{ of } x_i = \mathbf{b}_i^{IF(Path_{ik})} + \frac{1}{N} \sum_1^{\text{N cases}} \left(\sum_1^{\# \text{paths}} \left[(\text{constants}) \prod_{j \in \text{path}} (\text{constants}) \right] \right)$$

$$= \mathbf{b}_i^{\text{composite}}. \quad (2.60)$$

We now extend Liao's GLM-specific terminology (above) to the interpretation of effects of probabilistic ANN models:

- ANN METHOD 1.** SIGN and SIGNIFICANCE retain their meanings. Computing the significance analytically would require the imposition of distributional assumption on a highly nonlinear formula. We recommend the use of resampling techniques, especially bootstrapped variance or confidence intervals⁵. If significant, the sign carries the same meaning as in the GLM -- a positive (negative) sign suggests that the probability of the response event increases (decreases) with the level or presence of the predictor. Unfortunately, the fitted parameters only partially control the effects of the other variables, there being a new component of variance introduced by the distribution of activation derivatives. This reflects the differential degree of interaction for each case. We therefore recommend a 2-part bootstrap estimation of the variance:
- (1) Using the full dataset model, bootstrap the original dataset many hundred (or thousands) of times to compute variances (or compute confidence intervals) for the booted mean effects. This computation is cheap, because no new optimization is required. If any predictor's mean effect is thereby not significant, there is no need to look at the component of its variation due to parameter uncertainty.
 - (2) *For nonbinary predictors.* Using the same bootstrapped samples (minus the predictors determined to be insignificant in part (1)), segment the mean effects according to the level of the predictor, and again inspect for sign and significance. For interval and ordinal variables, use each level itself; for continuous predictors, divide each into meaningful quantiles, treating each as a level. The importance of this step is that, by looking at a single level, the observed variance reflects mainly interaction, the predictor variable's nonlinearity (curvilinearity) being relatively ignored. If any predictor's level-mean effect is thereby not significant, there is no need to look at the component of its variation due to parameter uncertainty. It may be reasonable, therefore, to collapse levels (or, recode continuous predictors into ordinal levels). Note: because their mean effects would have already been determined to be significant in step (1), retained binary predictors will be found significant at both levels in this step (because there is no nonlinearity to remove).
 - (3) Generate bootstrap samples of the dataset, and, using the identical architecture and weight initialization, fit these booted ANN models. For each predictor, compute mean effects for the booted models, and the variance (and significance) across these mean effects. Drop those predictors with low significance. Note: this computation is intensive, because of the time required to optimize so many models.

ANN METHOD 2. PREDICTED VALUES and significance of the model, given a set of values of the predictors. For a continuously valued prediction of an ANN, as in the case of the GLM, this is inherently meaningful. For a logistic output prediction, whether GLM or ANN, the logit itself is not intuitively meaningful. For a logistic probability model, as in the case of the GLM, this will be the logit or transformed logit, which does not have intuitive meaning.

ANN METHOD 3. MARGINAL EFFECT. For continuously valued-dependent variables, the impact of one unit increase in **X** is insightful, as in the case of the GLM without interaction effects. Rather than evaluating each parameter individually, the global mean effect must be estimated (see above, and sections 5.6.1.9 and 6.2). In probability models, a suitable transformation, such as the OR, can also provide meaningful information. For nonbinary predictors, one can also use the OR formulation to evaluate the contribution of interaction at different levels of the predictor. We recommend the following procedure (see section 6.2) for probability models:

- (a) Because we wish to inspect the OR as the predictor is incremented along its levels (or quantiles, if continuous), we take the lowest level as a reference (this is also done in GLM interpretation).
- (b) Unlike the GLM, the mean effect (logit) may differ across levels, so we need a way to obtain average inter-level changes in logit. Each level is given equal importance, by weighting its mean effect (logit) by half the distance between the 2 levels.

Details: the OR of any level relative to the reference level is found by exponentiating the cumulative logit to that level. The cumulative logit is the sum of the changes in logits between preceding levels. The change in logit between any two levels is computed by summing the product of the mean effect at one level by the half-distance between them, with the same product for the other level:

$$\begin{aligned} \text{Interlevel change in logit} = \\ (\text{mean effect at level A}) * [.5 * (\text{interlevel distance on predictor scale})] + \\ (\text{mean effect at level B}) * [.5 * (\text{interlevel distance on predictor scale})] \end{aligned}$$

Rearranging terms, we have,

$$\begin{aligned} \text{Interlevel change in logit} = \\ (\text{interlevel distance on predictor scale}) * \\ [(\text{mean effect at level A}) + (\text{mean effect at level B})] * .5 \end{aligned}$$

or,

$$\begin{aligned} \text{Interlevel change in logit} = \\ (\text{interlevel distance on predictor scale}) * \\ (\text{average of the mean effects at levels A \& B}) \end{aligned}$$

A geometric interpretation of this is given in figures 2.6.3 and 2.6.4.

- (c) This process moves the difference up or down on the logit scale, by a cumulative amount that can now be exponentiated to obtain an interpretable level-specific OR (see example in section 6.2).

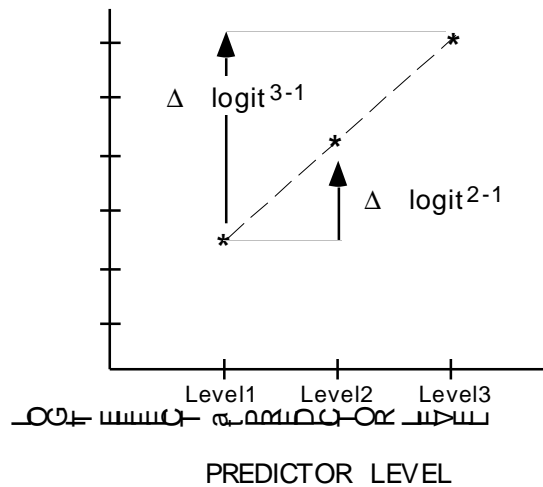


Figure 2.6.3. GLM Marginal Effects. Logistic GLM logit as a function of the level of a predictor. The GLM model constrains the logit to increase linearly with predictor level. That is, the direction (sign) of the effect cannot change over the levels of the predictor, so that the actual logit increments linearly. The odds ratio (relative to reference level) is computed by exponentiating the net change in logit between any two levels.

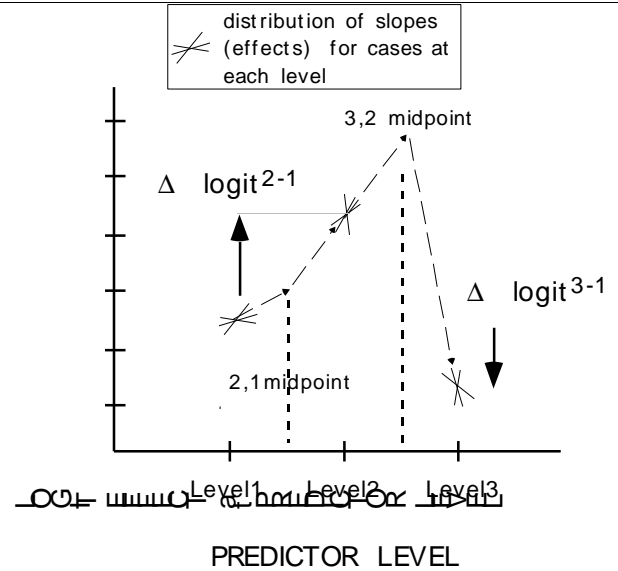


Figure 2.6.4. ANN Marginal Effects. Proposed ANN mean-logit changes with level of predictor. For continuous variables, the mean value of a quantile is chosen to represent its level. Because nonlinear effects are permitted, the logit may change nonmonotonically with increasing levels of a predictor. Method: Graphically, a vector is extended from the first (reference) level a distance half way to the next level. The slope of this vector is equal to the mean effect. At this midpoint, another vector is extended the remaining half-distance, with a slope equal to the mean effect at the next level. This process of vectorial addition is repeated at each subsequent level.

ANN METHOD 4. PREDICTED PROBABILITIES given a set of values of the predictors. We often want to know how the predicted probability will differ for different subsets of cases in a dataset. In the case of the logistic GLM response, we simply supply values or levels of a predictor of interest, set the values of the other predictors to one meaningful level at time (or to the mean, if continuous), compute the logit score, exponentiate to get the odds, and compute the probability. Even in the GLM this can be problematic, because the putative joint means or levels may not exist in the real distribution of data (the model is uncalibrated in those regions of data space). And with m predictors, there are potentially 2^m comparisons to examine in the case of binary predictors alone (and exponentially more for multilevel variables). Also keep in mind that absolute probabilities make sense only if the data was derived from a sample of cases at equal risk of an event. Furthermore, In the ANN model, the model parameters only partially adjust for other predictors' effects. We therefore suggest the following approach: once the model is fit, compute, for a predictor, each level's mean effect as described above in ANN METHOD 2. Transform it logistically to the probability scale. Because each mean effect is the average over cases in a subset of the original data, we are essentially supplying, for the other predictors, the means for each subset.

ANN METHOD 5. MARGINAL EFFECT on the PROBABILITY of an event. As discussed under this method for the GLM, including the final logistic function in the derivative of the output introduces a final nonlinearity in the parameters b^{H-O} in (2.6l). Effects passing through hidden units in an ANN already introduce this nonlinearity, which is why an ANN can only partially adjust for the presence of other predictors in the model. Intrator and Intrator⁴ suggested normalization by dividing by the activation derivative at each hidden unit (in the context of a SHL-ANN), but this no longer represents the partial derivative of the output w.r.t. the predictor in the fitted ANN model. The final nonlinearity may or may not contribute to further variance in the predictions, given that already introduced via the hidden units. We recommend that if a marginal probability statement is justified by the nature of the observations, bootstrapped variances (or confidence intervals) be generated for this statistic, and the p values compared with those of the OR's. Substantial deterioration suggests that the marginal probability may not be a reliable measure for the given dataset.

&2.7. Dealing with Missing Data

Missing elements of data sets are all-to-common in the real world. When all measurements (variables) arise from a single source (as in a questionnaire or survey), the mechanism of missingness is usually clearly understood, and globally applicable to the dataset. In more complex datasets, however, there are often *multiple* mechanisms of missingness. For instance, aggregated medical databases often contain demographic information drawn from administrative sources, laboratory values from hospital computers, and clinical status and outcomes extracted from chart summaries. Scientific and engineering databases may be created using measurements from a diverse array of laboratory devices.

MECHANISMS. The mechanism of missingness is said to be missing *completely* at random (MCAR) if missingness does not depend on the value of observed or missing variables (including the variable under consideration). Less stringently, if missingness depends on values of the observed data, but not on the missing data, then data are said to be missing at random (MAR). Reasons for missingness that depend on missingness are said to be not missing at random (NMAR).

MANAGEMENT. Detailed discussions can be found in books by Rubin⁶ and Little and Rubin⁷, and recent survey articles by Little and Rubin⁸, Little⁹, and Little and Schenker¹⁰. Below, we briefly review common methods use do handle missing data:

- COMPLETE-CASE (CC) ANALYSIS

Only complete cases are utilized; cases with missing elements are discarded. This remains the default method in most statistical and database software packages because of its ease of implementation. While CC analysis provides a baseline, the loss of the observed data reduces the predictive accuracy of the final model. Furthermore, inferences from CC models are valid only if the mechanism of missingness is MCAR, otherwise, estimates are biased toward only

data subsets with CC. Correction for such bias is hazardous, and feasible only with single mechanisms of missingness that are well understood (as in surveys).

- AVAILABLE-CASE (AC) ANALYSIS

On subsets of variables, the largest set of available cases are used depending on the parameters being estimated. For example, to estimate the correlation between two variables, only those cases complete in both variables are used. It appears to use information better, but even in this simple model, the correlation matrix may not be positive definite since the entries are derived from different sets of cases. For some estimates, like population means, AC is clearly better than CC. But this may not be true for other estimates (Haitovsky¹¹). Standard errors of estimates using AC require custom formulas. Of course, this approach reduces to CC analysis for models that use all the variables.

- IMPUTATION

Unconditional mean imputation. Missing values are assigned the unconditional observed-sample mean (or some other constant value, like the median). The method is easy to implement but suffers from inefficiency and inconsistency for many estimates even under MCAR assumption. Standard errors from the filled-in data are underestimated because sample size is overstated.

Conditional mean imputation. Missing values are predicted from the observed values of other variables, assuming MAR. This could be done by regression (see Harrell; Bucks¹²). Although an improvement over unconditional mean imputation, it may be unsatisfactory for estimating quantities that are not linear in the data such as percentiles and measures of association. Standard errors from the filled-in data are underestimated. A related method used in surveys is the “hot deck” (Madow et. al.¹³). In this technique, adjustment cells are formed by predictor variables. Missing values are imputed from observed values in the same cell. This is a form of a nearest neighbor algorithm.

Multiple imputation (MI). Multiple imputation was first proposed in Rubin [1978] in the context of sample surveys. Extensive treatment of the subject in the same context is found in Rubin¹⁴. The principle of MI is to draw $K=2$ imputations for each missing value, instead of imputing a single “best” value. This generates K complete data sets. Each data set is analyzed separately by standard methods. The K results are then combined into a single inference that reflects the sampling variability due to missing values. The imputation can also be carried out under different models for missingness so that sensitivity to model choice can be evaluated.

Expectation-Maximization (EM). Parameters of a model are estimated directly from a likelihood function; no imputation is needed. To accomplish this, joint distributions of observed and missing data must be specified.

$$p(X, M | \theta, \phi) = p(X | \theta) p(M | X, \phi).$$

Thus given the observed data (X_{obs}, M) , the likelihood function for the parameters is

$$L(\theta, \phi | X_{\text{obs}}, M)$$

$$= \int p(X | \theta) p(M | X, \phi) dX_{\text{mis}} \quad (2)$$

MLE of θ and ϕ are obtained by maximizing (2). Usually, θ is of interest whereas ϕ is a nuisance parameter. If data is MAR and θ and ϕ are functionally unrelated, i.e., if the missing data mechanism is ignorable (Rubin [1976]), then MLE of θ is obtained from the marginal likelihood function

$$L(\theta | X_{\text{obs}}) = \int p(X | \theta) dX_{\text{mis}}. \quad (3)$$

Thus assuming ignorability, model for missing data needs not be explicitly specified. To maximize (2) or (3) usually requires iterative procedures, the most well-known of which is the EM algorithm (Dempster, Laird and Rubin¹⁵) tailored to solve incomplete data problems.

ML estimates have desirable properties (consistency, efficiency, etc.) if model specification is correct. Asymptotic covariance matrix of ML estimates can be computed via standard procedure. While EM algorithms are relatively easy to code, for small or even moderate sized data sets, the likelihood functions may be highly nonnormal and have local maxima.

As before, let $p(X | \theta)$ be the model (distribution) of data parameterized by θ (not treated as a random variable). Let $l(\theta | X) = \log p(X | \theta)$ be the log likelihood function of the full data and $l(\theta | X_{\text{obs}}) = \log p(X_{\text{obs}} | \theta)$ be the log likelihood function of the observed (non-missing) data. Since only the X_{obs} portion of the data is available, the object of ML is to find an θ^* in the parameter space such that $l(\cdot | X_{\text{obs}})$ is maximized. The Expectation-Maximization (EM) algorithm (Dempster, Laird and Rubin [1986]) is an iterative maximization tool. The outline of EM given here applies to ignorable missing data mechanism. The extension to nonignorable case is straightforward; only the likelihood function is more complicated (see, e.g., Little and Rubin [1987], Chapter 11).

In EM, the conditional distribution of X_{mis} given X_{obs} is parameterized by θ :

$$\begin{aligned} p(X_{\text{mis}} | X_{\text{obs}}, \theta) &= \frac{p(X | \theta)}{p(X_{\text{obs}} | \theta)} \\ &= \frac{p(X | \theta)}{\int p(X_{\text{obs}}, X_{\text{mis}} | \theta) dX_{\text{mis}}} \end{aligned}$$

plays a key role. At iteration t , the iterate θ^t is available; let

$$p^t(X_{\text{mis}}) = p(X_{\text{mis}} | X_{\text{obs}}, \theta = \theta^t) \quad (4)$$

denote the current conditional distribution of X_{mis} given X_{obs} . the E-step consists of computing the expectation of the full-data log likelihood function $l(\theta | X) =$

$l(\theta | X_{\text{obs}}, X_{\text{mis}})$ (treated as a function of the random vector X_{mis}) assuming the missing values are distributed according to $p^t(\cdot)$:

$$Q^t(\theta) = E_{\{p^t\}}[l(\theta | X) | X_{\text{obs}}] \\ = \int l(\theta | X_{\text{obs}}, X_{\text{mis}}) p^t(X_{\text{mis}}) dX_{\text{mis}}. \quad (5)$$

The M-step is to maximize $Q^t(\theta)$ over the parameter space, say Θ . Formally, the EM Algorithm is given as follows:

- Step 0. Initialize θ^1 , for $t < 1$.
- Step 1. (E-step) Define $Q^t(\theta)$ as in (4)–(5).
- Step 2. (M-step) Find θ^{t+1} "in" $\arg\max\{Q^t(\theta) \mid \theta \text{ "in" } \Theta\}$.
- Step 3. $t \leftarrow t+1$. Return to Step 2.

The M-step may require a considerable amount of computational effort if the dimension of the parameter space is high. However, the maximization does not have to be exact; a generalized EM (GEM) only requires finding θ^{t+1} in Step 2 such that

$$Q^t(\theta^{t+1}) > Q^t(\theta^t).$$

GEM ensures that the log likelihood function $l(\cdot | X_{\text{obs}})$ iteratively increases. Under general regularity conditions GEM (and hence EM) converges to the true ML value θ^* . In particular, if θ^* is unique, the algorithm will find it.

EM for exponential families. EM algorithm is much simplified if the distribution of the complete data X is from the regular exponential family. That is, the distribution is of the form

$$p(X | \theta) = b(X) \exp(s(X) \theta) / a(\theta)$$

for some functions $b(\cdot)$, $s(\cdot)$, $a(\cdot)$ of their respective arguments only. Note that $s(X)$ is the natural complete data sufficient statistics for θ . In this case, it can be shown that the E-step in iteration t reduces to computing the conditional expectation of the complete data sufficient statistics evaluated at θ^t :

$$\text{E-step} \quad s^t \leftarrow E[s(X) | X_{\text{obs}}, \theta^t].$$

The M-step involves solving a system of equations:

$$\text{M-step} \quad \text{Solve } E[s(Y) | \theta] = s^t \text{ to obtain } \theta^{t+1}.$$

So, no maximization is involved in the M-step; the computational effort is mostly confined to the E-step. Further details can be found in Little and Rubin [1987] Section 7.6.

- BAYESIAN METHODS

ML methods may not work well for small data sets. The Bayesian solution is to impose a prior distribution on θ . Inferences then are based on the posterior distribution $p(\theta | X_{\text{obs}})$. Generally, it is a hard task to compute the posterior distribution. Techniques to simulate the posterior distribution have been proposed; these include data augmentation (Tanner and Wong¹⁶) and the Gibbs sampler (Geman and Geman¹⁷). Such techniques can also be applied in Step 1 of MI.

CHAPTER &3. OBTAINING & SETTING UP NevProp VERSION 3

&3.1 Availability

The most updated version of NevProp3 will be available by anonymous ftp from the University of Nevada, Reno:

3.1.1 Direct Internet file transfer (“ftp”)

- a. Create an FTP connection to:

```
> ftp ftp.scs.unr.edu
```

- b. login as “anonymous,” and use your email address as password.

- c. Change directory:

```
> cd /pub/cbmr/nevpropdir/
```

- d. At this point you should be able to request a listing of files:

```
> ls
```

- e. If you want to compile the program yourself, get (in binary mode) the compressed archive “NP.tar.gz”...

```
> binary
> get NP.tar.gz
```

On UNIX platforms, you unpack the file into a directory call “NevProp3/” using the command:

```
> gzip -d -c NP.tar.gz | tar xfv -
```

On other platforms, use a decompression program that handles gzip documents.

- f. Executable NevProp3 will be found in the subdirectories “dosdir” for DOS, and “macdir” for Macintosh versions, and should be transferred in binary mode.

- g. You may also download the current version of the NevProp manual (this document), it is presently available only as an MS Word document, readable by version 5 or higher (NPMAN.DOC).

3.1.2 World Wide Web

Open a connection to the following URL:

`ftp://ftp.scs.unr.edu/pub/cbmr/nevpropdir/`

The files named in the preceding section should be accessible. Remember that a WWW browser will attempt to *read* a file, rather than download it to disk, unless it is configured to “save” files with the specific extensions listed. For example, in Netscape® versions 2 & 3, under the “General Preferences” menu is the option to set “Helpers” (and direct to save to disk) according to specific extension or type.

&3.2 Compilation/Installation

NevProp3 has been successfully compiled and run (see below) under the following systems; the “make” command we used is also indicated:

```
HP-UX (HP 9000/750 and 9000/385)
    make "CC = cc -Aa"
Sun Solaris 2.4 and 2.5 (several SparkStation configurations)
    make "CC = gcc"
DEC OSF1 v3.2 (Alpha)
    make "CC = cc -std"
LINUX 1.1.13
    make
```

If you have compilation problems, first check that the compiler options are set for strict ANSI C compliance.

The Makefile text itself (viewed through a text editor) describes various compile-time options, and suggests optimizations for different platforms.

Some general remarks:

- The default Makefile defines `rand48()` as available. If you don't have `rand48()`, but do have `random()`, you can define it in the Makefile. Both are better randomization functions than the NevProp internal default `rand()`, which is used if neither `rand48()` or `random()` are defined at compile time. Macintosh and DOS executable versions use `rand()`.
- If you have GNU C compiler (`gcc`) on SunOS, you may achieve faster performance compiling with `-O2` and `-finline`, and `-static` link.

COMPILING UNDER PRE-ANSI C: NevProp should compile with either an ANSI or traditional C compiler. However, unless you compile it with `CALC_CINDEX` defined as 0, it currently requires the equivalent of the ANSI C library function `qsort()`, and the header file `stdlib.h`. If your system doesn't have one, try creating `stdlib.h` in the NevProp directory (`#include "stdlib.h"` is used so it need not be placed in a system directory), merely containing the declarations below. You may also need to cast the type of the second argument in calls to `qsort()`.

```
extern void malloc();
extern void free();
extern void qsort();
```


December 14, 1998 11:24 PM

```
extern void exit();
```

CHAPTER &4. INTRODUCTORY TUTORIALS

The following tutorials are intended to familiarize both novice and experienced analysts with NevProp3's basic design and features. A basic familiarity with linear regression theory and modeling is assumed. To the degree possible, the tutorials reflect realistic approaches to the statistical analysis using artificial neural networks.

For those with extensive experience in statistics and neural networks, it may suffice to simply read through the material.

&4.1 Continuous Prediction: Creating a Simple Data Set and Network

Time required: About 1 hour (after NevProp3 is completely installed)

Objectives:

- Understand the purpose of the network file
- Create a .net file with a few non-default settings
- Understand how to supply data to NevProp3
- Understand how to execute NevProp3 in fully-interactive and command-line modes
- Understand how to interpret the basic output of a NevProp3 run
- Understand how to properly integrate linear and nonlinear modeling approaches

SCENARIO. Consider the following artificial, but instructive scenario. You measure cholesterol on a sample of 10 patients hospitalized for a heart attack, their relative age, and the number of days each patient spent in the hospital. You will first perform exploratory data analysis, then create a linear model, then a nonlinear model, and finally reach a conclusion as to the best model to use.

STEP 1. Using a word processor, create a text file called "myfirst.net."

STEP 2. Enter the following lines of text (or copy and paste them, if viewing this document on-line). Any number spaces or tabs may be used to separate words, and any number of blank lines may separate lines of text:

```
# This is my first attempt at a NevProp3 network file.
# The following is a linear model.

Ninputs 2           Nhidden 0           Noutputs 1
Connect 1 2      3 3 /* connects inputs to output unit */

AutoTrain NO           ScoreThreshold 0.1

NHeaders 1           IDColumn YES
DATA
ID      chol      age      days
A       100       0       10
B       200       0       20
C       300       0       30
D       400       0       40
E       500       0       50
F       100       1       35
```

G	200	1	40
E	500	0	50
F	100	1	35
G	200	1	40
H	300	1	45
I	400	1	50
J	500	1	55
# The next ten cases were created by adding a			
# small random Gaussian deviation to the first ten.			
# The patient ID reflects each case's origin.			
AA	100	0	8
BB	200	0	14
CC	300	0	36
DD	400	0	32
EE	500	0	46
FF	100	1	33
GG	200	1	36
HH	300	1	41
II	400	1	54
JJ	500	1	57

Figure 4.1.1 This verbatim copy of the 'myfirst.net' network file can be copied and pasted into an active NevProp3 directory.

STEP 3. **EXPLANATION OF SETTINGS.** The pound sign (#) causes NevProp3 to ignore any subsequent text on that line. You can place comment lines to remind yourself about the file's contents and history. For more options, see section 7.1.

The line starting with 'NInputs' instructs NevProp3 to: (a) read the first two columns of data as inputs (predictor, or independent, variables), (b) create no "hidden" units, because we will first fit a linear model, and (c) read one column of data (following the two input columns) as the output target (dependent) variable, which will receive a weighted signal from each of the hidden units.

The Connect statement specifies that units indexed 1 through 2 (the inputs) are to be connected to unit 3 (the output). Actually, you could leave this statement out in this example, because, in the absence of hidden units, NevProp3 will default to connecting all inputs to all outputs.

The resulting directed graph of the network can be visualized as follows:

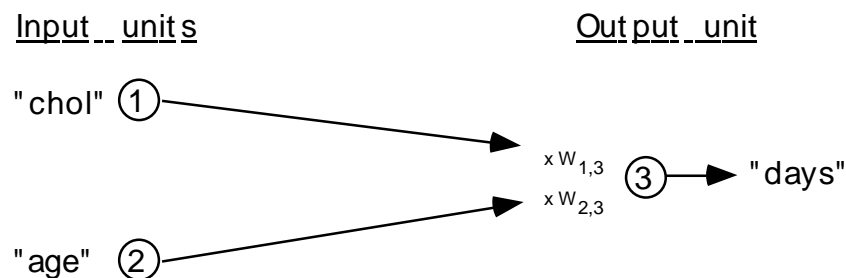


Figure 4.1.2 Directed graph of simple linear model. Numbers indicate the index of each unit. Weights, w , are multiplied by signals from corresponding units. At convergence to output unit, sums of weighted

signals are applied to an appropriate activation function (here, linear) to determine the output (prediction).

Notice that NevProp3 assigns sequential index numbers to the units. It also assigns paired indexes, and random initial values to the weights. Not shown are the connections from an implicitly defined bias (the “constant” in a regression formula) input to all hidden and output units. For details of the regression mathematics, please see Chapter 2.

The setting “ScoreThreshold 0.1” tells NevProp to count a prediction as correct if it is within 10% of the range of the output variable. Here, the range of ‘days’ is (55-10=) 45; predictions will therefore be counted as correct if within 4.5 days of the true target. (If omitted, the ScoreThreshold would have been set to the default value of 0.5.)

The setting “NHeaders 1” tells NevProp to read in a single line of text before the lines of numeric data. Here, we’re using the header to label the columns of data.

The setting “AutoTrain NO” tells NevProp not to internally split the training data to regularize the function, because we are not likely to overfit a simple linear model. More on ‘AutoTrain’ later, in the nonlinear model.

The setting “IDColumn YES” tells NevProp that the first column of each case in an alphanumeric identifier, or label, to be kept with each case.

Because we appended the training data within the .net file (rather than creating an external data file— see section 7.3.1), we must provide the keyword DATA, on its own line, after which follows the case data.

- STEP 4. **INSPECTION.** Before we run NevProp3, let’s explore the data graphically (this was done with a spreadsheet/graphics program). The first ten cases were chosen so that it is easy to see that that, as cholesterol (chol) increases, so does the hospital stay (days). In other words, the effect of cholesterol is to linearly prolong a patient’s hospital stay. One biological explanation might be that the higher the cholesterol, the more severe the atherosclerosis. This leads to more extensive heart damage, as reflected by a need for longer hospitalization. Cholesterol’s effect is relatively less strong for older than younger patients, as seen by the shallower slope— this could be due to the existence of other conditions associated with aging that influence the need for hospital stay.

Graphically:

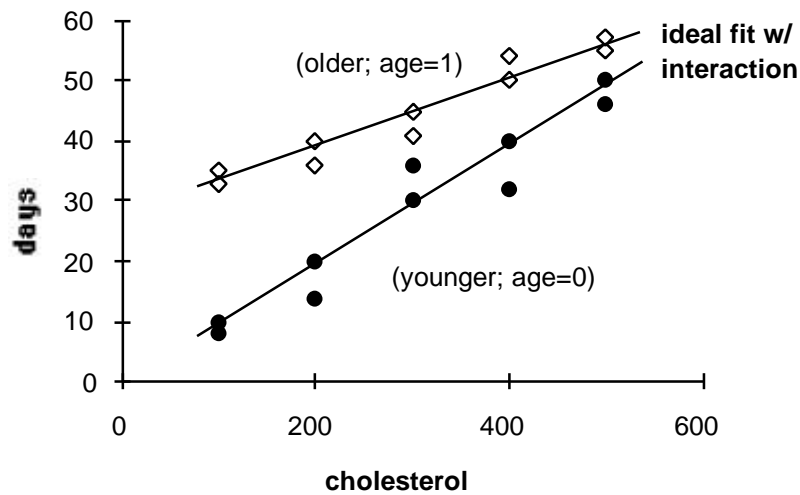


Figure 4.1.3 Plot of the 20 data points in `myfirst.net`, with ideal a priori bilinear fit. Differing slopes indicate an interaction of cholesterol and age.

Notice also that older (vs. younger) age is associated with a net upward shift in hospital stay. This makes sense, because age is a surrogate for other diseases that, cumulatively, may adversely impact a patient's health.

It is apparent from the figure that no single linear fit will accurately portray the relationship. The interaction between cholesterol and age must be accounted for, if the model is to accurately portray the underlying association.

One possibility is to ignore age completely; the best global fit would look like this:

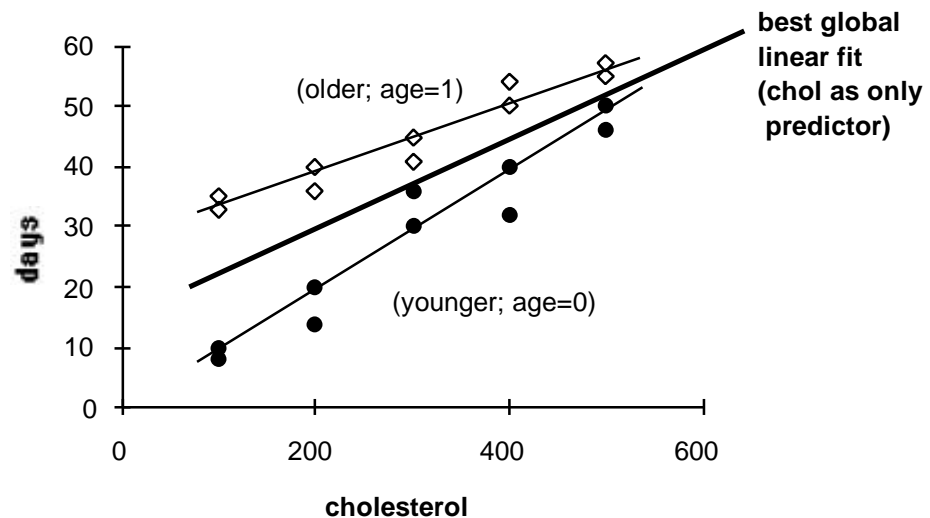


Figure 4.1.4 Simple linear fit of cholesterol to days in hospital, ignoring age.

Clearly the residual variation from the observed data to the linear fit would be large. Including age as a linear predictor, we can do better:

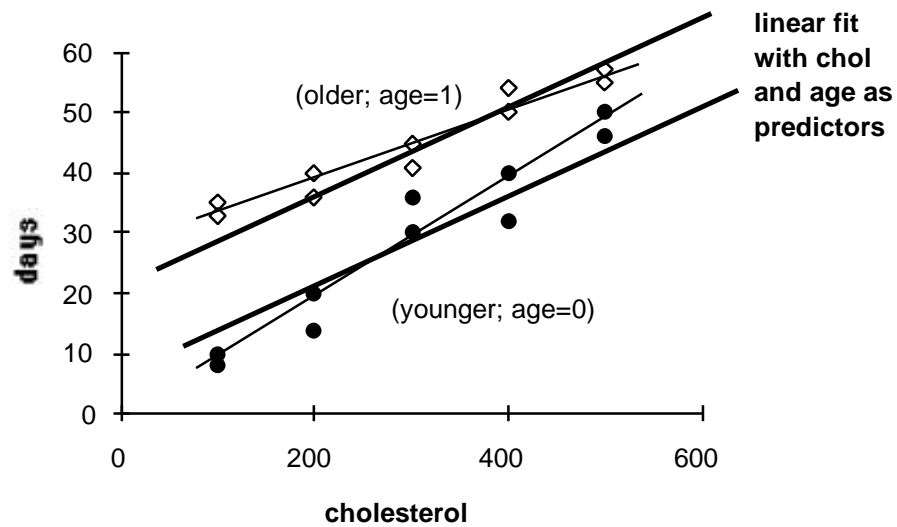


Figure 4.1.5 Linear fit of cholesterol to days in hospital, including age, but no interaction terms.

The residual variation is now less, because age, as a linear predictor, can add a fixed shift to the relationship of cholesterol and days. But the linear model cannot account for the fact that the slope (effect) of the relationship differs according to age group.

In this two-predictor toy problem, the simple nonlinear interaction is easy to see. The traditional statistical approach would be to include an artificial third predictor equal to the product of age and chol. The effect of cholesterol could then change with age, as in the first figure above.

For now, let us assume that the given predictors are two among many in a complex model, and proceed to approach the problem as proposed in figure 1.1 in Chapter 1.

STEP 5. FIT A LINEAR MODEL. In the .net file above, we specified no hidden units, so the model is linear. Execute NevProp3 in the fully-interactive mode:

Fully interactive: At your host's prompt, type:

```
> np
```

- When prompted for a random **seed**, enter any integer, or default return.
- When prompted for the name of the **.net** file, enter 'myfirst'.
- When prompted for the name a **results** file, enter 'myfirst'.
- When prompted for the name a **weights** file, decline by hitting enter.
- When prompted for **maximum number** of epochs, enter '60'.
- When prompted for **reporting interval**, enter '3' or accept default.
- When asked if **done** with training, enter 'y'.

- When asked to **save weights**, enter 'y', then accept default name.
- When asked to **save predictions**, enter 'y', then accept default name.

STEP 6. **CONFIRM THE SETTINGS.** You should now see a series of reports, beginning with the contents of the header, followed by regurgitation of format and setting information (you may scroll back up through your terminal's display, or open the 'myfirst.res' results file that was saved to disk):

```
##### Starting NevProp NP3.0
#####

HEADER #1: ID    chol    age    days

FYI: InputColumns & OutputColumns were not individually specified--
      NVars will be set to the sum of Ninputs and Noutputs (3).

FYI: FileFormat was not specified -- assuming v3 network file.

... SETTINGS were successfully read from "myfirst.net".
... SEED=12345 using lrand48(),srand48()
... Read in 20 TRAIN CASES

# DATA FILE SETTINGS
  NHeaders 1          IDColumn YES
  StandardizeInputs 1  ImputeMissing discard
  InputColumns 0
  OutputColumns 0
  NVars 3          ShuffleData YES
# REPORTING SETTINGS
  DescribeVars NO
  NBoots 0          NEffectBoots 0
  CalccIndex NO    ScoreThreshold 0.1
  OutputStatVars 0
# CONNECT CALLS
  Connect 1 2 3 3
# CONFIGURATION SETTINGS
  Ninputs 2          Nhidden 0          Noutputs 1
  kNN 0             lofN NO
  HiddenUnitType 1   OutputUnitType 3
  WeightRange 0.001
# TRAINING SETTINGS
  TrainCriterion 0
  BiasPenalty NO    WeightDecay -0.001
  OptimizeMethod 1   SigmoidPrimeOffset 0
  QPMaxFactor 1.75   QPModeSwitchThreshold 0
  Stochastic NO      LearnRate 0.01     SplitLearnRate NO    Momentum
0.01
# BEST-BY-HOLDOUT SETTINGS
  NHoldout 0          PercentHoldout 0.00
  AutoTrain NO        MinEpochs 200     BeyondBestEpoch 2
  NSplits 1           SepBootXVal YES
# AUTOMATIC RELEVANCE DETERMINATION SETTINGS
  UseARD NO           WhenARD Auto        ARDTolerance 0.05 ARDFreq
5
  GroupSelection Input BiasRelevance NO    ARDFactor 1
```

Figure 4.1.6 Display of NevProp3 settings at the start of a run.

STEP 7. **CONFIRM THE INTEGRITY OF THE DATA.** A descriptive summary of each variable is provided. This is a convenient way to ensure that you

NevProp3 is reading the data you anticipated. It is also a good way to remind yourself of the distributions, outliers, and missingness in your dataset.

```
##### Descriptive Statistics on Output (Dependent) Variables
#####
-----
      variable          n      unique      mean
      ^^^^^^^          ^^^^^      ^^^^^
1.  days              20         17      36.600
percentile:    0.05    0.10    0.25    0.50    0.75    0.90    0.95
value:        10.000  14.000  32.000  40.000  50.000  55.000  57.000

lowest:        8.0000  10.000  14.000  20.000  30.000
highest:       50.000  50.000  54.000  55.000  57.000
-----

##### Descriptive Statistics on Input (Predictor) Variables
#####
-----
      variable          n      missing      unique      mean
      ^^^^^^^          ^^^^^      ^^^^^      ^^^^^
1.  chol              20         0         5      300.00
    100.(            4, 20.0%)  200.(            4, 20.0%)  300.(            4, 20.0%)
    400.(            4, 20.0%)  500.(            4, 20.0%)

-----
      variable          n      missing      unique      mean
      ^^^^^^^          ^^^^^      ^^^^^      ^^^^^
2.  age              20         0         2      0.5000
    0.00(            10, 50.0%)  1.00(            10, 50.0%)

-----

##### End of Descriptive Statistics
#####
```

Figure 4.1.7 Display of variable descriptions at the start of a run, requested by the setting “DescribeVars YES”.

The variable descriptions, above, are based on the raw data. NevProp3 automatically chooses a display format (dichotomous, ordinal, or continuous) based on a preliminary analysis of each variable.

Prior to the start of optimization, the data residing in machine memory has been standardized (the default is to standardize all input variables to mean zero, units of standard deviation— see section 7.3.5). The rationale is that all input variables should be on a similar scale for reliable optimization, and for later interpretation of the model.

STEP 8. REVIEW THE OPTIMIZATION. A descriptive summary of each variable is provided. This is a convenient way to ensure that you NevProp3 is reading the data you anticipated. It is also a good way to remind yourself of the distributions, outliers, and missingness in your dataset.

Notice that performance is reported as *error* for both the criterion function (here, average square error) and scorethreshold fraction. Also, decimal places

are moved over 3 to the right. Seemingly awkward at first, this presentation provides a nearly optimal yet compact pattern of information to the user (given the constraints of a character-based interface).

```

-----
ABBREVIATIONS KEY:
LrnRat=LearnRate, FrGrdD=Fraction of weight updates using Grad Descent,
Av=Average, Sq=Squared, Wt=Weights, A=(Average over cases & outputs),
Pn=Penalized, CrEn=Cross-Entropy, Er=Error, Th=Thresholded, 1-c=(1 - c index)
-----

```

Train	LrnRat	AvSqWt	____TRAIN-SET_(20)____		
Epoch	(e-3)	(e-3)	APnSqEr	ASqEr	AThEr
			(e-3)	(e-3)	(e-3)
0	10.000	0.0003	1540077	1540077	1000.
3	9.4500	19799.	446473.	446472.	850.0
6	10.940	50230.	121211.	121209.	800.0
9	12.664	73896.	36184.9	36181.2	450.0
12	14.660	87543.	19153.5	19149.1	300.0
-- lines omitted for brevity --					
24	26.327	96876.	16419.9	16415.0	300.0
27	30.477	96896.	16419.8	16415.0	300.0
30	33.601	96898.	16419.8	16415.0	300.0
-- lines omitted for brevity --					
57	31.507	96898.	16419.8	16415.0	300.0
60	31.507	96898.	16419.8	16415.0	300.0

Figure 4.1.8 Display of the progress of model fitting. The reporting interval (here, every 3 epochs) is chosen by the user.

The first column indicates the current epoch (iteration) through the dataset. Epoch 0 indicates the performance after initial randomization of weights, prior to any optimization.

The header of the second column changes according to the method of optimization being used— here, the default is a globally adaptive gradient descent learning rate, so the column reports its current value. The third column reports the current value of the average square weight (excluding the bias weights). Here, the final AvSqWt is about 97 (obtained by moving the decimal point 3 places to the left).

The next 2 columns report the penalized and unpenalized objective function errors. Here, optimization monotonically improves the error, and becomes asymptotic after about 24 epochs (iterations) through the training dataset. The penalty is minimal, reflecting the default WeightDecay of -0.001, a value that *doesn't* provide substantial regularization— it's an ad hoc way to prevent locking up of sigmoidal hidden units in nonlinear models.

The last column shows the fraction of cases that fell outside the ScoreThreshold*range-of-the-output-variable. Asymptotically, the linear model ended up with 30% error (that is, 70% of the predictions were within 4.5 years of their targets).

STEP 9. REVIEW THE RESULTS. Next, the relevance of the two predictors is displayed (which assumes the predictors are on a similar scale). This is useful for comparison with in nonlinear models, especially using ARD (section 7.7).

```
##### Summary of Input Relevance Determination
#####
              ( sum square weights of ith input group)
Relevance formula: R_i = -----
              ( sum square weights of all input groups)

Input   Relevance
^^^^^^  ^^^^^^^^^
1       65.24%
2       34.76%
##### End of Input Relevance Report
#####
```

Figure 4.1.9 Display of simple relative relevance of (standardized) weight groups connected to each input for the linear model.

The next section reminds you about files saved to disk.

```
*-----*
-----*
Returning last trained network from Epoch 60.
... Weights saved to "myfirst.wts".
... Predictions saved to "myfirst.ptr".
```

Figure 4.1.10 Display at completion of NevProp3 run, confirming the creation of weights and prediction files.

Lastly, summary statistics are provided. Here, the linear model R2 (.918) is already high. Any increments in parametric and predictive inference we can expect to obtain by modeling the nonlinear interaction will be small– which shows that biologically important interactions can easily be obscured by linear main effects.

```
On TRAIN SET (nCases=20; nOutputs=1):
ASqEr= 16.4 R2=0.918 AThEr=0.300

Key to indexes:
^^^^^^^^^^^^^^
ASqEr: joint measure of error in calibration & discrimination
(BEST=0).
R2: 0-1 measure of uncertainty explained by the model (BEST=1).
(A linear transformation of the ASqEr.)
AThEr: fraction of predictions beyond ScoreThreshold*[target range]
(BEST=0).

##### Leaving NevProp...
#####
```

Figure 4.1.11 Display at completion of NevProp3 run, providing summary statistics appropriate for the type(s) of dependent variable(s).

STEP 10. INSPECT THE WEIGHTS FILE. Using an editor or word processor, open the newly created file, myfirst.wts:

EpochSaved	40		
3	0	5.34982	
3	1	0.0774959	
3	2	15.9992	

Figure 4.1.12 Partial contents of a weights file created by NevProp3. The first line indicates the epoch at which the weights were saved. Next are a pair of indexes indicating the connection of units, followed by the weight of that connection.

Here, the output unit (3) is connected to the bias unit (0) with a weight of about 5.3, to the cholesterol input (1) with a weight of 0.077, and to the age input (2) with a weight of 16. The weights shown are on the scale of the raw data, *not* the standardization performed on-the-fly in NevProp3. This allows the .wts file to be uploaded for future prediction or optimization.

STEP 11. COMPARISON WITH GLM STATISTICAL PACKAGE. For comparison purposes, here is the output of a statistical package (basically same for Systat, SAS, Minitab, and SPlus). Notice that the R2 and weights values are the same as that estimated by NevProp3:

DEP VAR:	DAYS	N:	20	MULTIPLE R:	0.958	SQUARED MULTIPLE R:	0.918
ADJUSTED SQUARED MULTIPLE R:	0.909	STANDARD ERROR OF ESTIMATE:	4.395				
VARIABLE	COEFFICIENT	STD ERROR	STD COEF	TOLERANCE	T	P(2 TAIL)	
CONSTANT	5.350	2.505	0.000	.	2.136	0.048	
CHOL	0.078	0.007	0.774	1.000	11.154	0.000	
AGE	16.000	1.965	0.565	1.000	8.141	0.000	
ANALYSIS OF VARIANCE							
SOURCE	SUM-OF-SQUARES	DF	MEAN-SQUARE	F-RATIO	P		
REGRESSION	3682.500	2	1841.250	95.343	0.000		
RESIDUAL	328.300	17	19.312				

Figure 4.1.13 Output of linear regression performed by a GLM statistical package.

Because NevProp3 does not make distributional assumptions, P values are not routinely displayed. Likewise, a formula for adjusting the R2 based on the relative number of parameters and cases is not assumed. If you'd like NevProp to generate resampling-based estimates of these distributions, re-run the model specifying the .net file setting `NBoots>0` to obtain model standard errors (sections 6.1, 7.8.4), and additionally specifying the setting `NEffectBoots>0` to obtain predictor confidence intervals (sections 6.2, 7.8.5).

Briefly here, "`NBoots 100`" would cause NevProp3 to repeat the entire optimization process using 100 bootstrapped samples of the original training dataset. Here are the results:

```
***** CONVERGENCE OF CUMULATIVE BIAS (of R2 index)
*****

Total number of boots: 100
Maximum Cumulative Mean Bias = 0.049
```

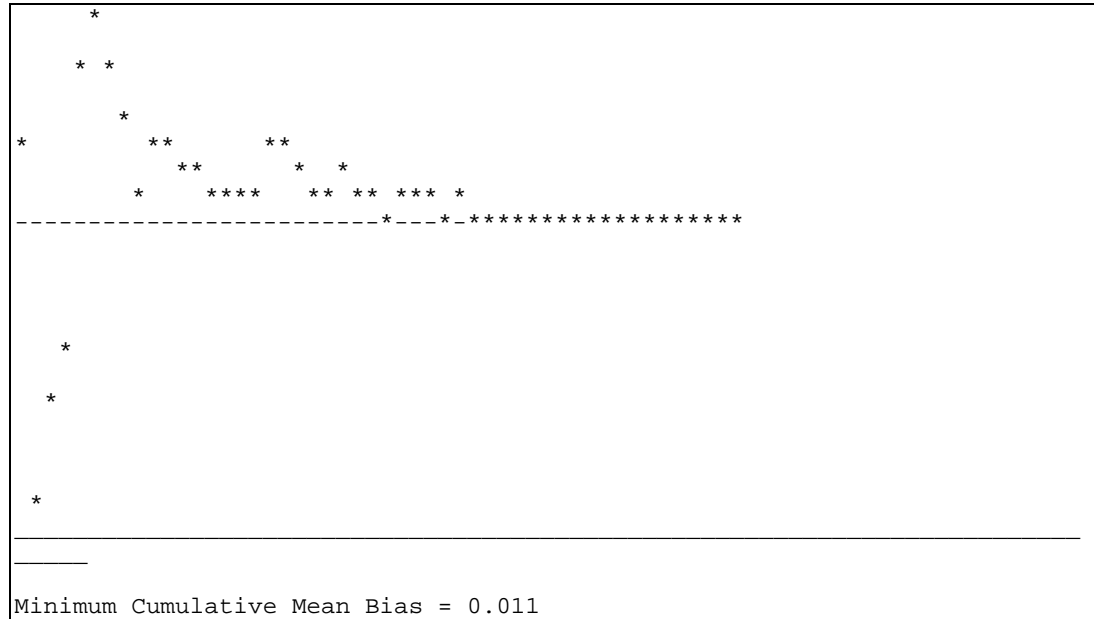


Figure 4.1.14 Convergence of the bootstrapped estimate of R2 bias after about 60 of the 100 booted models.

The NevProp3 graphic above shows adequate stabilization of the resampling-based estimate of R2 bias to justify the following estimation:

***** BOOTSTRAP RESULTS						

Number of boots: 100						
		(A)	(B)	(C)	(B-C)	A-(B-C) (of B)
INDEX	Model:	FULL	BOOT	BOOT		CORR. STD.
^^^^^^	Data:	FULL	BOOT	FULL	BIAS	INDEX ERROR
		^^^^^	^^^^^	^^^^^	^^^^^	^^^^^
AvSqEr		16.420	13.540	20.031	-6.491	22.911 3.9438
R2		0.918	0.932	0.900	0.032	0.886 0.0197

Figure 4.1.15 Summary statistics with bootstrapped-based corrections due to optimistic bias. This summary appears when the setting NBoots is used in the .net file.

The adjusted estimate of AvSqEr is 22.9, a more conservative value than the residual mean square error estimated by the statistical package (19.3). Likewise, the adjusted R2 is 0.886 is more conservative than suggested by the 0.909 ad hoc adjustment of the statistical package. That is, with repeated sampling, NevProp3 indicates that the original linear model may perform less well than the GLM package suggests. The overall model association is very statistically significant (the lower bound 95% confidence interval for R2 would be about $(0.886 - 1.96 \cdot 0.0197) = 0.847$).

If you add the setting “NEffectBoots 1” and rerun the model, NevProp3 would display net effect computations based on exact differential equations of effect (sections 2.4, 6.2, 7.8.5), across all bootstrapped models. The mean effects module is discussed further in STEPS 14 and 16. This allows computation of a population-based confidence interval. Here are excerpted results:

```
##### MEAN EFFECTS REPORT
#####

DEPENDENT VARIABLE (output) # 1 : "days" (nonbinary)
BOOTSTRAPPED SAMPLES used to derive CI's (NBoots): 100
SUB-BOOTSTRAPS used per MODEL (NEffectBoots): 1
TOTAL NUMBER OF BOOTS used for each CI: 100 x 1 = 100

VARIABLE*      Mean      95.% Confid. Interval | Nonlin-   Mean
                Effect    for mean effect      earity    Predict
                ^^^^^^   ^^^^^^^^^^^^^^^^^^ | ^^^^^^   ^^^^^^
1. chol        0.0775                                0
Over all boot models                                ( -2.3308, 2.4858 )
-----
2. age         15.999                                0
Over all boot models                                ( 13.883, 18.116 )
-----
##### END OF MEAN EFFECTS REPORT
#####
```

Figure 4.1.16 Summary statistics with bootstrapped-based confidence intervals for the mean effects of predictors in the linear model. This summary appears when the setting NEffectBoots is used (and booted estimates when NBoots also used) in the .net file.

STEP 12. RUN A NONLINEAR ANN. As suggested in figure 1.1, if a suitably configured artificial neural network does not outperform the corresponding linear model, then there’s no point in searching for interactions (or nonlinearities in nondichotomous predictors).

Modify myfirst.net by:

(a) adding Connect statements to specify a linear network to which is added three sigmoidal hidden units which capture nonlinearity in the effects (if no Connect calls were specified, NevProp3 would have created the 3 hidden unit net without linear shortcuts– see 7.4.6; if no NHidden was specified, NevProp3 would have assigned it a value of 0.5*number of inputs– see 7.4.2);

(b) changing AutoTrain to YES (or delete the setting, because the default is YES, with a PercentHoldout of 50 and NSplits of 5);

(c) adding the SaveWeightsFile statement, so you can upload saved weights at a later time;

(d) adding the SaveTrainPrdFile statement, so you inspect saved predictions;

(e) adding the `NBoots` setting to create bootstrapped models; and,

(f) adding the `UseARD` setting to get a final estimate of the relevance of predictors and the effective degrees of freedom in the model:

```
# The following adds effect nonlinearities
# to the original linear model, and early stopping
# to achieve regularization.

SaveWeightsFile myfirst.wts
SaveTrainPrdFile myfirst.ptr

Ninputs 2           Nhidden 3           Noutputs 1

Connect 1 2 6 6 /* linear, or "shortcut" connections */

Connect 1 2 3 5 /* connects inputs to 2 hidden units */
Connect 3 5 6 6 /* connects hidden to output unit */

AutoTrain YES      PercentHoldout 50      NSplits 5
/* the line above could be deleted -- it's the default */

ScoreThreshold 0.1      NBoots 100      UseARD LastEpoch

NHeaders 1           IDColumn YES

DATA
ID      chol      age      days
A       100       0       10
B       200       0       20
C       300       0       30
D       400       0       40
E       500       0       50
F       100       1       35
G       200       1       40
H       300       1       45
I       400       1       50
J       500       1       55
AA      100       0       8
BB      200       0       14
CC      300       0       36
DD      400       0       32
EE      500       0       46
FF      100       1       33
GG      200       1       36
HH      300       1       41
II      400       1       54
JJ      500       1       57
```

Figure 4.1.17 Revised 'myfirst.net' network file, for nonlinear ANN. Can be copied and pasted into an active NevProp3 directory.

How do we choose the number of hidden units? Conceptually, the shortcut connections can model linear effects. For the interaction, when $\text{age}=0$, weights to the hidden units should be such that no signals from *chol* are passed along to the output. When $\text{age}=1$, however, the signal from *chol* should be allowed to pass a stable effect through the hidden units to the output. *If* we knew in advance to expect a *pure* multiplicative interaction of *pure* linear effects with

normal error variance, as in our toy problem, a single hidden unit could serve as a “gate”: when age=0, a large bias would lock the sigmoid at a constant output insensitive to the chol signal; when age=1, the bias’ weight is cancelled, allowing the sigmoid to operate close to its nearly-linear central portion for the incoming chol signal.

In fact, if we run the NevProp3 model with 1 hidden unit, here’s the summary (because the model is so restricted, it doesn’t matter how the initial weights are randomized):

On TRAIN SET (nCases=20; nOutputs=1): ASqEr= 7.82 R2=0.961 AThEr=0.0500
--

Figure 4.1.18 Summary statistics for nonlinear ANN model with 1 hidden unit.

But if we already *knew* the chol and age effects to be linear with multiplicative interaction, predictions would have less variance if we merely created a third variable equal to chol*age, and run a linear model. Here’s the output from GLM statistical package for such a model:

DEP VAR:	DAYS	N:	20	MULTIPLE R:	0.978	SQUARED MULTIPLE R:	0.956
ADJUSTED SQUARED MULTIPLE R:	0.948	STANDARD ERROR OF ESTIMATE:	3.319				
VARIABLE	COEFFICIENT	STD ERROR	STD COEF	TOLERANCE	T	P(2 TAIL)	
CONSTANT	-0.500	2.461	0.000	.	-0.203		0.842
CHOL	0.097	0.007	0.969	0.500	13.072		0.000
AGE	27.700	3.480	0.978	0.182	7.959		0.000
CHOL*AGE	-0.039	0.010	-0.496	0.154	-3.716		0.002

Figure 4.1.19 Corresponding output of GLM statistics package.

Notice that the 1-hidden unit ANN still did slightly better, only because it does not assume a purely linear effect for each level of age– notice, below, that a shallow sigmoid actual fits the age=1 curve. Of course, if we generated more cases using a normal residual error model, this curvature would go away.

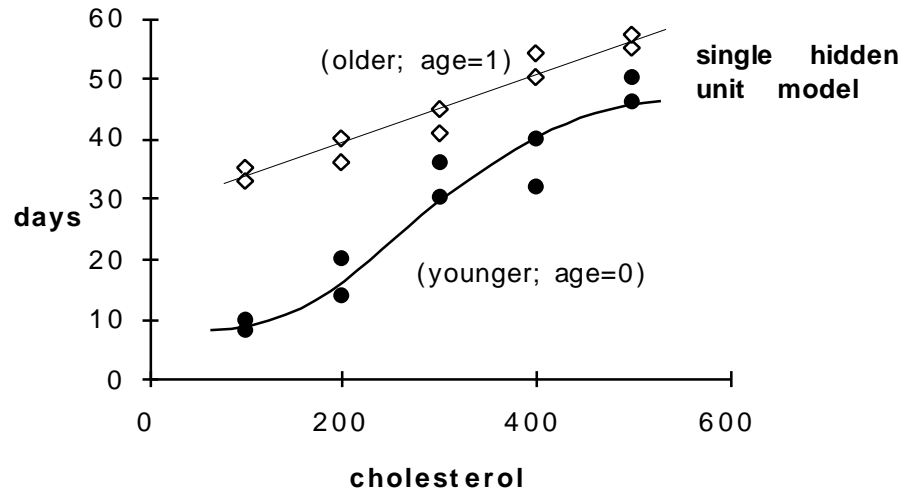


Figure 4.1.20 Single hidden unit fit of data.

NevProp3 can demonstrate that the sparse data do not justify even this mild degree of overfit. The bootstrap results reveal a large optimistic bias:

***** BOOTSTRAP RESULTS *****						

Number of boots: 100						
	Model:	(A)	(B)	(C)	(B-C)	A-(B-C) (of B)
INDEX	Data:	FULL	BOOT	BOOT	BIAS	CORR. STD.
^^^^^^	^^^^	^^^^	^^^^	^^^^	^^^^	^^^^
ASqEr		7.826	5.595	13.965	-8.370	16.195 2.4776
R2		0.961	0.972	0.930	0.042	0.919 0.0124

Figure 4.1.21 Summary statistics for single hidden unit ANN with bootstrapped-based corrections due to optimistic bias.

Returning now to reality... with complex data we want to make only the appropriate effect and distributional assumptions. The approach we recommend is to supply extra flexibility, and use a regularization method to prevent the weights from growing inappropriately large.

Overfit can occur even in the fitting of GLMs—ridge regression was developed to penalized large weights during that optimization. NevProp3 offers the option of a nonlinear extension of ridge regression (*WeightDecay*) as well as a hyperparameter that adjusts the penalty for groups of units, according to number of well-determined parameters in groups (*UseARD*). However, the simplest regularization option, is to use a form of initial cross-validation to determine when to stop optimization of a model using all the data. In NevProp3, we call this option 'AutoTrain.'

Let's now fit an ANN with extra flexibility, regularized with AutoTrain. The directed graph of such a network can be visualized as follows:

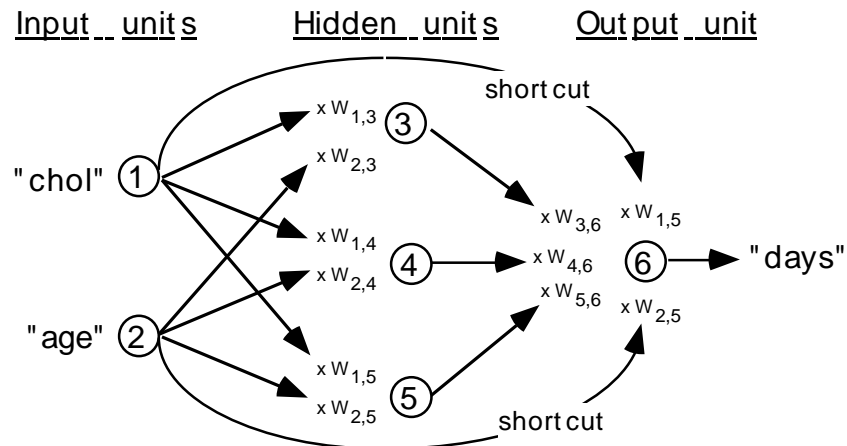


Figure 4.1.22 Directed graph of a 3-hidden unit nonlinear ANN model. Weights, w , are multiplied by signals from corresponding units. At convergence to hidden unit nodes, sums of weighted signals are applied to a sigmoidally-shaped activation functions.

Run this ANN using command-line initiation:

```
np myfirst.net 500 10 1234
```

STEP 13. OBSERVE THE AutoTrain PROCESS. By default, NevProp3 uses the AutoTrain mode of operation to prevent overfitting of the data (see section 7.6.4). In Phase I of AutoTrain, NevProp3 fits a model using the first PercentHoldout (here, 50%) of the cases (train subset) while comparing the model's predictive performance on the remainder (the hold-out subset). NevProp3 stores the train-subset error at the nadir of the hold-out error. The default is to use 5 splits of the data (NSplits 5) to reduce the variance inherent in this type of "cross validated" error estimation. Here's the display from the first of the 5 splits:

(Note: The actual numeric results will depend on how your computer's pseudo-random number generator initialized the weights.)

Start of Phase I: Find TRAIN-SUBSET error at best HOLDOUT-SUBSET criterion.								
			___TRAIN-SUBSET__(10)___			___HOLDOUT-SUBSET__(
10)___								
Train	LrnRat	AvSqWt	APnSqEr	ASqEr	AThEr	APnSqEr	ASqEr	
AThEr								
Epoch	(e-3)	(e-3)	(e-3)	(e-3)	(e-3)	(e-3)	(e-3)	(e-3)
-----	-----	-----	-----	-----	-----	-----	-----	---
--								
0	10.000	2.4391	1654725	1654725	1000.	1434759	1434759	
1000.								
10	13.297	17184.	86427.9	86426.9	600.0	204454.	204453.	
800.0								
20	21.660	24276.	12733.1	12731.7	200.0	38382.1	38380.7	
500.0								

```

30 35.281 23681. 9553.56 9552.26 200.0      21518.7 21517.4
400.0
40 57.469 22261. 8179.44 8178.21 100.0      15282.4 15281.2
300.0
50 93.611 21897. 7461.63 7460.43 100.0      12962.2 12961.0
200.0
-- lines omitted for brevity --
100 124.03 25422. 13551.6 13550.3 100.0      11234.8 11233.4
300.0
-- lines omitted for brevity --
150 76.035 29704. 5495.18 5493.55 100.0      17481.8 17480.2
300.0
-- lines omitted for brevity --
200 74.017 32143. 5170.31 5168.55 0.000      17816.7 17814.9
300.0
210 75.925 32397. 5163.80 5162.02 0.000      17721.7 17719.9
300.0
Done. Epoch 210 is beyond 2 X BestEpoch 100 (and MinEpochs).

(Up to) 10 BEST reports on HOLDOUT-SUBSET using Best-By criterion
graphed:

Epochs  APSE (e-3)  BEST      <---- APnSqEr ---->      WORST
-----
40 15282.448 *****
50 12962.217 *****
.....
70 14986.301 *****
80 12881.645 *****
90 11934.252 *****
100 11234.804 *
110 12500.533 *****
120 14941.847 *****
130 16651.411 *****
140 17139.599 *****

.....=>Skips at least one reporting period.
*-----*
-----*
For split #1 of 5, target error = 13.551650 from Epoch 100.

```

Figure 4.1.23 Display of the progress of model fitting during phase I of AutoTrain (see text). The graphic displays pattern of holdout-subset error during fitting of model using train-subset data.

Both the epoch-wise numeric display and the summary graphic that Average Square Error (ASqEr) monotonically declined in the train-subset, but the ASqEr on the holdout-subset reached a nadir at about 100 epochs, which corresponded to the train-subset's APnSqEr of about 13.6— this number, when averaged with those from the other splits— becomes the mean target for phase II early stopping, which was reached at epoch 40:

```

*-----*
-----*
Result of Phase I (5 splits):
Avg TRAIN_SUBSET target 8.82428 from average of 240 Epochs of
training.

*-----*
-----*
Start of Phase II: Re-train on entire TRAIN-SET to target error above.
_____TRAIN-SET_( 20)_____
Train LrnRat AvSqWt APnSqEr ASqEr AThEr
Epoch (e-3) (e-3) (e-3) (e-3) (e-3)

```

```

-----
      0 10.000 2.4391 1544742 1544742 1000.
     10 13.297 18828. 16150.4 16149.9 200.0
     20 21.660 20010. 10210.7 10210.2 150.0
     30 35.281 19918.  8914.22 8913.67 100.0
     40 57.469 20820.  8490.38 8489.81 100.0
REACHED TARGET ERROR = 8.824282 at Epoch 40

```

Figure 4.1.24 Display of the progress of model fitting during phase II of AutoTrain (see text).

STEP 14. INSPECT NONLINEAR EFFECTS. Because you specified that ARD be run after the last epoch of training, the variance-adjusted relevance is reported:

```

===== Summary of Automatic Relevance Determination
=====
ARD was invoked in LastEpoch mode (i.e., following non-ARD
optimization).
This permits an estimate of explanatory variable relevance, and the
final effective number of well-determined parameters fit.

Iterations of Hessian/inversion used: 50
(maximum iterations internally set at MaxEpochs*1= 1000)
No. of well-determined parameters in the model: 8.5, or 56%, of 15
total.

GROUP   FROM   TO      ARD HYPERPARAMETERS      # Well-determined
Parameters
^^^^^^  ^^^^   ^^      ^^^ ^^^^^^^^^^^^^^^^^^   ^ ^^^^^^^^^^^^^^^^^^
^^^^^^^^

0        bias   hidden          1.09          1.3, or 44%, of 3
1        input1 hidden          0.86          2.5, or 63%, of 4
                output1
2        input2 hidden          1.41          2.2, or 55%, of 4
                output1
3        hidden output1          0.06          2.4, or 59%, of 4
        bias

FYI: Because GroupSelection is INPUT, weights of shortcuts from
input to output units were assigned to input units' group.
##### Summary of Input Relevance Determination Based on ARD
#####
ARD Relevance formula: R_i = (1.0/alpha_i)/(sum of all 1.0/alpha_i),
                        where alpha_i is the ARD hyperparameter for the ith input

Input    Relevance
^^^^^^   ^^^^^^^^^
1         62.21%
2         37.79%
##### End of Input Relevance Report
#####
===== End of ARD Report
=====

```

Figure 4.1.25 Display of ARD variance-adjusted relative relevance of weights connected to each input (predictor) for the nonlinear model.

We find that the weights arising from both inputs (groups 1 and 2) each use slightly more than half a degree of freedom (63% and 55%, respectively). Also, overall, only 8.5 (56%) of the potential degrees of freedom have been used by the ANN model.

If we run the same model, but set “UseARD FULL” (so that an input-specific, variance-based penalty is applied throughout optimization), we find that only 6.5 (43%) effective weights are used, without any impairment of performance. ARD is discussed further in sections 6.3 and 6.4.

We can also obtain estimates of the 95% confidence interval for the mean effect of each predictor rerunning the ANN with “NEffectBoots 2000” specified. Alternatively, you could loading the existing weights (add the setting “ReadWeightsFile myfirst.wts”) and executing NevProp3 as:

```
> np myfirst.net 0 0
```

Here is an excerpt from the new summary material displayed:

MEAN EFFECTS REPORT				
VARIABLE*	Mean Effect	95.% Confid. Interval for mean effect	Nonlinearity	Mean Predict
^^^^^^	^^^^^^	^^^^^^^^^^^^^^^^^^^^	^^^^^^	^^^^^^
1. chol	0.0647	(0.0460, 0.0816)	0.6439	36.603

L5 (500.)	0.0763	(0.0682, 0.0844)	0.1223	51.920
L4 (400.)	0.0806	(0.0685, 0.0927)	0.1729	43.691
L3 (300.)	0.0301	(-0.0125, 0.0726)	1.6333	38.279
L2 (200.)	0.0605	(-0.0061, 0.1270)	1.2707	27.697
L1 (100.)	0.0759	(0.0684, 0.0834)	0.1137	21.465

2. age	11.917	(9.8564, 14.102)	0.4397	36.603

L2 (1.00)	10.287	(8.7054, 11.730)	0.2601	44.495
L1 (0.00)	13.546	(9.7302, 17.311)	0.4947	28.677
* L = level# (value at that level); Q# = quartile# (mean value).				
##### END OF MEAN EFFECTS REPORT				
#####				

Figure 4.1.26 Summary statistics with bootstrapped-based confidence intervals for the mean effects of predictors in the nonlinear model.

The single best estimate of the effect of cholesterol is .0647 days extra hospital stay per unit increase in cholesterol (or, about 6.5 days per 100 units increase in cholesterol), with a reasonably narrow CI. The effect of older age is to increase hospital stay a mean of about 12 days.

Both these effects are smaller in magnitude than the effects obtained from the pure linear model (figure 4.1.14). It would be nice to compare these effects to the GLM with the `chol*age` interaction term, but there is no way to properly combine the main and interaction effects to obtain an overall statistic. In other words, the GLM with interaction is less interpretable than the ANN! The reason for this is that the GLM multiplicative interaction is totally artificial—there is no mechanism for an independent explanatory variable like this in the physical world.

The NevProp3 mean effects display provides additional inference about the model, as described in section 6.2.

STEP 15. INSPECT PREDICTIONS WITH CONFIDENCE INTERVALS. Because you specified the bootstrapped modeling, the predictions file saves not only the prediction for each case, but it's 95% confidence interval. Here is the file, "myfirst.ptr":

SEQU	PRED1	LOWER951	UPPER951	TRUE1
DD	36.95	31.97	41.93	32
E	46.66	41.45	51.87	50
D	36.95	31.97	41.93	40
HH	43.56	40.46	46.67	41
G	37.91	35.07	40.75	40
J	57.49	54.53	60.46	55
AA	9.322	5.498	13.14	8
EE	46.66	41.45	51.87	46
I	50.61	47.94	53.28	50
B	17.10	12.35	21.85	20
II	50.61	47.94	53.28	54
CC	32.95	27.72	38.18	36
H	43.56	40.46	46.67	45
C	32.95	27.72	38.18	30
A	9.322	5.498	13.14	10
F	33.43	26.60	40.27	35
FF	33.43	26.60	40.27	33
JJ	57.49	54.53	60.46	57
GG	37.91	35.07	40.75	36
BB	17.10	12.35	21.85	14

Figure 41.27 Saved predictions file, based on cases used to train the model. A file of predictions could also be created for test (validation) data, if provided.

If bootstrapped models had not specified been in the .net file, only the mean predictions would have been saved. The sequence reflects shuffling that occurred after NevProp3 originally read the data (the default is "ShuffleData YES").

STEP 16. LINEAR vs NONLINEAR MODEL PERFORMANCE. Note that the *uncorrected* R2 of this nonlinear model (0.970) is much higher than that of the linear model (0.918). Even with regularization, however, we have little confidence that a nonlinear model based on 20 data cases reliably reflects its performance on future data. Therefore, we again use bootstrapping to correct the R2 estimate and obtain its standard error and find that the corrected R2 (0.932) is many standard deviations above that of the linear model (0.886):

***** BOOTSTRAP RESULTS *****						

Number of boots: 500						
	(A)	(B)	(C)	(B-C)	A-(B-C)	(of B)
Model:	FULL	BOOT	BOOT		CORR.	STD.
INDEX	FULL	BOOT	FULL	BIAS	INDEX	ERROR
*****	*****	*****	*****	*****	*****	*****
ASqEr	6.077	5.276	12.880	-7.604	13.681	2.6868

R2	0.970	0.974	0.936	0.038	0.932	0.0134

Figure 4.1.28 Summary statistics for 3-hidden unit ANN with bootstrapped-based corrections due to optimistic bias.

Notice also that the bias-corrected R2 (0.932) is within a standard deviation of the GLM-with-interaction adjusted R2 value (0.948). And overfit was no greater than with the single-hidden unit model (same .05 ScorethresholdError).

Another way to infer *predictor* nonlinearity is again inspect the Mean Effects summary in figure 4.1.27. In addition to the mean effect and its confidence interval are columns containing “Nonlinearity” score and the “Mean Predict.” The former is the coefficient of variation of the mean effect at that level across all cases (zero indicates no variability— a pure linear model). The latter is the average value of the dependent variable for each level of the predictor, based on the final regression model (that is, the marginal predicted value).

The progression of both the mean effect and mean prediction from level 1 (L1, cholesterol = 100) to level 5 (L5, cholesterol = 500) shows an down-up-down pattern, consistent with the sigmoidal fit to the sparse data (figure 4.1.29). Note also that there is substantial nonlinearity (Nonlinearity scores > 1) at L2 and L3, consistent with the slopes at those levels differing across the 2 levels of age.

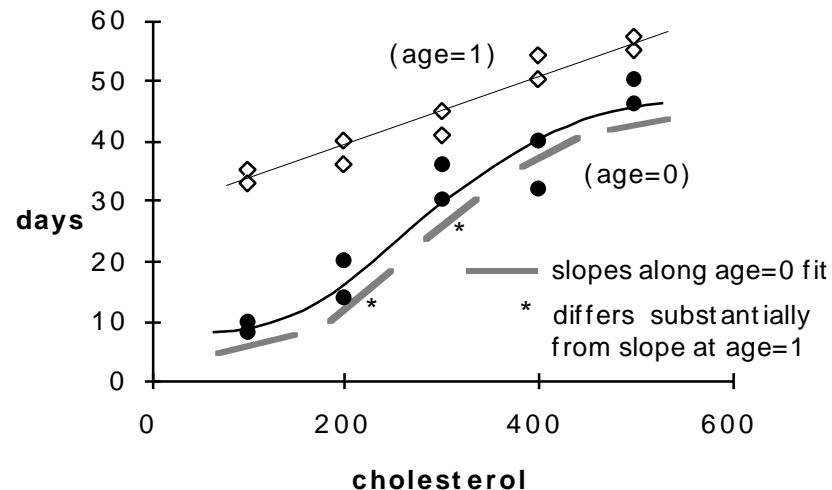


Figure 4.1.29 Heterogeneity of slopes (effects) in nonlinear model.

In this toy problem, we conclude that there *are* substantial interactions, although we do not know to what degree coding nonlinearity in the cholesterol predictor itself may account for the statistically superior performance of the nonlinear ANN.

STEP 17. **REMOVE PREDICTOR NONLINEARITIES.** As indicated above and in figure 1.1, we need to determine to what degree the superior performance of

the ANN was due to nonlinear transformation of the predictors versus nonlinearity in the effects of the predictors (interactions). Why not always use the full nonlinear ANN? Experience and theory support Occam's razor: use the least complex model necessary to explain a phenomenon. Statistically speaking, the variance of our predictive estimates increases with the number of parameters— we want precise predictions, and therefore the simplest model that explains the data.

In our toy problem, we know there is not supposed to be nonlinearity in the cholesterol \rightarrow days relationship. In real data, however, we usually will not be able make such assumptions. In small models, we can do some exploratory analysis, take logarithms of count variables, etc. In larger models, nonlinearities in the predictors can be "straightened out" through several approaches. Here are two: (a) preprocess in a GLM, temporarily making each predictor a dependent variable, and fitting the remaining predictors with splines (e.g., the S-Plus function `transcan`); or, (b) create an ANN with sets of hidden units unique to each nondichotomous predictor (because there are no hidden units common to the predictors, no interaction is coded, but predictor nonlinearity is smoothed).

We can easily modify our nonlinear ANN by severing all connections from age to the hidden units:

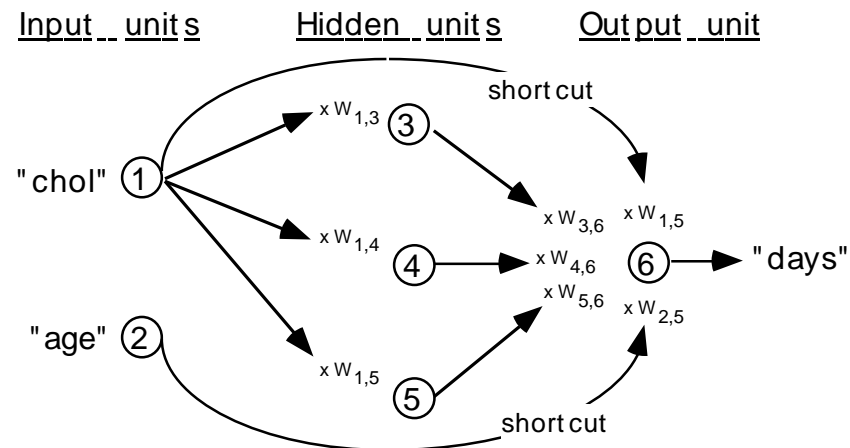


Figure 4.1.30 Directed graph of an ANN that will fit potential nonlinearity in the predictor `chol`, but not interaction with `age`. Generated from figure 4.1.22 by severing the connections from `age` to the hidden units.

The only changes to make in your `.net` file is to the second `Connect` statement:

Connect	1 2	6 6	/* shortcut connections - no change */
Connect	1 1	3 5	/* only input 1 goes to hidden units */
Connect	3 5	6 6	/* hidden to output - no change */

Figure 4.1.31 Revised Connect statements in `myfirst.net`, to fit only the (potential) nonlinearity of cholesterol in predicting days of hospital stay.

Using the same regularization and bias correction methods, we find (figure 4.1.32):

***** BOOTSTRAP RESULTS *****						

Number of boots: 100						
		(A)	(B)	(C)	(B-C)	A-(B-C) (of B)
	Model:	FULL	BOOT	BOOT		CORR. STD.
INDEX	Data:	FULL	BOOT	FULL	BIAS	INDEX ERROR
^^^^^^		^^^^^	^^^^^	^^^^^	^^^^^	^^^^^
ASqEr		15.827	12.469	21.201	-8.732	24.559 3.5641
R2		0.921	0.938	0.894	0.044	0.878 0.0178

Figure 4.1.32 Summary statistics for ANN with 3-hidden units connected only to `chol`, with bootstrapped-based corrections due to optimistic bias.

The adjusted R2 (.878) essentially identical (within a standard deviation) to that of the pure linear model (adjusted R2 .886, figure 4.1.15), and far from that of the fully connected ANN (adjusted R2 .932, figure 4.1.27). Although the unadjusted model R2 (.921) was slightly higher than that in the linear model (.918), the extra (unneeded) flexibility introduced by the nonlinear coding of cholesterol resulted in excessive bootstrap bias correction, to an adjusted R2 below that of the linear model.

STEP 18. CHOOSING THE “BEST” FINAL MODEL— GLM or ANN? We determined above that fitting *parametric interaction*, not predictor nonlinearity, accounted for the superior performance of the fully-interconnected ANN.

Should we then use the full ANN, or the GLM with interactions as our final model? In this 2-predictor problem, we found that inserting the artificial `chol*age` variable into a GLM achieved an adjusted performance similar to that of the full ANN (as it should, based on our design of the interaction).

In the real world, there are 3 reasons to prefer the ANN:

- **ANALYST EFFORT & EFFICIENCY.** Through the use of hierarchical modeling of ANNs we determined that predictor nonlinearity was not substantial— if it had been, the ANN would have modeled the nonlinearity. Making the same determination with a multivariate GLM would have required a number of regression diagnostics— and a variety of ad hoc attempts to linearize suspected relationships. From this perspective, ANNs can simplify and shorten the analyst’s effort by simultaneously modeling predictor and parameter nonlinearities.

- **MULTIPLE COMPARISON ARTIFACT.** Empirically testing 2-way (and higher) interactions in a GLM requires the generation of many models, with the associated risk of finding (false) significant differences due to the multiple comparisons. The concern about excessive parameterization is addressed by using an ANN with proper regularization to limit the effective degrees of freedom (number of well-determined weights).
- **INFERRING COMPLEX EFFECTS.** In the GLM with multiplicative interaction terms, there is no *single* effect of the involved predictors. For example, what is *the* effect of cholesterol in figure 4.1.19? It appears as part of 2 variable's effects. In other words, the simplicity of GLM interpretation is lost when interactions are specified. Using an ANN permits inference on effects from at least two perspectives: first, Bayesian relevance penalties can be used to suggest the overall relative importance of explanatory variables (including nonlinear effects); and second, mean effects computation (section 6.1) can quantify the impact, nonlinearity, and uncertainty the effects explanatory variables. If the goal is truly to model a *complex* system, then the flexibility of the (properly configured) ANN more closely achieves this goal. That is, the ANN model may be more directly informative in the testing of complex scientific hypotheses.

&4.2 Classification: Fisher's Iris Data Set

Time required: About 1 hour

Objectives: Upon completion of the first tutorial...

- Include/exclude specific variables from a dataset
- Create prediction files with 95% prediction intervals
- Supply external training and testing (validation) data to NevProp3
- Interrupt NevProp3 to modify settings on-the-fly
- Impute missing data

SCENARIO. Consider the network file `Iris.net` supplied with NevProp3. In this 3-category classification problem adapted from Fisher, 4 measurements were made on each of 150 flowers: sepal length (`seplen`), sepal width (`sepwid`), petal length (`petlen`), and petal width (`petwid`). We added a 5th predictor, `Zrandom`, which is Gaussian "noise" of mean 0 and units of standard deviation. Under discriminant analysis, the third category of flower (`Species3`) is linearly separable from the other two— among these, only the three cases 16 (`Species1`), 66 and 79 (`Species2`) are not linearly separable.

STEP 1. **IDENTIFY** `Iris.net`, supplied with NevProp3 created . Open the file with a word processor and look it over. The order of the settings is arbitrary (as long as they precede the `DATA` command, after which the 150 internally-supplied rows of data appear).

STEP 2. SUBSETTING of predictor variables. Notice the settings:

```
InputColumns 1 2 3 4
OutputColumns 6
NVars      8
```

Here, we instruct NevProp3 to use the first 4 predictor variables as inputs, ignoring the Gaussian noise variable for now. Only the sixth variable (Species1) is used as an output (dependent) variable; that is, we ignore the binary variables coding Species2 and Species3. Thus, the goal is to design a model that predicts whether or not the flowers belong to Species1.

Because we are subsetting, NevProp3 must be explicitly told how many variables (NVars) constitute a single case (because cases are allowed to extend over any number of lines with carriage returns). If we did not subset, NevProp3 would implicitly assume NVars to be equal to Ninputs plus Noutputs.

If we had many variables, and wanted to exclude just a few, it would be more efficient to place a negative sign before undesired variables, thereby excluding them from the model. Although not desirable here, settings equivalent to those above would be:

```
InputColumns -5 -6 -7 -8
OutputColumns -1 -2 -3 -4 -5 -7 -8
NVars      8
```

STEP 3. RUN A SIMPLE NONLINEAR MODEL. The .net file is set to connect each of the 4 inputs to 3 hidden units, each of which in turn connects to the single output:

```
Ninputs      4           Nhidden      3           Noutputs      1
Connect      1 4      5 7 #all inputs connected to all hidden
Connect      5 7      8 8 #all hidden connected to output
OutputUnitType 3      #3 means automatically determined
```

Because OutputUnitType is set to automatic, NevProp3 will scan the output variable, find that it is binary, and set the output unit activation (inverse link) to be a 0-1 ranged sigmoid. (Alternatively, we could have directly set "OutputUnitType 2".)

Now run the model in command-line mode, using an arbitrary random seed:

```
( np netfilename maxepochs reportinterval randomseed )
  np iris.net      500          25          999
```

STEP 4. CONFIRM THE SETTINGS & INTEGRITY OF THE DATA. Here's what it should look like:

```
goodman@humboldt$ np Iris.net 500 25 999
##### Starting NevProp NP3r1
#####

HEADER #1: This is the first header. The next contains labels for the
variables.

HEADER #2: Flower SepLen SepWid  PetLen  PetWid  ZRandom Species1
Species2 Species3
```

```

... SETTINGS were successfully read from "Iris.net".
... SEED=999 using lrand48(),srand48()
... Read in 150 TRAIN CASES [random 75 cases (50%) to a holdout
subset]

# DATA FILE SETTINGS
  NHeaders 2          IDColumn YES
  StandardizeInputs 1 SaveStandWts NO ImputeMissing median
  InputColumns 1 2 3 4
  OutputColumns 6
  NVars 8            ShuffleData YES
# REPORTING SETTINGS
  DescribeVars YES
  NBoots 0           NEffectBoots 0
  CalccIndex YES     ScoreThreshold 0.5
  OutputStatVars 0
# CONNECT CALLS
  Connect 1 4 5 7
  Connect 5 7 8 8
# CONFIGURATION SETTINGS
  Ninputs 4          Nhidden 3          Noutputs 1
  kNN 0             lofN NO
  HiddenUnitType 1   OutputUnitType 3
/* OutputUnitTypes assigned: 2 */
  WeightRange 0.001
# TRAINING SETTINGS
  TrainCriterion 2
  BiasPenalty NO     WeightDecay -0.001
  OptimizeMethod 1   SigmoidPrimeOffset 0
  QPMaxFactor 1.75   QPModeSwitchThreshold 0
  Stochastic NO      LearnRate 0.01     SplitLearnRate NO    Momentum
0
# BEST-BY-HOLDOUT SETTINGS
  PercentHoldout 50.00
  AutoTrain YES     MinEpochs 50        BeyondBestEpoch 1.5
  NSplits 5         SepBootXVal YES
# RELEVANCE DETERMINATION SETTINGS
  UseARD NO         WhenARD Auto     ARDTolerance 0.05   ARDFreq 25
  GroupSelection Input BiasRelevance NO ARDFactor 1

```

As in the first tutorial, review the descriptive statistics on each output and predictor variable: (If, for future runs, you do not wish to have these stats displayed, set "DescribeVars NO" in the .net file.)

```

##### Descriptive Statistics on Output (Dependent) Variables
#####
-----
variable      n      unique      mean
-----
1. Species1    150      2      0.3333
0.00( 100, 66.7%) 1.00( 50 33.3%)
-----

##### Descriptive Statistics on Input (Predictor) Variables
#####
-----
variable      n      missing      unique      mean
-----
1. SepLen      150      0      35      5.8433
percentile:    0.05  0.10  0.25  0.50  0.75  0.90  0.95
value:         4.6000 4.8000 5.1000 5.8000 6.4000 6.9000 7.3000

```

lowest:	4.3000	4.4000	4.4000	4.4000	4.5000		
highest:	7.7000	7.7000	7.7000	7.7000	7.9000		

variable	n		missing		unique	mean	
^^^^^^^^	^^^^^^^^		^^^^^^^^		^^^^	^^^^	
2. SepWid	150		0		23	3.0573	
percentile:	0.05	0.10	0.25	0.50	0.75	0.90	0.95
value:	2.3000	2.5000	2.8000	3.0000	3.3000	3.7000	3.8000
lowest:	2.0000	2.2000	2.2000	2.2000	2.3000		
highest:	3.9000	4.0000	4.1000	4.2000	4.4000		

variable	n		missing		unique	mean	
^^^^^^^^	^^^^^^^^		^^^^^^^^		^^^^	^^^^	
3. PetLen	150		0		43	3.7580	
percentile:	0.05	0.10	0.25	0.50	0.75	0.90	0.95
value:	1.3000	1.4000	1.6000	4.4000	5.1000	5.8000	6.1000
lowest:	1.0000	1.1000	1.2000	1.2000	1.3000		
highest:	6.4000	6.6000	6.7000	6.7000	6.9000		

variable	n		missing		unique	mean	
^^^^^^^^	^^^^^^^^		^^^^^^^^		^^^^	^^^^	
4. PetWid	150		0		22	1.1993	
percentile:	0.05	0.10	0.25	0.50	0.75	0.90	0.95
value:	0.2000	0.2000	0.3000	1.3000	1.8000	2.2000	2.3000
lowest:	0.1000	0.1000	0.1000	0.1000	0.1000		
highest:	2.4000	2.4000	2.5000	2.5000	2.5000		

##### End of Descriptive Statistics							
#####							

STEP 2. INTERPRET THE OPTIMIZATION & PREDICTIVE STATISTICS. Here's an excerpt of the display (actual numerical results depend on your platform):

##### Start of Model Fitting									
#####									

ABBREVIATIONS KEY:									
LrnRat=LearnRate, FrGrdD=Fraction of weight updates using Grad									
Descent,									
Av=Average, Sq=Squared, Wt=Weights, A=(Average over cases &									
outputs),									
Pn=Penalized, CrEn=Cross-Entropy, Er=Error, Th=Thresholded, 1-c=(1 -									
c index)									

Start of Phase I: Find TRAIN-SUBSET error at best HOLDOUT-SUBSET									
criterion.									
_____TRAIN-SUBSET_(75)_____HOLDOUT-SUBSET_(
75)_____									
Train LrnRat AvSqWt APnCrEn ACrEn AThEr 1-c APnCrEn ACrEn									
AThEr 1-c									
Epoch (e-3) (e-3) (e-3) (e-3) (e-3) (e-3) (e-3) (e-3) (e-3) (e-									
3) (e-3)									

```

-----
--
  0 10.000 0.0005 693.287 693.287 640.0 714.5 693.340 693.340
693.3 811.9
 25 27.644 0.0027 653.430 653.430 360.0 24.31 623.345 623.345
306.7 8.361
 50 93.611 219.96 501.530 501.508 213.3 22.38 449.510 449.488
120.0 10.03
 75 317.00 4869.3 65.0271 64.5402 13.33 0.000 84.5144 84.0274
40.00 1.672
 100 364.89 10672. 53.9811 52.9139 26.67 0.000 78.8181 77.7509
40.00 2.508
 125 360.02 13301. 23.3741 22.0440 0.000 0.000 138.547 137.217
40.00 1.672
 150 355.21 14613. 18.5921 17.1309 0.000 0.000 134.902 133.441
40.00 1.672
 175 350.46 15937. 15.5668 13.9731 0.000 0.000 166.347 164.753
40.00 1.672
Done. Epoch 175 is beyond 1.5 X BestEpoch 100 (and MinEpochs).

```

The .net file specifies that half the training data be temporarily held out ("PercentHoldout 50.00"), and that this "cross-validation" process be done five times ("NSplits 5") after reshuffling the cases before each split. Only the first split is shown above. At the conclusion of the 5 splits, the average target error is reported, and, in Phase II, the entire training set is reassembled to fit a model. Because .net file specifies stopping of optimization when the target is reached ("AutoTrain YES"), we find that training is halted at epoch 100:

```

*-----*
-----*
Result of Phase 1 (5 splits):
Avg TRAIN_SUBSET target 0.0686937 from average of 80 Epochs of
training.

*-----*
-----*
Start of Phase II: Re-train on entire TRAIN-SET to target error above.
              _____TRAIN-SET_( 150)_____
Train LrnRat AvSqWt  APnCrEn  ACrEn  AThEr  1-c
Epoch (e-3) (e-3) (e-3) (e-3) (e-3) (e-3)
-----
  0 10.000 0.0005 693.315 693.315 666.7 763.0
 25 27.644 0.1750 636.324 636.323 333.3 19.40
 50 93.611 2190.5 156.854 156.745 40.00 3.200
 75 171.11 14441. 313.693 312.971 173.3 15.20
100 196.96 14537. 68.2911 67.5642 26.67 2.000

```

Moving the decimal points over 3 places to the left and rounding, the actual final statistics are a learning rate of 0.197, an average squared weight of 14.5, an average penalized cross entropy (-log likelihood) of 0.0683, raw cross entropy of 0.0676, average thresholded error of 0.0267 (that is, only 2.67% of cases did *not* fall within ScoreThreshold*targetrange), and a c index of (1-.002=) 0.998.

These and two other statistics are reported more clearly at the conclusion of the NevProp3 run (see section 5.6 for further details on interpreting these statistics):

```
On TRAIN SET (nCases=150; nOutputs=1):
  ACrEn=0.0676  R2-N=0.944  ASqEr=0.0174  AThEr=0.0267  C index=0.9980

Key to indexes:
^^^^^^^^^^^^^^^^
  ACrEn (avg -log likelihood): max likelihood objective function
(BEST=0).
  Nag-R2: 0-1 measure of uncertainty explained by the model (BEST=1).
          (A monotonic, but nonlinear transformation of the ACrEn.)
  ASqEr (Brier Score): joint error in calibration & discrimination
(BEST=0).
  AThEr: fraction misclassified, outside ScoreThreshold*[dichot range]
(BEST=0).
  C index (area under ROC curve): .5-1 nonparam discrimination measure
(BEST=1).

##### Leaving NevProp...
#####
```

Why did this nonlinear neural net misclassify (.0267*150=) 4 cases? Recall that a linear discriminant analysis “misclassifies” only 3 cases. If you allow NevProp3 to train the current 3-hidden-unit network on the entire dataset (without AutoTrain turned on), it “misclassifies” only 2 cases. Furthermore, if you add a few more hidden units, you can get NevProp3 to “misclassify” none of the cases.

The explanation is that, based on multiple cross-validations to estimate an optimal stopping error, a regression function (hypersurface) was fit that should *generalize* to future data. Folding in the surfaces (i.e., increasing the functional complexity) to capture the 4 cases mentioned above, would potentially result in excessive misclassification on cases yet to be encountered. Remember, the iris flower data is empirical, so that there could have been errors in collection and recording of the original data, and an error in the “true” classification by the taxonomist. After all, defining species is a somewhat arbitrary process, especially when spontaneous mutations are known to occur. It is, therefore, possible that the 4 flowers mentioned above are genetic variants. If Fisher had available a nonlinear classifier that used early stopping, perhaps he would have made inquiries about those 4 flowers to his geneticist friend (the original reason Fisher analyzed the data related to establishing genetic linkages).

STEP 3. INTERPRET THE EFFECTS.

Now inspect the file `Iris.wts`. For the moment, skip to the bottom of the file. Here, you will see a summary of the “relevance” of each predictor:

```
UseARD NO
```

```

##### Summary of Input Relevance Determination
#####
Relevance formula: R_i = ( sum square weights of ith input group)
                      ( sum square weights of all input groups)

Input   Relevance
^^^^^   ^^^^^^^
1       2.85%
2       0.54%
3       35.19%
4       61.42%
##### End of Input Relevance Report
#####

```

Because the inputs were standardized to roughly similar scales ("StandardizeInputs YES"), the relevance is a useful comparison of the geometric mean weight magnitudes for each predictor. Because ARD (see section) was not used to optimize the model, the relevance here reflects only final fitted weight magnitudes, and not the influence of weight variances computed during optimization.

You may have noticed that the table of relevances is also displayed to standard output and the results file (*Iris.res*) during the NevProp3 run.

Per unit on the scale of standard deviations, the first 2 variables (SepLen and SepWid) have relatively little impact on the predictions. To confirm that sepal measurements do not meaningfully contribute in the linear model, you may wish rerun a model, selecting only predictors 3 and 4. The summary statistics will be essentially unchanged, confirming the irrelevance of the first 2 predictors. The *Iris.net* settings would be modified from Step 2, above, as follows:

```

InputColumns 3 4 #first 2 predictors no longer desired
OutputColumns 6 #no change; target is still 6th variable
NVars      8    #no change; still have 8 in dataset

Ninputs      2      Nhidden      3      Noutputs      1
Connect      1 2    3 5 #indexes revised to reflect no. vars
Connect      3 5    6 6 #indexes revised to reflect no. vars
OutputUnitType 3    #no change

```

Inspecting the weights themselves is not useful unless they (that is, unless the input variables) are on a similar scales. Because the scales of the iris flower measurements differ substantially, it is necessary to tell NevProp3: (a) to standardize the predictors ("StandardizeInputs YES"), which was done, and, (b) to save the weights on that standardized scale ("SaveStandWts YES"), which was not the case in the run, above.

For example, here are the weights, saved from our run:

EpochSaved	100		
5	0	37.9871	
5	1	-2.29138	
5	2	0.110295	
5	3	-3.00499	
5	4	-8.90654	
6	0	-28.9481	

6	1	0.804246
6	2	0.502287
6	3	2.46424
6	4	7.78723
7	0	6.81241
7	1	0.948955
7	2	2.08931
7	3	-1.84204
7	4	-5.7594
8	0	-2.55329
8	5	-3.4112
8	6	3.53442
8	7	-5.4909

The weights are indexed from receiving-to-sending index of the unit (unit “0” is always the constant bias unit). In this network, we have 4 inputs (indexed 1-5), 3 hidden (indexes 5-7) and one output unit (index 8). Here are the weights in that file:

Compare these with the following weight set, saved from a run that was identical except for the change to (“SaveStandWts YES”):

5	0	2.96032
5	1	-1.8974
5	2	0.0480744
5	3	-5.30469
5	4	-6.78892
6	0	-4.11292
6	1	0.665965
6	2	0.218932
6	3	4.3501
6	4	5.93574
7	0	4.91536
7	1	0.785792
7	2	0.91067
7	3	-3.25175
7	4	-4.39004
8	0	-2.55329
8	5	-3.4112
8	6	3.53442
8	7	-5.4909

It is now clear that the magnitudes of the weights from inputs 3 and 4 (PetalLength and PetalWidth) to all 3 hidden units are much larger than those from the first two inputs. (It is from these standardized weights that the simple relevances are computed, even when SaveStandWts is set to NO.)

Notice that the output unit (index 8) takes negative weights of similar magnitude from hidden units indexed 5 & 7. However, these hidden units take weights of opposite sign from inputs 3 & 4, so they are not obviously redundant. Even on this simple network, it is not too revealing to make any inferences about the effects of the predictors, because of the nonlinearity in effects. A better way is to use mean nonlinear effects, as demonstrated in section 6.2.

STEP 4. **CREATE PREDICTION FILES WITH 95% CONFIDENCE INTERVALS.**

In Step 2, above, we interpreted the summary statistics, generated across all predictions. In many applications, it is important to make point estimates of predictions on individual cases (especially on new data not used in the

training process). At this point, please review the explanation and example in section 5.4.2.

The Iris.net file included a setting to save the predictions to a file ("SaveTrainPredFile Iris.ptr"). Here is an excerpt from that file:

SEQU	PRED#1	TRUE#1
125	0.000155	0
44	0.677528	1
144	0.000155	0
89	0.000182	0
54	0.000307	0
112	0.000155	0
127	0.000155	0
119	0.000155	0
8	0.959143	1
126	0.000155	0
139	0.000155	0
113	0.000155	0
13	0.909896	1
59	0.000466	0
31	0.704852	1
30	0.971595	1
...		
17	0.394228	1
66	0.766237	0
72	0.555601	0
79	0.716780	0

These point estimates are generally very close to the true target category values. Notice that the sequence of the cases reflects the shuffling of the data specifies by "ShuffleData YES". For your interest, the 4 values that fell outside the Scorethreshold are included in bold.

In most applications, it is valuable to have a sense of certainty, or confidence, in the point estimates. In linear models, under assumptions of Gaussian distribution of errors, it is common to compute a standard deviation and use that value to synthesis confidence limits. In generalized nonlinear models, however, such assumptions are not usually valid. It is necessary to use other techniques to compute prediction intervals. NevProp3 uses bootstrapped datasets to create multiple predictions for each case. To turn on this feature, rerun the earlier model, now setting:

NBoots 50 #Booted models to get confidence intervals

The run time will be considerably longer than before, because a total of 51 models will now be fit. Here is the excerpt from the newly created Iris.ptr:

SEQU	PRED#1	LO95CI#1	UP95CI#1	TRUE#1
125	0.000155	0.000147	0.000163	0
44	0.677528	0.028444	0.993411	1
144	0.000155	0.000147	0.000163	0
89	0.000182	0.000099	0.000336	0
54	0.000307	0.000205	0.000461	0
112	0.000155	0.000147	0.000163	0
127	0.000155	0.000147	0.000163	0
119	0.000155	0.000147	0.000163	0
8	0.959143	0.830358	0.991196	1
126	0.000155	0.000147	0.000163	0
139	0.000155	0.000147	0.000163	0
113	0.000155	0.000147	0.000163	0
13	0.909896	0.601554	0.985411	1

59	0.000466	0.000125	0.001731	0
31	0.704852	0.082370	0.984505	1
30	0.971595	0.889115	0.993193	1
...				
17	0.394228	0.041179	0.907930	1
66	0.766237	0.162674	0.982239	0
72	0.555601	0.103982	0.930887	0
79	0.716780	0.130458	0.977112	0

For point estimates very close to their targets, the confidence intervals are, in general, quite narrow. As you might expect, the uncertainty increases as the point estimates deviate from their targets. Notice the extremely wide confidence intervals for the four “misclassified” cases. Clearly the data provides little statistical evidence for an accurate assignment of these cases. See the next Step for making interval estimates for test set data.

Running with booted models also generates estimates of bias-corrected model accuracy. See section 6.1 for discussion and results on the Iris dataset.

STEP 5. USING EXTERNAL TRAINING & TESTING DATA. (that is, data not supplied within your .net file)

EXTERNAL TRAINING DATA. Make a copy of `Iris.net`, and rename it as `Iris.trn` (the .trn stands for *train* data). Now delete all lines preceding the data, including the line with the `DATA` command. The resultant file should begin with the headers, and end with the last line of data. See section 5.3.2 for details on the formatting, which is very flexible.

Now open the `Iris.net` file, and delete the `DATA` command and all lines of data below it. Enter the following setting and argument:

```
ReadTrainFile  Iris.trn
```

This setting tells NevProp3 to expect training data to be contained within the external file `Iris.trn`.

Now rerun the analysis above. The results should be identical, indicating that you successfully provided an external dataset to NevProp3.

EXTERNAL TESTING DATA. If you have new cases to test, you can instruct NevProp3 to read in that test data at the end of model fitting, and generate a test prediction file. One way to do demonstrate this is to randomly cut, say, 9 cases from the Iris dataset, and paste them into a new file. Call this file `Iris.tst` (the .tst stands for *test* data). Enter the following settings into your `Iris.net` file:

```
ReadTestFile  Iris.tst
SaveTestPredFile  Iris.pts
```

If you wish to have upper and lower 95% confidence limits for each point estimate, specify, say, 50 bootstrapped models, by including the setting:

```
NBoots  50
```

Rerun the earlier model (except that the training data is now comprised of 141 cases). At the conclusion of the run, NevProp3 summarizes the predictive performance on the 9 cases extracted to the test set.

NevProp3 will display the feedback that `Iris.pts` was saved. Here is the `Iris.pts` file:

SEQU	PRED#1	LO95CI#1	UP95CI#1	TRUE#1
2	0.988690	0.631435	0.999776	1
25	0.939258	0.664450	0.991787	1
37	0.986750	0.618096	0.999708	1
71	0.009641	0.000212	0.309347	0
81	0.032757	0.005152	0.181318	0
97	0.031662	0.000710	0.600713	0
106	0.000236	0.000004	0.012977	0
130	0.000236	0.000005	0.011916	0
146	0.000236	0.000005	0.010735	0

Compare these predictive intervals to those generated earlier, when the same cases where part of the training dataset (excerpted from `Iris.ptr` in the previous step):

2	0.970291	0.889870	0.992482	1
25	0.907146	0.549637	0.987375	1
37	0.968805	0.875376	0.992770	1
71	0.051601	0.004778	0.381438	0
81	0.167461	0.005100	0.887542	0
97	0.150386	0.011107	0.736120	0
106	0.000155	0.000147	0.000163	0
130	0.000155	0.000147	0.000163	0
146	0.000155	0.000147	0.000163	0

Notice that while point estimates do not differ much, confidence intervals are substantially broader (i.e., less certain) when the cases are held out for testing, as expected.

STEP 6. INTERRUPT NevProp3 TO MODIFY SETTINGS ON-THE-FLY.

The Iris dataset is quite small, so that NevProp3 computation procedes quickly on modern platforms. In real-world problems, however, computation could take many minutes to hours. You may therefore wish to change settings on-the-fly, rather than waiting until a model is completely fit.

Section 5.1.3 provides an example of an interrupt. On UNIX systems, the interrupt is simultaneous use of `Control` and `c` keys; on DOS systems, the F1 function key; and on the Macintosh, the key combination of `command(alt)` and `period(.)`.

To try out the interrupt function on `Iris.net`, rerun the model above, but hit the interrupt as soon as the first progress report line (Epoch 0) appears.

Respond to the prompt to inspect the current setting values, then change one of them (e.g., increase the magnitude of the `WeightDecay` by a factor of 100). Return to processing, and inspect the final summary statistics to see if the changes substantively affected the results.

STEP 7. IMPUTING MISSING DATA.

Try deleting a few of the predictor variable entries in the Iris.net training data (replace the value with either a period, or the characters "na"). Because Iris.net contains the setting,

```
ImputeMissing median
```

missing values on predictors will be replaced by the median of the complete values for that variable. The descriptive statistics reflect the change; for example, after we deleted 5 values for the first predictor (compare this to the display shown in Step 4):

```
##### Descriptive Statistics on Input (Predictor) Variables
#####
-----

```

variable ^^^^^^	n ^^^^^^		missing ^^^^^^		unique ^^^^	mean ^^^^	
1. SepLen	150		5		35	5.8227	
percentile:	0.05	0.10	0.25	0.50	0.75	0.90	0.95
value:	4.6000	4.8000	5.1000	5.7000	6.4000	6.9000	7.3000
lowest:	4.3000	4.4000	4.4000	4.4000	4.5000		
highest:	7.7000	7.7000	7.7000	7.7000	7.9000		

```

"SepLen " mean of cases not missing "SepLen " = 5.8269
"SepLen " mean of cases with imputed "SepLen " = 5.7000

"Species" mean of cases not missing "SepLen " = 0.3103
"Species" mean of cases with imputed "SepLen " = 1.0000
-----

```

Notice the appearance of new lines subset information. Because the mean in the line starting with the variable name includes the imputations, the variables mean values are broken out, below, for those missing and not missing (the median is 5.7, which is imputed for all missing elements).

Also shown is the mean value of the dependent variable, "Species", for case with and without missing values. We deleted values only for Species 1.

Alternatively, you could have NevProp3 impute means, impute a random draw, or just delete cases with any missing values. A more complex procedure (kNN-ANN) uses Expectation-Maximization theory to impute the values based on the joint distribution of the variables. The method selected should be determined by the type of data and hypotheses (see section 5.5.3, 7.3.6). Optionally, you can instruct NevProp3 to save a new datafile with the missing values filled in by the imputations (see sections 7.2.4 & 7.2.5).

CHAPTER &5. GENERAL OPERATION

&5.1 Interface Options

5.1.1 Fully-Interactive Mode

NevProp1 operated only in this mode. By executing "np," a series of prompts was provided to ascertain names of the .net file, the .res file, any previous .wts file to upload, and important settings of operation.

Although somewhat tedious for repetitive operation, the fully-interactive interface is the easiest approach for novices, and is still supported. Upon initiation, the user is prompted for a random seed, name of the network (.net) and (optional) results (.res) file names, maximum number of epochs, and reporting interval. After each optimization set, the user is prompted about quitting versus continuing with additional epochs of optimization.* In AutoTrain mode, the user is prompted between the two phases.

Optimization can be interrupted to change certain settings— see 5.1.3.

At the completion of the run, the user is prompted for the (optional) names of weights (.wts) and prediction (.ptr and .pts) files.

Note: The user is not prompted for .res, .wts, .ptr, and .pts file names if these names were specified in the .net file.

* This is the only feature not available in command-line mode.

5.1.2 Command-Line Argument Mode

A feature added with version 2 was the command-line specification of basic operating settings. The user must minimally provide 3 arguments following the "np," and an optional fourth argument. If only 1 or 2 arguments are specified, the following error message is reported:

```
!!!! Either enter just "np" for INTERACTIVE MODE,  
or a minimum of 3 arguments following "np" for COMMAND-LINE MODE:
```

```
Usage: np [ NetFileName MaxIterations ReportFreq [Randomseed] ]
```

As indicated in the Usage statement, the arguments must be (1) the .net file name (appending the ".net" extension is optional), (2) the maximum number of epochs allowed for this run, (3) the frequency of reporting training progress (and hold-out statistics, if using stopped-training), and (4) an optional seed for the pseudo-random number generator (otherwise, clock time is used). Pseudo-random numbers are used to initialize the weights to break the symmetry if starting a new model, to shuffle training data before starting (if requested by the .net file setting ShuffleData), and to repeatedly rerandomize the data in stochastic updating mode (.net file setting Stochastic).

Interrupting training to modify setting *is* allowed after command-line startup— see 5.1.3.

REDIRECTING STDOUT & STDERR. Under UNIX, standard output and standard error may be redirected to files rather than the terminal display. Standard output receives statistical reporting during the course of training, the graphic display at the end of training (when stopped training is used), and statistics at the end of bootstrapping (when `NBoots > 0`). Standard error receives warnings, feedback to the user on the names of files read or saved, and interactive prompts.

5.1.3 Interrupting Training

UNIX. Once training has commenced (from either fully-interactive or command-line startup), `^c` (simultaneous `Control` and `c` keys) can be sent to interrupt the training before the start of the next epoch of training. Although the training is interrupted immediately, it may take a few seconds before the display buffer flushes to the screen. A double-bell is issued when the following prompt appears:

```
!!!! ^c received -- interrupting the program...
...Simulation interrupted at Epoch 20
```

Enter exact <setting name> to inspect or change its current value.

Need a reminder?

Enter 's' to inspect the current values of dynamic settings.

Then, 'c' to continue simulation
-or- 'q' to quit NevProp.

```
>>[<name>scq]:[c]
```

At this prompt, the user may enter the exact name (it's not case-sensitive) of the setting to be changed, and NevProp will echo the setting name and indicate its current value. For example, to inspect the current value of the `Momentum` (which turns out to be `.01`), and increase it to `.05`:

```
>>[<name>scq]:[c] momentum
momentum[0.01]: .05
>>[<name>scq]:[c]
```

To see a listing of alterable settings, enter 's'; to continue with training enter `c` or just a carriage return; and to quit NevProp abruptly (*without* saving any `.wts`, `.ptr`, or `.pte` files) enter `q`.

If you choose to quit, you will be offered an additional option:

```
>>[<name>scq]:[c] q
Exiting...
Save requested .wts &/or prediction files before quitting[ny]?[n]
```

Responding “y” is a convenient way to save prediction and/or weights files as of the current number of iterations, without completing the originally-intended number of training epochs.

DOS. The function key F1 is used to interrupt. Otherwise same as UNIX description, above.

MACINTOSH. The key combination of command(alt) plus period(.) is used to interrupt. Otherwise same as UNIX description, above.

&5.2 Formatting the Network (.net) File

NevProp3 requires an ASCII file that contains information needed to configure, train and test a neural network model. This is called the network file, and *must* end with the extension **.net**.

If NevProp3 finds a character, word, or number that doesn't match the format of the **.net** file, it will complain with a specific warning or error (see Chapter 8), then give a location statement that looks something like this:

```
!!!! Please check line 4 in input file.
```

There are 5 types of information that can be provided in the **.net** file:

- (1) User-specified comment lines can be place at any location in the **.net** file, including any appended data (and within **.trn** and **.tst** data files as well). Comment lines must follow one of the following 2 formats (both formats can be used in the same **.net** file):

, a single pound sign, anywhere in the line. All subsequent text *between the pound sign and the end of that line* is ignored by NevProp3.

/* comment-text-here */, that is, text (which may encompass multiple lines) sandwiched between paired slash-asterick and asterick-slash. Paired comments embedded within paired comments are not allowed.

For example, the following are 2 equivalent ways, in a **.net** file, of ignoring the setting and argument of **NEffectBoots**, forcing it to take it's default value:

NBoots	5	# NEffectBoots 500
--------	---	--------------------

NBoots	5	/* NEffectBoots 500 */
--------	---	------------------------

To ignore settings or data across multiple lines, here are 2 alternative commentings, where the middle 2 lines of data are to be ignored by NevProp3:

5	6.7	3	5.2	2.3	0.4847	1	0	0
#6	6.7	3.3	5.7	2.5	-1.3032	1	0	0
#7	6.8	3.2	5.9	2.3	-1.1097	1	0	0
8	5.8	2.7	5.1	1.9	-1.2641	1	0	0

5	6.7	3	5.2	2.3	0.4847	1	0	0
/*								
6	6.7	3.3	5.7	2.5	-1.3032	1	0	0
7	6.8	3.2	5.9	2.3	-1.1097	1	0	0
*/								
8	5.8	2.7	5.1	1.9	-1.2641	1	0	0

(2) Single-argument Settings are words followed (after 1 or more spaces or tabs) by a single numeric or textual argument. Settings and their argument must be on the same line. They may be inserted anywhere within the **.net** file, entered as any combination of upper and lower case text. We tried to select words that were meaningful, and whose first few letters were unique (to facilitate a quick search through the file). See subsequent sections for specific settings.

(3) Multi-argument Settings are words followed on a *single* line by a multiple numeric or textual argument (each separated by 1 or more spaces or tabs). Settings and all respective arguments must be on the same line. They may be inserted anywhere within the **.net** file, entered as any combination of upper and lower case text. See subsequent sections for specific settings.

(4) Keywords. At present, the keywords are:

FileFormat 3.x, placed at the top of the file. While not necessary for the operation of NevProp3, it checks for consistency in file formats, and is needed for future update patches.

DATA, placed *on its own line*, below the last setting in the file, tells NevProp3 that that training data (possibly preceded by headers— section 7.3.3) is appended below.

(5) Appended Training Data. If the keyword **DATA** appears (see above), NevProp3 will seek training data (possibly preceded by headers— sections 5.3 and 7.3.3) on subsequent lines in the **.net** file. In this circumstance, NevProp3 will ignore data contained in an external file specified by the **ReadTrainFile** setting (see 5.3 and 7.3.1).

&5.3 Formatting & Reading Data

5.3.1 Data appended within the .net File

Small datasets are conveniently kept within the network file by appending the lines of data following the keyword **DATA**. For example, here is part of figure 4.1.1:

```
# This is my first attempt at a NevProp3 network file.
# The following is a linear model.

Ninputs 2          Nhidden 0          Noutputs 1
Connect 1 2      3 3 /* connects inputs to output unit */
```



```

AutoTrain NO          ScoreThreshold 0.1
NHeaders 1            IDColumn YES

DATA

ID      chol    age    days
A        100     0      10
B        200     0      20
C        300     0      30
D        400     0      40
[remaining lines omitted--see figure 4.1.1]

```

Notice that the keyword **DATA** appears on it's own line after all settings are specified. Headers, if any, and data lines follow. Any number of blank lines may separate header and data lines. Any amount of blank spaces or tabs may separate entries on a line. Individual records may extend across any number of physical lines (that is, across lines terminated by carriage returns).

Header(s) used to provide column labels for the variables should be placed just before the data lines begin. The reason is that NevProp3 starts with the end of the last header and works backwards and upwards until it reads in the number of label tokens corresponding to the total number of variables. In the example above, NevProp3 expects $N_{inputs} + N_{outputs} + 1$ (the IDColumn) = 4 labels, so it starts with the last header (here, the only header), reading in "days," "age," "chol," and then "ID" to serve as variable labels.

Comments (see 5.2) may be intermingled with data. For example, the following is interpreted by NevProp3 identically as the file above:

```

DATA

ID      chol    age    days  #(This the header row.)
A        100 0  10    #Spaces used to separate entries.
/* here is a comment I inserted to remind
me that I changed something */
B        200     0      20 #Tabs for separation, here.
C        300 /*Single record across 2 lines, here.*/
              0      30
D
400
0
40    # Single record across 4 lines, here.

```

5.3.2 External data files (.trn & .tst)

Large datasets are best saved separately from the .net file. Data to be used to *train* a model must be saved in a file with the extension .trn. Data to be used to *test* a model must be saved in a file with the extension .tst.

To access an external **.trn** dataset, the network file must *not* contain the keyword **DATA**. Instead, somewhere in the file should be the setting-filename pair "ReadTrainFile *anyfilename.trn*", where *anyfilename* is the (arbitrarily chosen) main part of the filename. You can confirm that the proper file was read by observing the display at the beginning of the NevProp3 run.

Likewise, to specify a test dataset to be passed through an existing model, include in the network file the setting-filename pair "ReadTestFile *anyfilename.tst*".

As a convenience, only the main portion, *filename*, need be specified in the .net file—NevProp3 assumes the extensions **.trn** and **.tst**. For example, the following 2 specifications are equivalent:

ReadTrainFile	MyData.trn
ReadTestFile	MyData.tst

ReadTrainFile	MyData
ReadTestFile	MyData

Here is a sample **.trn** or **.tst** file (the corresponding .net file would need to specify, "NHeaders 1" and "IDColumn YES"):

ID	chol	age	days
A	100	0	10
B	200	0	20
C	300	0	30
D	400	0	40

Header(s) used to provide column labels for the variables should be placed just before the data lines begin. The reason is that NevProp3 starts with the end of the last header and works backwards and upwards until it reads in the number of label tokens corresponding to the total number of variables. In the example above, NevProp3 expects $N_{\text{inputs}} + N_{\text{outputs}} + 1$ (the IDColumn) = 4 labels, so it starts with the last header (here, the only header), reading in "days," "age," "chol," and then "ID" to serve as variable labels.

Statistics resulting from the processing of train and test file data will be displayed to standard output (and to the results file, if a ResultsFile was specified in the .net file).

If training data was provided (either after a **DATA** keyword, or in an external **.trn** file), test data will be passed through the model at the end of training. Alternatively, test data may be passed through an *existing* model, saved previously as a **.wts** file. In this case, the user may specify zero maximum epochs of training, which instructs NevProp3 to use existing weights and any test file specified.

For example, say you have already fit several models. Based on bootstrapped bias correction or regularization, you select the neural network model you believe best represents the true regression function; call the corresponding weights file "Best.wts." Here are network file specifications that will use existing weights to generate and save summary statistics and predictions:

```
/* excerpt from MyModel.net—other settings not shown */
ReadWeightsFile      Best.wts
ReadTestFile          MyData.tst
SaveTestPrdFile       MyData.pts
```

Using the command-line mode, you could call NevProp3 to perform the prediction only, with no further training, as follows at the system prompt:

```
np MyModel.net 0 0
```

It is important to realize that if the *same* test file is run on multiple models, and final model selection is based on test data performance, then the test data was actually used part of the model *fitting*. In this case, the test file could *not* be considered as independent validation data.

5.3.3 Handling Missing Data in NevProp3

The NevProp3 settings “ImputeMissing” (section 7.3.6) and “kNN” (sections 6.5 and 7.4.8) offer ways to deal with missing data elements in predictor variables. In section 2.7, basic terminology and theoretical concepts were presented. It is important that management of missing data be appropriate for the mechanism of missingness. Below, NevProp3 options are listed according to approaches discussed in section 2.9.

- COMPLETE-CASE (CC) ANALYSIS: Cases with missing are deleted.

NevProp3 option: “ImputeMissing delete” (the default)

- AVAILABLE-CASE (AC) ANALYSIS: On subsets of variables, the largest set of available cases are used depending on the parameters being estimated.

NevProp3 option: Subset the predictor variables using the InputColumns setting, and set “ImputeMissing delete.”

- IMPUTATION

Unconditional mean imputation. Missing values are assigned the unconditional observed-sample mean or median.

NevProp3 option: “ImputeMissing mean” or “ImputeMissing median”

Conditional mean imputation. Missing values are predicted from the observed values of other variables, assuming MAR.

NevProp3 option: Each predictor variable is sequentially modeled as the dependent variable, using the remaining predictors as independent input variables. This can be performed using only the complete cases (i.e., setting “ImputeMissing delete.”) Alternatively, the complete cases are used only for the first variable’s model: as each predictor-as-dependent variable model is fit, its missing values are imputed from the model. This allows additional cases to appear complete, and to be used in the fitting of the next predictor-as-independent variable model. The

imputation process is repeated between subsequent models, until all missing elements are imputed. This process would be very labor-intensive.

Multiple imputation (MI).

NevProp3 option: “ImputeMissing random” causes a random draw to be made from the observed distribution, to impute missing elements on each variable. Running the identical architecture (using different random seeds at startup) generates multiple models differing in the imputed values. Subsequent analyses should include means and variance, across the models, in summary statistics and predictions.

Expectation-Maximization (EM).

NevProp3 option: Specifying a value for `kNN` (sections 6.5 and 7.4.8) causes NevProp3 to generate that many nearest neighbors for each case (vector of predictors). The network file is set up so that the number of outputs is the sum of predictor and dependent variables (if any). Missing elements are initially imputed according to `ImputeMissing` (mean, median, or random), but subsequently updated iteratively as inputs are used to predict their nearest neighbors and dependent variables (if any) as outputs. Summary statistics and predictions on the dependent variables are viewed separately by using the `OutputStatVars` setting. The iteratively-updated values imputed for missing elements are not actually seen by the user (although final imputed datasets may be saved using the settings `SaveTrainImputFile` or `SaveTestImputFile`).

- BAYESIAN METHODS:

NevProp3 option: The multilayer perceptron is a direct prediction/probability model. However, the `kNN` mode (sections 6.5 and 7.4.8) can be used to estimate the joint distribution of predictors. Also, `ARD` (sections 6.3 and 6.4) can be used during model fitting to adaptively enforce prior distribution assumptions on parameters or groups of parameters.

&5.4 Saving NevProp3 Results (.res), Predictions (.ptr, .pts), Weights (.wts) and Imputation (.itr, .its) Files

5.4.1 Results (.res) File

If “ResultsFile *anyfilename.res*” is specified in the `.net` file, NevProp3 will save to the current directory an ASCII file by that name, containing the standard output of the run. (As a convenience, only the main portion, *anyfilename*, need be specified in the `.net` file— NevProp3 assumes the extension `.res`.)

A time & date stamp is placed at the start and end of each results file. For example:

```
##### NevProp (NP3.0) started on Mon Feb 12 22:44:09 1996

[----- results of run displayed here -----]

*****
NevProp completed on Mon Feb 12 22:44:13 1996
```

UNIX users may, alternatively, redirect standard output from the screen to an output file, using a command format like:

```
np Iris.net 500 25 999 > Iris.out
```

Note, however, that date & time stamps are not saved using the latter approach.

5.4.2 Predictions (.ptr, .pts) Files

If specified in the **.net** file, NevProp3 will save to the current directory an ASCII file containing predicted and true target values of dependent (output) variables. Following the fitting of a model, the user can save the predictions of the final model on the training data itself by specifying the setting `SaveTrainPredFile` followed by the name of a file (see 7.2.2 for format). Likewise, predictions on a test (validation) dataset may be obtained by specifying the setting `SaveTestPredFile` followed by the name of a file (see 7.2.3 for format).

The prediction files always begin with a header row that identifies the columns of the rows that follow. Tabs are used to separate entries, to facilitate uploading of the prediction files into spreadsheet programs. Here is the general header row format:

SEQU	PRED#1	TRUE#1	PRED#2	TRUE#2	etc...
------	--------	--------	--------	--------	--------

Explanation:

SEQU: If the original data files contained an `IDColumn`, **SEQU** will reproduce those assignments. Otherwise, **SEQU** represents the numerical sequence in which each line appeared in the train or test datafile.

PRED#i: The model's prediction of the *i*th dependent variable.

TRUE#i: The target value of the *i*th dependent variable, as supplied in the data file.

If, in addition, bootstrapped models were generated (by specifying `NBoots > 0` in the **.net** file), NevProp3 will use the bootstrapped standard deviation of predicted values on the training data to estimate upper and lower 95% confidence limits. For dichotomous dependent variables, standard deviations are computed on the log-odds scale, then the inverse logit transformation is used to provide confidence limits on the probability scale. (Note: in the current version of NevProp3, the weights sets corresponding to booted models are not saved after the original model is fit. Therefore, confidence limits cannot be computed for test data.)

Here are examples of training data prediction files resulting from two runs of **Iris.net** (where `OutputColumns` was used to select a single dependent variable). Because "ShuffleData YES" was specified in the **.net** file, the sequence of records in the prediction file reflects the initial pre-randomization— however, the corresponding original **SEQU** identifiers from the `IDColumn` are preserved.

In the first run, `NBoots` was set to 0:

SEQU		
8	0.995577	1
53	0.000319	0
39	0.997251	1
5	0.999915	1
81	0.387698	0

90	0.001512	0
79	0.541439	0

In the second run, NBoots was set to 100, causing NevProp3 to estimate 95% confidence intervals for each prediction:

SEQU				
8	0.995577	0.964808	0.999459	1
53	0.000319	0.000275	0.000370	0
39	0.997251	0.990700	0.999191	1
5	0.999915	0.999904	0.999924	1
81	0.387698	0.010544	0.974109	0
90	0.001512	0.000548	0.004168	0
79	0.541439	0.021850	0.984230	0

Notice that the variance in the predictions of the cases was very small. An exception is case number 79, which is not linearly separable, and requires a highly nonlinear model to classify according to the target (which raises the question about correctness of classification by the person who supplied the flower data to Fisher).

5.4.3 Weights (.wts) File

If "SaveWeightsFile *anyfilename.wts*" is specified in the **.net** file, NevProp3 will save to the current directory an ASCII file by that name, containing the weights and several additional pieces of information. If the **.wts** extension is not explicitly supplied, it will be automatically appended upon saving to disk (that is, specifying "SaveWeightsFile Iris" is the same as specifying SaveWeightsFile Iris.wts").

Here is a **.wts** file saved from a model of the Iris data, using a single dependent variable (unit 8):

EpochSaved	75		
5	0	-2.24208	
5	1	-0.305499	
5	2	-0.35299	
5	3	0.589514	
5	4	1.40845	
6	0	-1.1344	
6	1	0.0999478	
6	2	-0.398554	
6	3	0.217984	
6	4	0.648066	
7	0	3.10812	
7	1	-0.00368835	
7	2	0.597523	
7	3	-0.531628	
7	4	-1.34295	
8	0	-1.05513	
8	5	21.1305	
8	6	2.04793	
8	7	-20.6307	
TrainingPrior for output 1 is		0.333333	
UseARD NO			

```
##### Summary of Input Relevance Determination
#####
Relevance formula: R_i = ( sum square weights of ith input group)
                      ( sum square weights of all input groups)

Input    Relevance
^^^^^    ^^^^^^^^^
1         1.49%
2         2.56%
3         44.47%
4         51.48%
##### End of Input Relevance Report
#####
```

Explanation:

1. EpochSaved: The epoch (iteration) of training at which the model was saved. This may be of interest to the user for later comparison of models. Also, if the .wts file is later read into NevProp3 to continue training, this epoch number is uploaded as the initial epoch.
2. Indexed weights: (one line per connection) The first two numbers are indexes of the unit *to which* it ends, *from which* the connection starts, respectively. The third number is the corresponding weight. Unit 0 is the constant, or bias, unit, which always transmits the value 1. In the example above, the lines,

5	0	-2.24208
5	1	-0.305499

indicate that the weight along the connection from unit 5 (the first hidden unit, because there are 4 inputs) to the bias unit is -2.24208, and that from unit 5 to the first input unit is -0.305499. Likewise, the lines,

8	0	-1.05513
8	5	21.1305

indicate that the weight along the connection from unit 8 (the single output unit, because there are 4 inputs + 3 hidden = 7 preceding units) to the bias unit is -1.05513, and that from unit 8 to the first hidden unit is 21.1305.

Note: For the sake of upload transportability for testing future data, by default the weights are reported on the scale of the original variables, regardless of any rescaling performed using the `StandardizeInputs` setting. However, to compare the magnitudes of the weights in a given model, force NevProp3 to save weights on the scale of the *transformed* variables by the setting "SaveStandWts YES." For example, the weights obtained from a run of `Iris.net` identical to the above, except for the change to "SaveStandWts YES," are as follows:

5	0	-1.20183
5	1	-0.252972
5	2	-0.153858
5	3	1.04067
5	4	1.07358
6	0	-0.172452
6	1	0.0827628
6	2	-0.173718
6	3	0.384806

6	4	0.493982
7	0	1.30489
7	1	-0.00305418
7	2	0.260443
7	3	-0.93848
7	4	-1.02365
8	0	-1.05513
8	5	21.1305
8	6	2.04793
8	7	-20.6307

Direct comparison of weight magnitudes may now be had. Notice, for instance, that the strength of the connections from the output unit (index 8) from two of the hidden units are large but opposite in sign (indexes 5 and 7; weights 21.1305 and -20.6307), but relatively small from the other hidden unit (indexes 6; weight 2.04793). One possibility is that if the connections from those two hidden unit to the inputs are redundant, the effects could be cancelled at the output unit, and hidden unit index 6 could, in fact, be relevant. However, those hidden-to-input connections are largely opposite in polarity: (index 5 to indexes 1-4: -0.252972, -0.153858, 1.04067, 1.07358) vs (index 7 to indexes 1-4: -0.0030541, 0.260443, -0.93848, -1.02365). Thus, the conclusions are: (1) unit 6 is largely ignored, and at most 2 hidden units would probably have sufficed; (2) the strongest effects arise from input units 3 and 4. A better way to summarize the effects is given by the relevance determination summary, below.

3. **TrainingPrior**: The mean value of each dependent variable. Although this may be of general interest to the user (it can also be obtained using the `DescribeVars` setting), it is mainly provided here so that, when uploading the `.wts` file to generate summary statistics on a test dataset, the R^2 and Nagelkerke R^2 are calibrated to the priors of the training dataset. This is necessary because a small difference in priors produces a large offset in the R^2 calculations, upsetting the the 0-to-1 scale of the R^2 .
4. **UseARD**: Indicates the ARD setting used to fit the model, as a reminder to the user.
5. **Relevance Determination summary**: Reproduction of the information reported to standard output at the end of the run. This is provided for the convenience of the user in later comparison of models. Please refer to sections 5.6.1 and 6.4 for more detailed descriptions.

5.4.4 Imputation (.itr, .its) Files

If the data file (`.trn` and/or `.tst`) contained missing elements (indicated by the `.` or NA placeholders), NevProp3 can save ASCII files that contain a complete, imputed data file. To request this, the `.net` file should specify “`SaveImpTrainFile filename.itr`” and/or “`SaveImpTestFile filename.its.`” In the general operating mode of NevProp3, imputation is performed according to the `ImputeMissing` setting (see 5.5.3 and 7.3.6). In kNN mode, imputation is performed according to an Expectation-Minimization algorithm (see 5.5.3 and 7.4.8).

The purpose of the imputation option is two-fold: first, to permit the user to inspect the actual values imputed for model fitting, and, second, to generate complete data files for use in other applications.

For example, consider the predictor-variable portion of the first six records of the Iris training data (no missing elements initially):

1	5.9	3	5.1	1.8	0.5799
2	6.2	3.4	5.4	2.3	0.5863
3	6.5	3	5.2	2	0.39
4	6.3	2.5	5	1.9	-0.2844
5	6.7	3	5.2	2.3	0.4847
6	6.7	3.3	5.7	2.5	-1.3032

Now, let's arbitrarily lesion the data, removing one element from each of the five predictor variables. 'NA' is used to hold the places of the missing elements. The sixth record is left complete, to demonstrate that it is not affected by the subsequent imputation:

1	NA	3	5.1	1.8	0.5799
2	6.2	NA	5.4	2.3	0.5863
3	6.5	3	NA	2	0.39
4	6.3	2.5	5	NA	-0.2844
5	6.7	3	5.2	2.3	NA
6	6.7	3.3	5.7	2.5	-1.3032

Let's name the file to receive the imputed dataset, "Iris.itr." In the Iris.net file, specify "SaveImpTrainFile Iris.itr," and "Imputemissing mean." Run NevProp3 (at least 1 epoch), and inspect the file "Iris.itr." Notice that the missing elements have been imputed with the *mean* of each corresponding variable (as can be confirmed by setting "DescribeVars YES"). Below, we add the underlining for clarity:

1	<u>5.84295</u>	3	5.1	1.8	0.57990
2	6.2	<u>3.05503</u>	5.4	2.3	0.58630
3	6.5	3	<u>3.74832</u>	2	0.39000
4	6.3	2.5	5	<u>1.19463</u>	-0.28440
5	6.7	3	5.2	<u>2.3</u>	<u>-0.06935</u>
6	6.7	3.3	5.7	2.5	-1.30320

If, instead, we had specified "Imputemissing median," the missing elements would have been imputed with the *median* of each corresponding variable (as can be confirmed by setting "DescribeVars YES"):

1	<u>5.8</u>	3	5.1	1.8	0.57990
2	6.2	<u>3</u>	5.4	2.3	0.58630
3	6.5	3	<u>4.3</u>	2	0.39000
4	6.3	2.5	5	<u>1.3</u>	-0.28440
5	6.7	3	5.2	<u>2.3</u>	<u>-0.16190</u>
6	6.7	3.3	5.7	2.5	-1.30320

If, instead, we had specified "Imputemissing random," the missing elements would have been imputed randomly (with replacement) from each corresponding variable:

1	<u>6.4</u>	3	5.1	1.8	0.57990
2	6.2	<u>3.4</u>	5.4	2.3	0.58630
3	6.5	<u>3</u>	<u>1.5</u>	2	0.39000
4	6.3	2.5	5	<u>2.3</u>	-0.28440
5	6.7	3	5.2	<u>2.3</u>	<u>-0.28380</u>
6	6.7	3.3	5.7	2.5	-1.30320

Using the imputation methods, above, in the general operating mode of NevProp3, the substituted values in one predictor in no way depend on the values of other predictors. However, in kNN mode, an EM algorithm is used to estimate the *joint* distribution of the predictors, which, in turn, is used to impute missing elements. For example, here we ran `Iris.net` for 6000 epochs (about where errors and weights became asymptotic), with the following non-default settings: “SaveTrainImputFile `Iris.itr`,” “ImputeMissing median,” “kNN 1,” “Ninputs 5 Nhidden 30 Noutputs 6,” “AutoTrain NO,” and “PercentHoldout 0”:

1	5.79244	3	5.1	1.8	0.57990
2	6.2	2.81386	5.4	2.3	0.58630
3	6.5	3	4.33519	2	0.39000
4	6.3	2.5	5	1.98497	-0.28440
5	6.7	3	5.2	2.3	-0.13279
6	6.7	3.3	5.7	2.5	-1.30320

Compared to the earlier non-kNN imputation methods, the kNN-imputed values more consistently approximate the original values (except for the last variable, which is Gaussian noise without correlation to other variables).

&5.5 General Approach to Network Design & Regularization

The nonlinearity required to classify the last 2 of 150 cases suggests that these are aberrant cases, and should probably be deleted, and a simpler model fit. However, in more complex real-world models, it will not be so clear when deleting cases is appropriate or detrimental to creating a model that generalizes well. Regularization techniques are used to prevent excessive nonlinearity (i.e., to prevent growth of weights leading to nonlinear activation functions). NevProp3 provides three general methods: a constant proportional weight penalty (`WeightDecay`), a Bayesian-inspired dynamic weight decay (`UseARD`), and early stopping based on preliminary cross-validation (`AutoTrain`).

&5.6 Interpreting the Results of NevProp Procedures

5.6.1 Overview of results display

The results displayed to standard output (and, optionally, saved to a `.res` file) is a sequence of 8 potential types of information, listed then described, below:

- 5.6.1.1. “Regurgitation” of headers and settings
- 5.6.1.2. Descriptive statistics on the variables
- 5.6.1.3. Progress report during model fitting
- 5.6.1.4. Graphical summary of cross-validation
- 5.6.1.5. Summary of predictor effects
- 5.6.1.6. File saving activity report

- 5.6.1.7. Summary statistics of predictive accuracy
- 5.6.1.8. If bootstrapped models are specified (i.e, NBoots > 0), 1-5 above will first be displayed for the full data model, followed by 3-5 for each bootstrapped model, followed by a bootstrap bias graphic and bias-adjusted predictive accuracy statistics.
- 5.6.1.9. If mean predictor effect bootstraps are specified (i.e, NEffectBoots > 0), a summary table is displayed.

5.6.1.1. **REGURGITATION.** The “regurgitation” is intended to be self-explanatory. First, any headers in the training datafile are reproduced. This allows descriptive headers to appear in the output, useful for later reference to variable labels. For example:

```
##### Starting NevProp NP3.0
#####

HEADER #1: This is the first header. You can use it to describe the
data.

HEADER #2: Flower SepLen SepWid PetLen PetWid Species1 Species2
Species3
```

Next comes confirmation that settings were successfully read, the random seed (if training de novo), and the number of training cases read (and holdout subset size, if specified):

```
... SETTINGS were successfully read from "Iris.net".
... SEED=1234 using lrand48(),srand48()
... Read in 150 TRAIN CASES [random 75 cases (50%) to a holdout
subset]
```

Next, the regurgitated settings appear, formatted to allow a copy-and-paste to serve as the backbone of a new .net file:

```
# DATA FILE SETTINGS
NHeaders 2 IDColumn YES
StandardizeInputs 1 SaveStandWts NO ImputeMissing median
InputColumns 1 2 3 4
OutputColumns 6
NVars 8 ShuffleData YES
# REPORTING SETTINGS
DescribeVars NO
NBoots 100 NEffectBoots 0
CalccIndex YES ScoreThreshold 0.5
OutputStatVars 0
# CONNECT CALLS
Connect 1 4 5 7
Connect 5 7 8 8
# CONFIGURATION SETTINGS
Ninputs 4 Nhidden 3 Noutputs 1
kNN 0 lofN NO
HiddenUnitType 1 OutputUnitType 3
/* OutputUnitTypes assigned: 2 */
WeightRange 0.001
# TRAINING SETTINGS
TrainCriterion 2
BiasPenalty NO WeightDecay -0.001
OptimizeMethod 3 SigmoidPrimeOffset 0
```

```

QPMaxFactor 1.75      QPModeSwitchThreshold 0
Stochastic NO         LearnRate 0.001      SplitLearnRate NO    Momentum
0
# BEST-BY-HOLDOUT SETTINGS
  PercentHoldout 50.00
  AutoTrain YES      MinEpochs 50          BeyondBestEpoch 1.5
  NSplits 3          SepBootXVal NO
# AUTOMATIC RELEVANCE DETERMINATION SETTINGS
  UseARD NO          WhenARD Auto          ARDTolerance 0.1  ARDFreq 25
  GroupSelection Input BiasRelevance NO    ARDFactor 1

```

5.6.1.2. DESCRIPTIVE STATISTICS. It is important to ascertain the integrity and quality of data used for model fitting. It is therefore recommended that, at least the first time a given dataset is accessed, "DescribeVars YES" be specified in the .net file. This descriptive information may also be useful later for interpretation of the model effects.

NevProp3 first scans each variable to determine whether it is either multilevel (2 to 10 discrete values), or continuous (defined as > 10 values). The first display line shows the variable name (if provided as a header; otherwise by sequence), total number of cases, number of missing elements (allowed only in predictor variables; signified in the dataset by the . or NA placeholder), the number of unique values found, and the arithmetic mean. Subsequent lines show, for multilevel variables, the level-value and percent of cases at each level; for continuous variables, a percentile distribution is shown. Note that even if StandardizeInputs is set to 1 (z-transformed) or 2 (-.5 to .5 rescaled), variables are described on their original scales. Here is the descriptive statistics segment from a run using Iris.net, where, for brevity here, the first 2 predictors were selected using the InputColumns setting, and a single output variable was selected using OutputColumns setting:

```

##### Descriptive Statistics on Output (Dependent) Variables
#####
-----
      variable      n      unique      mean
      ^^^^^^^      ^^^^^      ^^^^^
1.  Species1      150         2      0.3333
    0.00(   100, 66.7%)  1.00(   50 33.3%)
-----

##### Descriptive Statistics on Input (Predictor) Variables
#####
-----
      variable      n      missing      unique      mean
      ^^^^^^^      ^^^^^      ^^^^^      ^^^^^
1.  SepLen      150         0         35      5.8433
percentile:      0.05      0.10      0.25      0.50      0.75      0.90      0.95
value:          4.6000  4.8000  5.1000  5.8000  6.4000  6.9000  7.3000

lowest:          4.3000  4.4000  4.4000  4.4000  4.5000
highest:         7.7000  7.7000  7.7000  7.7000  7.9000
-----

      variable      n      missing      unique      mean
      ^^^^^^^      ^^^^^      ^^^^^      ^^^^^
2.  SepWid      150         0         23      3.0573
percentile:      0.05      0.10      0.25      0.50      0.75      0.90      0.95
value:          2.3000  2.5000  2.8000  3.0000  3.3000  3.7000  3.8000

lowest:          2.0000  2.2000  2.2000  2.2000  2.3000

```

```

highest:      3.9000  4.0000  4.1000  4.2000  4.4000
-----

```

```

##### End of Descriptive Statistics
#####

```

For each predictor variables with missing elements, two additional statistics are reported, to aid the user in determining the mechanism of missingness:

1. the means of the *predictor* variable for nonmissing cases of the predictor; and, if an imputation method was selected using the ImputeMissing setting, the mean of those imputed values; and,
2. the means of the *dependent* variable(s) for missing and nonmissing cases of the predictor.

For example, here is a descriptive summary, displayed when the training data had 5 of the 50 cases of the first predictor set to missing, with the setting “ImputeMissing median”:

```

-----
variable      n      missing  unique  mean
^^^^^^^^      ^^^^^      ^^^^^      ^^^^^
1. SepLen      150        5        35     5.7993
percentile:    0.05    0.10    0.25    0.50    0.75    0.90    0.95
value:        4.6000  4.8000  5.1000  5.7000  6.4000  6.9000  7.3000

lowest:       4.3000  4.4000  4.4000  4.4000  4.5000
highest:      7.7000  7.7000  7.7000  7.7000  7.9000

"SepLen " mean of cases not missing "SepLen " =  5.8269
"SepLen " mean of cases with imputed "SepLen " =  5.0000

"Species" mean of cases not missing "SepLen " =  0.3103
"Species" mean of cases with imputed "SepLen " =  1.0000
-----

```

Note that “n” remains 150, because the missing elements were imputed (using the setting “ImputeMissing delete,” “n” would have been reduced to 145). The number originally “missing” is correctly reported as 5. The value of the *predictor* variable itself, “SepLen,” for the nonmissing cases is 5.8269. The median value, 5.0, was imputed for all missing cases. The mean values of the *dependent* variable, “Species,” differs between missing and nonmissing cases, because only cases with dependent value 1 were intentionally chosen for deletion. Clearly, this indicates to the user that the data is not missing completely at random (see sections 2.9 and 5.3.3).

5.6.1.3. PROGRESS REPORT DURING MODEL FITTING. The optimization is preceded by a key to column abbreviations to follow:

```

-----
ABBREVIATIONS KEY:
LrnRat=LearnRate, FrGrdD=Fraction of weight updates using Grad
Descent,

```

```

Av=Average, Sq=Squared, Wt=Weights, A=(Average over cases &
outputs),
Pn=Penalized, CrEn=Cross-Entropy, Er=Error, Th=Thresholded, 1-c=(1 -
c index)
-----
-----

```

Not all terms are used for all types of optimization. For instance, FrGrdD is applicable only when using quickprop ("OptimizeMethod 3"), because it only occasionally defaults to first-order gradient descent. "Av" and "A" both signify arithmetic average, or mean. Average squared error (ASqEr) is the objective criterion appropriate for continuous dependent variables under the assumption of normally distributed residual error, whereas average cross entropy (ACrEn), or negative log likelihood, is more appropriate for dichotomous dependent variables. NevProp3 allows for penalized (Pn) optimization by specifying a WeightDecay (see 7.5.10) or by using ARD (see 6.3 and 7.7.1)– both penalized and unpenalized criterion values are displayed.

The average thresholded error (AThEr) is the proportion of case predictions falling outside the value ScoreThreshold*range; for binary variables, the range is one, so the AThEr is simply the proportion of case predictions not within ScoreThreshold of the target. The c index is a pure measure of discrimination (see 5.6.2 for details).

Next is the abbreviated column header. To facilitate easy numeric scanning of the report, accuracy measures have been presented as *errors*; that is, "lower is better." Also, measures are multiplied by 1000, to avoid excessive zeros after the decimal point. Seemingly complicated at first, the user will readily appreciate the format when familiar with the display. Here is an example, with Iris.net specifying a single dichotomous dependent variable (OutputColumns 6), gradient descent with a globally adaptive learning rate ("OptimizeMethod 1"), and no cross-validation ("PercentHoldout 0"), and "ScoreThreshold 0.5"; the command-line initiation specifies a total of 100 epochs of training, with reporting at an interval of 25 epochs:

```
np Iris.net 3000 25
```

The first report line displays statistics prior to any new training. If the network is being initialized with random weights, the statistics reflect these baseline predictions (Epoch 0). (Had training started from an uploaded weights file, the appropriate saved epoch number would be displayed, with statistics corresponding to the saved epoch.)

____TRAIN-SET_(150)____						
Train	LrnRat	AvSqWt	APnCrEn	ACrEn	AThEr	1-c
Epoch	(e-3)	(e-3)	(e-3)	(e-3)	(e-3)	(e-3)
-----	-----	-----	-----	-----	-----	-----
0	1.0000	0.0003	693.212	693.212	666.7	942.3
25	2.7644	0.0002	649.057	649.057	333.3	829.2
50	9.3611	0.0004	636.598	636.598	333.3	52.10
75	31.700	2.8913	633.392	633.392	333.3	19.20
100	107.35	3035.7	130.403	130.251	26.67	2.600
125	228.92	8276.7	93.2536	92.8398	33.33	1.400
150	142.23	9192.1	50.5748	50.1152	13.33	1.000
175	140.33	10321.	45.3584	44.8424	20.00	1.400
200	138.46	10886.	43.2126	42.6683	13.33	1.400
225	136.61	11445.	42.3739	41.8016	13.33	1.400
---- lines omitted for brevity ----						
2150	135.86	24090.	31.2420	30.0375	13.33	0.400
2175	114.90	24819.	29.7392	28.4983	6.667	0.400
2200	132.25	25455.	29.1200	27.8473	13.33	0.400

```

2225 111.85 26275. 27.9124 26.5986 6.667 0.400
2250 110.35 26950. 26.6759 25.3284 6.667 0.400
2275 127.02 27795. 25.9722 24.5824 6.667 0.400
2300 107.42 28495. 24.9307 23.5059 6.667 0.400
---- lines omitted for brevity ----
2525 111.04 35941. 18.3306 16.5336 6.667 0.000
2550 149.11 36713. 17.7625 15.9269 0.000 0.000
2575 126.10 37579. 17.1879 15.3089 6.667 0.000
2600 145.16 38368. 16.8148 14.8964 0.000 0.000
2625 122.76 39263. 16.3936 14.4305 6.667 0.000
2650 141.30 40009. 15.9591 13.9587 0.000 0.000
2675 139.42 40859. 15.3877 13.3448 0.000 0.000
2700 137.55 41635. 15.1958 13.1140 0.000 0.000
---- remaining lines omitted for brevity ----

```

As described in comments atop the `Iris.net` file, the Iris flower data are linearly separable with the exception of 3 cases. Optimization proceeds rapidly to correctly classify all but these 3 (`AThEr` of $3/150 = 20e-3$) by epoch 175; the average square weight needed was only about 10. The next case is then readily classified with only a minor increase in average weight size, leaving 2 (`AThEr` of $2/150 = 13.33e-3$) errors. However, these 2 remaining cases require the model to form highly nonlinear decision surfaces: the next case is correctly classified only after 2200 epochs (`AvSqWt` about 26), and the last case only after 2650 epochs (`AvSqWt` about 40).

The nonlinearity required to classify the last 2 of 150 cases suggests that these are aberrant cases, and should probably be deleted, and a simpler model fit. However, in more complex real-world models, it will not be so clear when deleting cases is appropriate or detrimental to creating a model that generalizes well. Regularization techniques are used to prevent excessive nonlinearity (i.e., to prevent growth of weights leading to nonlinear activation functions). A simple way to achieve regularization is to limit the number of hidden units. Additionally, `NevProp3` offers three algorithmic regularization methods (see also 5.5): a constant proportional weight penalty (`WeightDecay`; see 7.5.10), a Bayesian-inspired dynamic weight decay (`UseARD`; see 6.3 and 7.7.1), and early stopping based on preliminary cross-validation (`AutoTrain`; see 7.6.4).

During cross-validation phases, error statistics on both the train and holdout subsets are displayed; for example:

75)____		____TRAIN-SUBSET_(75)____				____HOLDOUT-SUBSET_(
Train	FrGrdD	AvSqWt	APnCrEn	ACrEn	AThEr	1-c	APnCrEn	ACrEn	
AThEr	1-c								
Epoch	(e-3)	(e-3)	(e-3)	(e-3)	(e-3)	(e-3)	(e-3)	(e-3)	(e-3)
0	N/A	0.0004	693.058	693.058	346.7	754.3	693.042	693.042	
320.0	752.0								
50	0.0000	21395.	72.2529	70.1134	40.00	3.140	79.5804	77.4409	
40.00	2.451								
100	0.0000	23379.	64.4067	62.0688	26.67	2.355	41.2546	38.9168	
13.33	1.634								
150	0.0000	20923.	63.2067	61.1144	26.67	3.140	39.0644	36.9721	
13.33	0.817								
200	0.0000	18960.	62.7451	60.8491	26.67	2.355	38.4300	36.5340	
13.33	1.634								
250	0.0000	17783.	62.2823	60.5040	26.67	2.355	37.9679	36.1896	
13.33	1.634								

300	0.0000	17668.	61.7833	60.0165	26.67	2.355	37.4636	35.6967
13.33	1.634							
350	0.0000	17606.	60.4220	58.6615	26.67	1.570	42.9954	41.2349
13.33	2.451							
400	0.0000	22178.	54.5798	52.3620	26.67	0.785	82.6483	80.4305
13.33	4.085							

Notice that while the train-subset errors progressively decrease, the errors on the holdout-subset first fall, then rise. Relative to the holdout, the model is overfit— see the next section for an easier way to inspect this pattern.

5.6.1.4. GRAPHICAL SUMMARY OF CROSS-VALIDATION. To avoid having to manually inspect the numeric display for cross-validation results, NevProp3 provides a graphic summary of the penalized error on the holdout-subset, for the 10 *best* reporting epochs. (These epochs need not be contiguous.) Corresponding to the previous cross-validation display, we have the graphic:

(Up to) 10 BEST reports on HOLDOUT-SUBSET using Best-By criterion graphed:					
Epochs	APCE (e-3)	BEST	<---- APnCrEn ---->	WORST	
100	41.254621	*****			
125	39.739981	*****			
150	39.064430	*****			
175	38.355321	*****			
200	38.429976	*****			
225	38.460441	*****			
250	37.967887	*****			
275	38.304742	*****			
300	37.463572	*			
325	41.005298	*****			

For split #3 of 5, target error = 0.061783 from Epoch 300.					

Visual inspection suggests this is a true minimum (the optimization proceeded for 500 epochs, all of which had higher holdout-subset errors). The last line indicates that this graphic represents the third of five cross-validations being used to find a good mean target error at which to stop, when later fitting the final model using all cases training.

5.6.1.5. SUMMARY OF PREDICTIVE EFFECTS. Presented here are relevance determination summary statistics from a run using *Iris.net*. The relevance formula (which depends on how *UseARD* is set) is reproduced each time; note that the computation is made on the standardized scale (if *StandardizeInputs* was set to 1 or 2), so that weight groups may be properly compared. An input “group” comprises all connections from a given input to its hidden units. When “skip” connections are specified (i.e., direct connections from input to output units), those weights are included in the input units’ group by default, or in the output units’ group if *GroupSelection* is set to *OUTPUT* (see 7.7.5) in the *.net* file.

The format of the display depends on the setting of *ARD* (see 6.4). First, with *ARD* turned off, the relevance display is limited to the normalized weights, as indicated by the formula:


```
##### Summary of Input Relevance Determination
#####
Relevance formula: R_i = ( sum square weights of ith input group)
                      ( sum square weights of all input groups)

Input   Relevance
^^^^^   ^^^^^^^^^
1       1.49%
2       2.56%
3       44.47%
4       51.48%
##### End of Input Relevance Report
#####
```

With ARD turned on (to any of its modes), a more informative display is provided; for example:

```
===== Summary of Automatic Relevance Determination
=====
ARD ( FULL ) was turned on at Epoch 10

No. of well-determined parameters in the model: 6.4, or 33%, of 19
total.

GROUP   FROM   TO      ARD HYPERPARAMETERS      # Well-determined
Parameters
^^^^^   ^^^^^   ^^      ^^^ ^^^^^^^^^^^^^^^^^^   ^ ^^^^^^^^^^^^^^^^^^
^^^^^^^^
0       bias   hidden      0.88                      1.1, or 36%, of 3
1       input1 hidden      155.71                    0.1, or 3%, of 3
2       input2 hidden      10.54                     0.9, or 29%, of 3
3       input3 hidden      1.09                      0.9, or 31%, of 3
4       input4 hidden      0.92                      1.0, or 32%, of 3
5       hidden output1    0.00                      2.4, or 60%, of 4
        bias

##### Summary of Input Relevance Determination Based on ARD
#####
ARD Relevance formula: R_i = (1.0/alpha_i)/(sum of all 1.0/alpha_i),
                          where alpha_i is the ARD hyperparameter for the ith input

Input   Relevance
^^^^^   ^^^^^^^^^
1       0.31%
2       4.51%
3       43.77%
4       51.42%
##### End of Input Relevance Report
#####
```

FULL ARD was turned on at Epoch 10 (see section 6.5). On average, only a third of the total available parameter power was used, suggesting that a simpler model with about 6 or 7 weights would have sufficed (in fact, this suggests a nearly linear model, since bias + 4 inputs = 5 weights). For each group (which now includes output group(s) encompassing connections from hidden units to each output), the final optimization penalty (ARD HYPERPARAMETER) is shown, and, from Hessian inversion, the corresponding effective ("well-determined") number of parameters used in each group.

The ARD-based relevance for each group, with a more complex formula, is then displayed. Note, however, that the values of the relevance are nearly identical to those obtained above with ARD off. The reason is that both runs used stopped-training (“AutoTrain YES”) regularization, so that, in this dataset, ARD could not improve upon the regularization. However, using ARD did permit the computation of the number of effective parameters, which is useful for fine-tuning the model. In practice, if early stopping is going to be employed, nearly the same information can be obtained (more efficiently) by just computing the hyperparameters and Hessian at the *completion* of model fitting, by setting “ARD LastEpoch” (see 6.4).

In the two examples above, the relevance of the bias units was excluded (the default), to facilitate direct comparison among predictive effects. If the user desires to make comparison of predictor effect magnitudes with the bias effect (something like relative importance compared to predicting just a mean output value), set “BiasRelevance YES”; here, the result would have been:

Input	Relevance
^^^^	^^^^^^
bias	34.98%
1	0.20%
2	2.93%
3	28.46%
4	33.43%

While the *ratio* of the predictor relevances of input 3 to input 4 is unchanged, it can now be seen that they each contribute on the same order of magnitude as (roughly) the mean prediction. A statistical comparison would require fitting a model without predictors, and computing a statistic comparing that nested model to the full model.

5.6.1.6. FILE SAVING ACTIVITY REPORT. If weights, prediction, and/or imputation files were specified in the .net file, NevProp3 reports that activity after completing those tasks. Here is an example, wherein previous .wts and .ptr files already existed (note: in fully interactive mode, the user is prompted for permission to overwrite existing file names):

```
... Overwriting file "Iris.wts"
... Weights saved to "Iris.wts"
... Overwriting file "Iris.ptr"
... Predictions saved to "Iris.ptr"
... Train set imputations saved to "iMSG.itr"
```

5.6.1.7. SUMMARY STATISTICS OF PREDICTIVE ACCURACY. Based on the final model, measures of accuracy are displayed for the training dataset (if a SaveTrainPrdFile was specified) and the testing dataset (if a ReadTestFile was specified). These summary measures always include the criterion function used, and an appropriate transformation of the criterion to a 0-1 scale (an R-squared statistic). As a reminder, a key to the indexes is appended after each run.

The accuracy measures are described in more detail in section 5.6.2.

For example, here are the summary statistics from a run using Iris.net at its default settings, wherein the criterion function is cross-entropy:

```
On TRAIN SET (nCases=150; nOutputs=1):
  ACrEn=0.0363  R2-N=0.971  ASqEr=0.0104  AThEr=0.0133  C index=0.9990

Key to indexes:
^^^^^^^^^^^^^^
  ACrEn (avg -log likelihood): max likelihood objective function
  (BEST=0).
  Nag-R2: 0-1 measure of uncertainty explained by the model (BEST=1).
           (A monotonic, but nonlinear transformation of the ACrEn.)
  ASqEr (Brier Score): joint error in calibration & discrimination
  (BEST=0).
  AThEr: fraction misclassified, outside ScoreThreshold*[dichot range]
  (BEST=1).
  C index (area under ROC curve): .5-1 nonparam discrimination measure
  (BEST=1).
```

Here is the same model, fit forcing squared-error as the criterion:

```
On TRAIN SET (nCases=150; nOutputs=1):
  ASqEr=0.0111  R2=0.950  AThEr=0.00667  C index=0.9984

Key to indexes:
^^^^^^^^^^^^^^
  ASqEr: joint measure of error in calibration & discrimination
  (BEST=0).
  R2: 0-1 measure of uncertainty explained by the model (BEST=1).
      (A linear transformation of the ASqEr.)
  AThEr: fraction of predictions beyond ScoreThreshold*[target range]
  (BEST=0).
  C index (area under ROC curve): .5-1 nonparam discrimination measure
  (BEST=1).
```

If the dependent variable had not been dichotomous, the C index would not have appeared in the display.

5.6.1.8. BOOTSTRAP-BASED BIAS CORRECTION. If `NBoots` > 0 in the `.net` file, that number of bootstrapped models will be fit, to estimate a bias correction to the measures of predictive accuracy (see 7.8.4). To ascertain that sufficient boots were performed to stabilize the bias correction estimates, first inspect the graphic display of the R^2 index:

[illegible]

```

*

*
_____
Minimum Cumulative Mean Bias = 0.042

```

The bias appears to have stabilized by the time 2/3 of the 200 bootstrapped models were averaged. The subsequent BOOTSTRAP RESULTS table displays a line for each measure of accuracy. The first numeric column, labeled (A), shows the accuracy measure estimated from the model trained on the complete data. Column (B) shows the mean value of the same statistic across all bootstrapped models, each tested with its own bootstrapped dataset. Column (C) shows the mean value of the same statistic across all bootstrapped models, this time tested with the full, complete dataset. Because each bootstrapped model was fit with only about 63% of the unique cases in the full dataset, accuracy measures in column (C) will be worse than in (B). The next column shows this difference as the BIAS, which is shown in the next column subtracted from column (A). The last column reports the standard deviation of the bootstrapped measures, an estimate how the full model measures would vary under resampling.

```

***** BOOTSTRAP RESULTS
*****
Number of boots: 200

      (A)      (B)      (C)      (B-C)      A-(B-C) (of B)
Model:  FULL   BOOT   BOOT   (B-C)      CORR.      STD.
Data:  FULL   BOOT   FULL   BIAS      INDEX      ERROR
^-----^
MCE           0.081   0.041   0.134   -0.093   0.174   0.0341
NAG-R2        0.942   0.971   0.891    0.080   0.862   0.0250
Brier         0.026   0.013   0.027   -0.014   0.039   0.0104
c Index       0.996   0.998   0.995    0.003   0.993   0.0022

*****
*****

```

The table above contains the accuracy measure reported when the criterion function is cross entropy (negative log-likelihood). When the criterion function is squared error, the usual R^2 replaces the Nagelkerke R^2 , and average squared error replaces the cross entropy and Brier Score (see also 5.6.2, below).

5.6.1.9. MEAN EFFECTS DISPLAY. If `NEffectBoots > 0` in the `.net` file, that number of bootstraps of the original data will be used to estimate the mean effects (including nonlinearities and interactions) of each predictor (see 7.8.5). The inference drawn in this situation relates, then, only to how the full model effects might perform on future data (drawn from a similar, larger, population). If, additionally, `NBoots > 0`, then `NEffectBoots` will be generated for *each* bootstrapped model. The confidence

intervals so generated relate to effects to be expected upon drawing data and using that data each time to form a new model. The choice of inference depends upon the hypotheses being evaluated.

Mean effects are demonstrated in detail in section 6.2. Below is an example of the display of such a run using *Iris.net*. Mean effects are computed sequentially for each predictor. Because of the computational intensity, NevProp3 feeds back to the user an asterisk (*) after completing each NEffectBoot. Because, in this example, there are 4 predictors, 4 progress lines are reported.

```
0% complete*****100%
complete
0% complete*****100%
complete
0% complete*****100%
complete
0% complete*****100%
complete
```

The subsequent mean effects report begins with a reminder of the settings. Here, NEffectBoots of 250 were performed for each of the 50 (NBoots) booted models, for a total of 12,500 effects for each predictor. (Because each effect computation involves a loop through the dataset, the processing time is proportional to the number of cases in the dataset.)

```
##### MEAN EFFECTS REPORT
#####

DEPENDENT VARIABLE (output) # 1 : "Species1" (binary)
BOOTSTRAPPED SAMPLES used to derive CI's (NBoots): 50
SUB-BOOTSTRAPS used per MODEL (NEffectBoots): 250
TOTAL NUMBER OF BOOTS used for each CI: 50 x 250 = 12500
```

Next, for each predictive variable, is shown a sequence of statistics. First, the mean effect (the average over all computed effects). If the dependent variable was binary, the effect is also represented as an odds ratio (see 2.6 for rationale). In either case, a 95% confidence interval for the mean or odds ratio effect is presented. If NEffectBoots < 200, the mean effects are used to calculate a standard deviation, which is used to derive the confidence intervals. If NEffectBoots = 200, the actual distribution of booted effects is used to form a confidence interval.

The confidence interval on the main line relates to the full model's effects under NEffectBoots; if NBoots>0, a subsequent line displays the (always broader) confidence interval based on the NBoots X NEffectBoots effects that were generated.

VARIABLE*	Mean	Odds	95% Confid. Interval	Nonlin-
Mean	Effect	Ratio	for Odds Ratio**	earity
Predict	^^^^^	^^^^	^^^^^^^^^^^^^^^^^^^^	^^^^^^
^^^^^^				
1. SepLen	-0.4596	0.6316	(0.5781, 0.6944)	1.0539
0.4971				

Over all boot models	(0.3884, 1.0269)
----------------------	--------------------

Following these lines is displayed the empiric values of the *full-model* mean effects, by percentile. Note: percentiles are always displayed on the original mean effects scale, and therefore will not be equal to odds ratio confidence limits.

Percentile:	0.05	0.10	0.25	0.50	0.75	0.90	0.95
Mean Effect:	-0.5317	-0.5198	-0.4892	-0.4578	-0.4267	-0.4023	-0.3867

This allows the analyst to estimate the shape of the distribution near the tails, and the adequacy of the confidence limits. If `NEffectBoots`<200 (i.e., a Gaussian distribution was assumed), it is especially useful to compare the confidence limits to the 0.05 and 0.95 percentile values (or exponentiated percentile values, in the case of odds ratios). If the values do not reasonably correspond, consider repeating the modeling process using several-fold greater `NEffectBoots` (if `.wts` were saved, the full model can be retrieved without any new fitting— only the `NEffectBoots` will be recomputed).

Next is shown a relative estimate of overall interaction nonlinearity. This is simply the coefficient of variation (standard deviation over all effects, divided by the mean effect). In linear models, there is only a single effect, so that the “nonlinearity” score will be zero (see example in tutorial 4.1).

Lastly, inference on the mean prediction, rather than the effect, is displayed. Sections 2.5 and 2.6 further discuss the relevance of difference ways to draw inference on models.

The subsequent lines display the same statistics, but now for levels of each predictor. If the predictor has greater than 10 unique values in the dataset, `NevProp3` considers the variable to be continuously-valued, and breaks the dataset into quartiles based on those levels. For predictors with less than 10 levels, `NevProp3` shows the statistics for each natural level represented in the dataset.

Q4	(7.52)	-0.1982		0.5480	(0.4734, 0.6262)		1.7351
0.4387							
Over all boot models				(0.2720, 1.1039)			
Q3	(6.80)	-0.3941		0.6775	(0.6218, 0.7350)		0.9196
0.4678							
Over all boot models				(0.4718, 0.9729)			
Q2	(6.10)	-0.6593		0.9796	(0.9748, 0.9840)		0.8453
0.4979							
Over all boot models				(0.9575, 1.0021)			
Q1	(6.06)	-0.3036		1.0000			1.2890
0.4996							

The following displays the mean effect statistics for the remaining 3 predictors:

2. SepWid	-0.7417		0.4763	(0.4422, 0.5067)		0.4567	
0.4971							
Over all boot models				(0.2411, 0.9412)			

Percentile:	0.05	0.10	0.25	0.50	0.75	0.90	0.95
Mean Effect:	-0.8042	-0.7916	-0.7697	-0.7428	-0.7183	-0.7014	-0.6894

```

-----
      Q4 ( 3.60) -0.5294 | 0.5854 ( 0.5516, 0.6238 ) | 0.5205
0.4707
      Over all boot models ( 0.3842, 0.8921 )
      Q3 ( 3.15) -0.7944 | 0.7886 ( 0.7702, 0.8092 ) | 0.4026
0.4883
      Over all boot models ( 0.6566, 0.9471 )
      Q2 ( 2.75) -0.7507 | 1.0741 ( 1.0632, 1.0846 ) | 0.4384
0.5035
      Over all boot models ( 0.9902, 1.1652 )
      Q1 ( 2.85) -0.6795 | 1.0000 | 0.5910
0.4998

      3. PetLen 2.7478 | 15.609 ( 10.212, 23.142 ) | 0.6900
0.4971
      Over all boot models ( 7.7074, 31.610 )
-----
Percentile: 0.05 0.10 0.25 0.50 0.75 0.90 0.95
Mean Effect: 2.3865 2.4639 2.6066 2.7581 2.8761 3.0371 3.0994
-----

      Q4 ( 6.30) 1.3332 | 440.99 ( 249.88, 890.60 ) | 0.9270
0.9896
      Over all boot models ( 116.18, 1673.9 )
      Q3 ( 5.35) 3.1901 | 51.442 ( 36.572, 79.447 ) | 0.5966
0.6927
      Over all boot models ( 20.283, 130.47 )
      Q2 ( 4.55) 3.9309 | 2.9805 ( 2.6604, 3.3577 ) | 0.3938
0.3164
      Over all boot models ( 1.8496, 4.8027 )
      Q1 ( 4.11) 1.0615 | 1.0000 | 0.5215
0.1545

      4. PetWid 4.8078 | 122.46 ( 64.591, 259.29 ) | 0.6820
0.4971
      Over all boot models ( 19.459, 770.73 )
-----
Percentile: 0.05 0.10 0.25 0.50 0.75 0.90 0.95
Mean Effect: 4.2686 4.3979 4.5851 4.7781 5.0188 5.2496 5.3852
-----

      Q4 ( 2.40) 1.8902 | 83.737 ( 52.749, 127.65 ) | 0.5624
0.9633
      Over all boot models ( 22.656, 309.49 )
      Q3 ( 2.05) 4.0722 | 29.496 ( 21.647, 40.606 ) | 0.5815
0.8001
      Over all boot models ( 11.729, 74.177 )
      Q2 ( 1.65) 8.4376 | 2.4164 ( 2.2548, 2.6424 ) | 0.2987
0.4672
      Over all boot models ( 1.7199, 3.3951 )
      Q1 ( 1.50) 3.3263 | 1.0000 | 0.6320
0.3503

* L = level# (value at that level); Q# = quartile# (mean value).
** Odds ratio of mean effect , relative to lowest level or quartile.

##### END OF MEAN EFFECTS REPORT
#####

```

Notice that the first two predictors, based on measurements of the sepals, have much weaker and statistically marginal effects than the two predictors based on petal measurements. Section 6.2 describes these issues in greater detail.

5.6.2 Understanding NevProp3's Measures of Accuracy

Based on the final model, measures of accuracy are displayed for the training dataset (if a `SaveTrainPrdFile` was specified) and the testing dataset (if a `ReadTestFile` was specified).

In the case of multiple dependent variables, the summary measure is internally computed for each. By default, only the average across all dependent variables is reported. However, accuracy values for specific dependent variables can still be reported out, using the `OutputStatVars` setting).

MEASURE: **ACrEn** (Average Cross Entropy)

DESCRIPTION: Average Cross Entropy is the negative log likelihood of the data. Under maximum likelihood optimization, the goal is to find values of the parameters controlling the shape of the regression function (see Chapter 2), such that the joint probability of the observed data is most likely. Call the vector of these unknown

parameters $\mathbf{b} = \{b_0, b_1, b_1, \dots, b_n\}$ to be estimated from a sample n observations Y_1, \dots, Y_n . For the i^{th} observation, we assume there exists some function, $f(Y_i; \mathbf{b})$, that describes its probability of occurring (or for a continuously-valued Y_i , its probability density). Then, for any estimate of \mathbf{b} , the joint likelihood of a given data set is simply $L(\mathbf{b}) = \prod_{i=1}^n f(Y_i; \mathbf{b})$. Now, the optimization function that maximizes $L(\mathbf{b})$ will also

maximize its negative logarithm, $-\log L(\mathbf{b}) = -\sum_{i=1}^n f(Y_i; \mathbf{b})$, which is easier to deal with numerically. For a binary dependent variable,

AVAILABILITY: Only if *all* dependent variables are dichotomous, and `OptimizeMethod` is specified as 2 or 3.

MEASURE: **ASqEr** (Average, or Mean, Squared Error)

DESCRIPTION: ASqEr is the mean of the squared differences between predictions and their corresponding dependent variable target. Used as the criterion when conditions for ACrEn are not met, under assumption of normally distributed residual error.

AVAILABILITY: Always (sometimes called the **Brier Score** when used with dichotomous dependent variables).

MEASURE: **R²**

DESCRIPTION: R^2 is a linear transformation of the ASqEr, forced to a 0-1 range. R^2 is commonly interpreted as the fraction of variance explained by the model. The "zero" reference level assumes a model will, at worst, predict the mean of the dependent variable (ie, only a bias input, with no informative predictors). It is therefore possible

that, on a new data set with different mean, an overfitted model will actually have a negative R^2 . (i.e., predict worse than the constant mean of the original model).

AVAILABILITY: When ASqEr is the criterion.

MEASURE: **R^2 -N** (Nagelkerke R^2)

DESCRIPTION: R^2 -N is a nonlinear (but monotonic) transformation of the ACrEn, forced to a 0-1 range. It therefore measures both calibration and discrimination. Because the ACrEn criterion does not require normally distributed errors, there is no real equivalent to the linear total amount of “variance” explained. However, this transformation facilitates comparison between models, on a given dataset. The “zero” reference level assumes a model will, at worst, predict the mean of the dependent variable (ie, only a bias input, with no informative predictors).

AVAILABILITY: When ACrEn is the criterion.

MEASURE: **AThEr** (Average Thresholded Error)

DESCRIPTION: AThEr is the average, across all output variables, of the fraction of case predictions *not* within the ScoreThreshold setting (see 7.8.3) of the corresponding targets. ScoreThreshold is the portion of an output variable’s range (maximum - minimum) within which a prediction must fall to be classified as a correct (i.e., within the threshold). As optimization proceeds, predictions get closer to their targets, so that the fraction of training case predictions entering the ScoreThreshold*range window increases (i.e., the AThEr decreases). For a binary output variable, the range is one, so that the ScoreThreshold itself is the range within which a prediction must fall to be classified as correct. In this probability (classification) model, the best setting for ScoreThreshold is determined by the costs of false positive and false negative classification (for this reason, the default is set to 0.5, assuming uniform costs).

AVAILABILITY: Always.

MEASURE: **C index** (Concordance Index)

DESCRIPTION: The C (concordance) index is a nonparametric measure of discrimination (the ability to separate output categories). The C index is approximately equal to the area under the receiver operating characteristic curve. In the NevProp3 implementation, the C index is applied only to dichotomous dependent variables. It is the probability that the classification model will assign a higher probability to a case with the higher outcome label (e.g., 1) than to that with a lower label (e.g., 0); i.e., that the model ordinarily discriminates among cases from differing classes. It can be visualized as follows: A pair of cases, one from each outcome category, is repeatedly and randomly drawn (with replacement) from the dataset. The predictions of each pair are compared. The predictions are said to be concordant if the case with the higher outcome value (say, 1) also has the higher predicted probability. Ties are given half a concordance credit. The C index is 0.5 when the predicted probability for the cases is random with respect to their target outputs, and rises to a maximum of 1.0 if all cases are concordant.

AVAILABILITY: When CalccIndex is set to YES, and all dependent variables are dichotomous (if CalccIndex is YES, NevProp3 automatically screens the dependent variables at the start of the run to ensure they are all dichotomous).

&5.7 Single Dichotomous Output (dependent) Variable (binary classification; probability model)

Prediction of scores bounded between 2 limits is a valuable procedure. Examples include probabilistic classification, and prediction scores bounded between low and high limits (like test scores). Because of these constraints, a logistic-shaped (rather than linear) output activation should be used, to prevent predictions from lying outside the limits. These outliers effect not only the final prediction evident to the user, but the entire optimization process leading to the final prediction. The appropriate logistic output unit type can be forced, or NevProp3 can scan the output and automatically set the appropriate function by using "OutputUnitType 3" (see 7.4.4).

The variance of a probability prediction (i.e., with only 2 discrete values for the dependent variable) is smallest near the limits, and maximal near 0.5. The proper criterion function is, therefore, one that seeks to maximize the probability of correct classification (i.e., maximum likelihood) without an assumption of uniform, let alone normally, distributed residual errors. For this reason, NevProp3 will maximize the likelihood (cross-entropy) when an appropriate "TrainCriterion" is set, or NevProp3 can scan the output and automatically set the appropriate function by using the default, "TrainCriterion 3" (see 7.5.1).

For an example of predicting a single binary dependent variable, see the tutorial in section 4.2.

&5.8 Multiple Dichotomous Output Variables (1-of-N vs M-of-N classification)

When all output units are binary, the constraints and NevProp3 assignment procedures described in 5.7 still apply. An additional consideration depends the nature of the problem to be solved: must the sum of all outputs be constrained to equal 1 (1-of-N probabilistic classification), or can there be overlapping probabilities of membership across the output categories (M-of-N probabilistic classification)?

By default, NevProp3 assumes M-of-N probabilistic classification. If 1-of-N classification is the appropriate scenario for the problem at hand, the user must set "1ofN YES" in the .net file. This invokes a normalizing function during optimization that forces outputs to sum to one, analogous to the Softmax concept.

For example, consider the Iris data in the file `Iris.net`. The file contains 3 binary dependent variables, representing the putative species of each flower; the 4 predictors are measurements made on each flower. Because the delineation of distinct species is to some degree arbitrary, it is not at all obvious for this dataset, whether 1-of-N (unique assignment) or M-of-N (overlapping assignment) is appropriate. Therefore, let's examine both approaches.

In the tutorial (section 4.2), only the first variable is used (1 indicates membership in that flower species, 0 indicates membership in one of the other two species). From a similar NevProp3 run, here are excerpts from the settings, and the corresponding summary statistics :

```
# DATA FILE SETTINGS
  OutputColumns 6 /*vars 1-4 are measurements, 5 is noise, 6-8 are
species*/
  OutputStatVars 0 /*same as setting it to 6, since it's the only
output*/
# CONFIGURATION SETTINGS
```

Ninputs	4	Nhidden	3	Noutputs	1
		lofN	NO		

```
On TRAIN SET (nCases=150; nOutputs=1):
ACrEn=0.0487 R2-N=0.960 ASqEr=0.0150 AThEr=0.0267 C index=0.9988
```

AutoTrain was used, so the stopping point reflects stopping of the training process to improve generalization. In the next run, we specify that all 3 outputs be considered (for now, leaving lofN at its default value of NO):

```
# DATA FILE SETTINGS
  OutputColumns 6 7 8 /* all 3 outputs now used for optimization */
  OutputStatVars 6 /* so we can look separately at variable 6, the
first
                        of the 3 outputs */
# CONFIGURATION SETTINGS
  Ninputs 4      Nhidden 3      Noutputs 3 /* increased
to 3*/
                        lofN NO /* allowing overlapping probabilities
*/
```

```
Subset summary statistics on variables 6
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
On TRAIN SET (nCases=150; nOutputs=1):
ACrEn=0.0841 R2-N=0.929 ASqEr=0.0211 AThEr=0.0400 C index=0.9988
```

This still used the M-of-N classification scheme, but we added parameters (weights) from each of the hidden units to the two additional outputs, without increasing the number of hidden units. The model is therefore relatively less complex, given it's task of distinguishing three rather than one species. The result is that the average cross entropy (negative log likelihood) error is somewhat higher, which this is not necessarily meaningful because of the nonlinearity of this score at low values. Similarly, the thresholded error indicates that 6 (.04 x 150) cases rather than 4 (.0267 x 150) cases were not within .5 of their targets, likely due to cases already near the threshold. Notice, however, that the c index, an ordinal measure of concordance, is the same— so the relative predictive values are unchanged.

We now set “lofN YES” to achieve 1 of 3 classification (that is, probabilities across each case must sum to 1):

```
# DATA FILE SETTINGS
  OutputColumns 6 7 8
  OutputStatVars 6
# CONFIGURATION SETTINGS
  Ninputs 4      Nhidden 3      Noutputs 3
                        lofN YES /* forcing 1-of-3 classification */
```

```
Subset summary statistics on variables 6
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
On TRAIN SET (nCases=150; nOutputs=1):
ACrEn=0.350 R2-N=0.606 ASqEr=0.0933 AThEr=0.00667 C index=0.9988
```

Under the 1-of-N classification scheme, the average cross entropy and thresholded errors have moved in a favorable direction. The c index remains the same (in this dataset, it would be hard to improve upon). In general, these 150 flowers are easy to separate into 3 distinct species, with only a few borderline cases. The 1-of-N is therefore more appropriate than M-of-N classification.

&5.9 Continuous Outputs

If all outputs may be considered continuously valued and reasonably unbounded, a linear output activation function is appropriate. The linear output unit type can be forced, or NevProp3 can scan the output and automatically set the appropriate function by using by using the default, "OutputUnitType 3" (see 7.4.4). An example is provided in the tutorial in section 4.1.

In NevProp3, the variance of a continuous prediction is assumed to be uniform and roughly Gaussian. (If the variance is known to change systematically as a function of the output value, an appropriate mathematical transformation should first be applied; see any advanced text on linear regression.) A proper scoring function is, therefore, one that reflects squared deviation of the prediction from the true output target. For this reason, NevProp3 will minimize the squared error when an "TrainCriterion 0" is set, or NevProp3 can scan the output and automatically set the appropriate function by using the default, "TrainCriterion 3" (see 7.5.1).

&5.10 Mixed dichotomous & Nondichotomous outputs

In the present version of NevProp3, the user cannot manually assign combinations of linear and logistic output unit types. However, by leaving the default setting of "OutputUnitType 3" (see 7.4.4), NevProp3 will scan all outputs, then internally assign logistic activation units to dichotomously-valued dependent variables, and linear activation units to all others. As feedback, the assignment of output unit types is reported during the initial display of setting values during a run (see 5.6.1).

Training criterion functions cannot be individually assigned. In the current implementation, the squared error will be minimized across all dependent variables when mixed output types are detected. Ideally, a common likelihood scale would be computed by transformations of cross-entropy (from dichotomous outputs) and squared error (from nondichotomous outputs).

CHAPTER &6. ADVANCED TOPICS

&6.1 Bootstrapped-Corrected Model Performance & Standard Errors

Statistical inference is the process of extrapolating to a much larger universe of data, of which we observe only the dataset at hand. As discussed in chapter 2, two types of inference may be addressed, depending on real-world hypotheses:

COMPARATIVE INFERENCE. How well do competing models explain the observations? This is especially clear when the models conform to physical scientific models.

PARAMETRIC & PREDICTIVE INFERENCE. Given an accepted model formulation, what are reasonable values that the parameters should take (*parametric inference*)? What predictions may safely be drawn when this model is presented with new observations (*predictive inference*)?

Data-Splitting. What these processes share is the need to use existing data. If a portion of the available data has been held out and never used in any way for model development, the predictive validity of the models can be tested (validated). Occasionally, validation can be performed by awaiting prospective collection of new data. Usually, though, quality data is hard to come by, due to technical and economic constraints. Furthermore, the split may be fortuitously good or bad— a different split could yield a different estimate of validity. And the optimal proportion for splitting is problem-dependent. Lastly, validation on the held-out test set yields only point estimates of performance, rather than confidence intervals. For all these reasons, it is undesirable to split available data into separate model-fitting and model-testing sets.

Two “resampling” techniques are commonly used to estimate the impact of model variability without holding back data:

Cross-Validation. Cross-validation splits the data into 2 or more subsets, allowing model development and testing on separate subsets. The extreme would be leaving one case out at a time, developing a model on the remaining $N-1$ cases, then repeating this process for all N cases (taking the average across all tests as the estimate of performance)¹⁸. Grouped cross-validation (k -fold) may be more accurate¹⁹. Performance on subsets can be averaged to yield overall accuracy estimates and variances. Cross-validation suffers from the need to determine how to best split the data. And the entire dataset is never used to generate the final model. Additionally, it is not commonly appreciated that to achieve accurate estimates, the entire splitting process must be repeated many times²⁰. Although NevProp3 uses a form of cross-validation as part of the AutoTrain procedure, the technique is used only to estimate a stopping target for subsequent model fitting on the full dataset.

Bootstrapping. A more efficient and generally applicable approach to resampled model validation is called bootstrapping. An excellent practical overview is presented in a recent text by Efron and Tibshirani²¹. A more developed application of the bootstrap to linear model validation can be found in a course syllabus by Harrell²². To bootstrap a dataset of size N , simply perform N draws (with replacement). The new “booted” dataset has N cases, although some are replicates, and not all of the original cases are represented. This process is repeated B times, to create B booted datasets, each of size N . The booted datasets are samples of the original dataset, in the same “fractal” sense that the original data is a sample of the larger universe of data to which we wish to generalize. Unfortunately, we have only one sample of the universe of data, but we can draw an almost unlimited number of booted samples. This allows us to explore the behavior of a distribution of booted models, and derive statistics that apply approximately to the behavior of the original dataset relative to the larger universe of data.

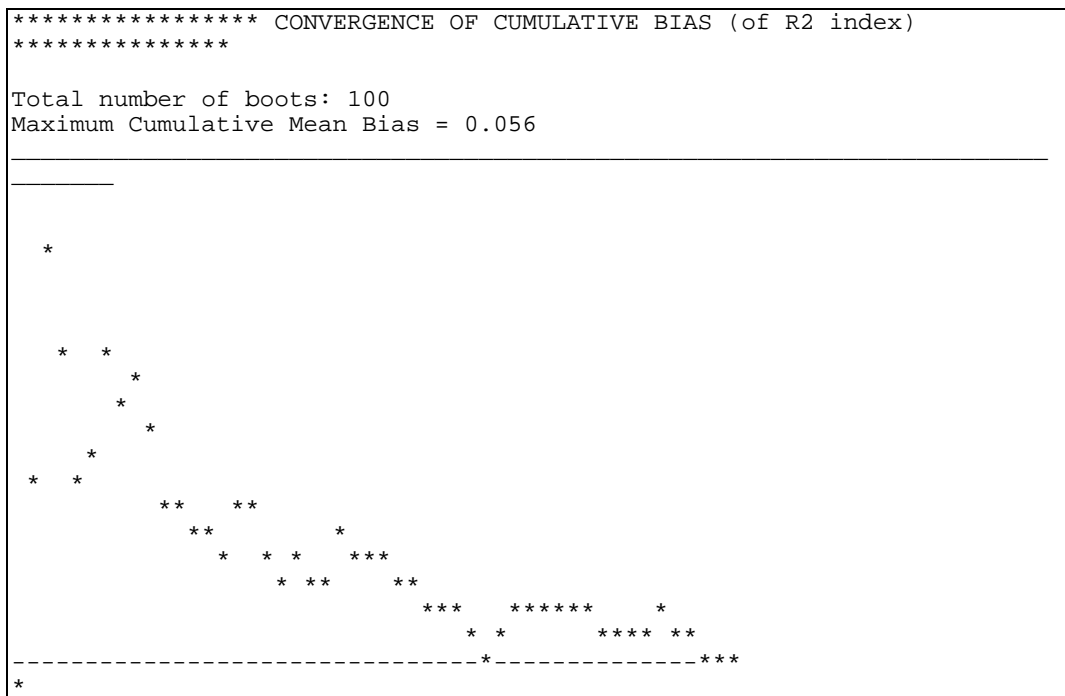
In the case of model validation, we can develop the full model using all the data, and compute various measures of accuracy. It is likely, however, that despite our best efforts to limit overfitting, the measures of accuracy are still optimistically biased, because there is no independent test set. To estimate the optimism (which can be subtracted from the accuracy scores of the full model), we use the B booted samples to develop B booted models (using exactly the same steps as for the full model; the process must be automated, because subjective decisions invalidate the fractal relationship of booted to full models).

Now we compute the optimism for each measure of accuracy. First, for each bootstrapped model, we compute the measure of accuracy using the bootstrapped dataset used to create that model. That is analogous to the way we use the full model with the original data to estimate the full model accuracy. Next, we pass the original dataset through each of the bootstrapped models. Because the original data contains cases not used to develop each bootstrapped model, this process is fractally analogous to having further samples from the larger universe of data to pass through our full model. Each bootstrapped model now gives us an estimate of optimistic bias: simply subtract the accuracy obtained by passing the bootstrapped sample through, from that obtained by passing the original sample through. Do this for all B models, and compute the average and variance of these bias estimates. Lastly, correct the full model accuracy by subtracting the average bootstrapped optimistic bias. Additionally, the bootstrapped variance can be used as an estimate of the variance of the full model.

Tutorial 4.1 presents an example of bootstrap-based bias-correction for a synthetic linear model. Below, we show the bias-correction for the classification problem of tutorial 4.2. This is based on the Iris dataset, supplied with NevProp3.

NevProp3 first fits the full model and generates appropriate measures of accuracy (see 5.6.2). Then, if `NBoots > 0` in the .net file, NevProp3 uses the identical training procedure to create NBoots bootstrapped samples and corresponding models. Using the procedure described above, the optimistic bias is estimated.

Because of the sampling-based process, the optimistic bias will vary across bootstrapped models. The cumulative average will ultimately converge, however. To ensure that a sufficient NBoots were specified, NevProp3 first displays the average bias as a function of the number of bootstrapped models processed. The R2 index is used for this purpose, because it can be computed for both classification and continuous prediction problems.



Minimum Cumulative Mean Bias = 0.031

The graph, above, shows that although the range of cumulative mean bias ranged from .031 to .056, it settled to around .033 about 2/3 the way through the 100 NBoots. In general, stable optimistic bias estimation requires at least 50 NBoots, and almost never more than 200.

Next, the accuracy adjustment process is explicitly summarize in tabular format. Each line represents one measure of accuracy. The first numeric column is the accuracy measure computed on the full model using the full (original) dataset. The next two columns present the accuracy measures using the booted models– first passing through the booted samples used to created each model, then passing through the full dataset. The fourth column is the difference of these values, which is subtracted from the first column in the next column. The last column is an estimate of the standard deviation of the accuracy measure, derived from the variance of the booted model/booted sample accuracies.

***** BOOTSTRAP RESULTS *****							

Number of boots: 100							
INDEX	Model:	(A)	(B)	(C)	(B-C)	A-(B-C)	(of B)
*****	Data:	FULL	BOOT	BOOT		CORR.	STD.
*****		FULL	BOOT	FULL	BIAS	INDEX	ERROR
*****		*****	*****	*****	*****	*****	*****
MCE		0.041	0.031	0.069	-0.038	0.079	0.0197
NAG-R2		0.967	0.975	0.942	0.033	0.934	0.0166
Brier		0.014	0.010	0.018	-0.008	0.022	0.0060
c Index		0.999	0.999	0.998	0.001	0.998	0.0009

Notice that the bias may be positive or negative, depending on the directionality of the accuracy measure. For example, because MCE (mean cross entropy; negative log likelihood) values closer to zero are better, the optimistic bias will be negative, serving to elevate the MCE when subtracted. On the other hand, the Nagelkerke R2 improves as it approaches 1; the bias is therefore a positive value.

Here, the accuracy scores are minimally adjusted (with 2 standard deviation units for each measure). Given the excellent performances, this model is likely to generalize well to future data, and the effects should be trusted. But was the nonlinear ANN really needed?

Let's compare the model above (nonlinear ANN with 3 hidden units, regularized by stopped-training via AutoTrain) to 2 alternative models: the same architecture by without any regularization, and the simple linear model (NHidden set to 0, no regularization):

		(A)	(B)	(C)	(B-C)	A-(B-C)	(of B)
	Model:	FULL	BOOT	BOOT		CORR.	STD.
INDEX	Data:	FULL	BOOT	FULL	BIAS	INDEX	ERROR
^^^^^^		^^^^	^^^^	^^^^	^^^^	^^^^	^^^^
MCE (NoReg)		0.008	0.007	0.117	-0.109	0.117	0.0078
MCE (AutoTr)		0.041	0.031	0.069	-0.038	0.079	0.0197
MCE (lin)		0.040	0.024	0.063	-0.038	0.079	0.0161
NAG-R2 (NoReg)		0.994	0.994	0.893	0.101	0.892	0.0063
NAG-R2 (AutoTr)		0.967	0.975	0.942	0.033	0.934	0.0166
NAG-R2 (lin)		0.968	0.980	0.947	0.033	0.935	0.0133
Brier (NoReg)		0.004	0.004	0.020	-0.017	0.021	0.0027
Brier (AutoTr)		0.014	0.010	0.018	-0.008	0.022	0.0060
Brier (lin)		0.013	0.008	0.017	-0.009	0.022	0.0051
c Index (NoReg)		1.000	1.000	0.997	0.003	0.997	0.0003
c Index (AutoTr)		0.999	0.999	0.998	0.001	0.998	0.0009
c Index (lin)		0.999	1.000	0.998	0.002	0.998	0.0005

In the absence of regularization, the model is clearly overfitted. Although the full model accuracies are higher than for the other 2 models, the corresponding biases are even greater. The result is that the *corrected* indexes are poorer than with regularization. Thus the inferiority of the complex but unregularized model is apparent.

Notice also that there are no meaningful differences in the measures or corrections between the AutoTrained and pure linear models. This suggests that the nonlinear ANN regularization constrained the model to produce essentially linear effects.

Correctly classifying the few aberrant cases that are misclassified by both AutoTrained and linear models causes such a severe distortion of the decision surfaces (as in the nonregularized model) that the model is untenable. The take-home message is that the aberrant cases were either mismeasured or misclassified by the biologist. The model should not be forced to compensate!

&6.2 Parametric Inference— Determining Mean Effects of Predictors

6.2.1 Overview

As discussed in section 2.2, parametric inference is particularly useful when the parameters correspond to real phenomena. Parameters are usually interpretable only for generalized linear and additive models (see section 2.5). Unfortunately, the presence of interactions greatly limits the interpretability of even the simplest linear model: if a new variable is coded as the product of 2 other variables, there is no longer a *single* statistical “effect” of either variable.

Here is a listing of the inference methods discussed in sections 2.5 and 2.6:

METHOD 1. SIGN and SIGNIFICANCE of the EFFECT (all models)

METHOD 2. PREDICTED VALUES of the MODEL (continuous prediction models)

METHOD 3. MARGINAL EFFECTS of the PARAMETER VALUES (all models)

METHOD 4. PREDICTED PROBABILITIES given a set of cases (probability models)

METHOD 5. MARGINAL EFFECTS on the PROBABILITY (probability models)

“Effect” here refers to *statistical* effect (i.e., association), not necessarily causal relationship. We present these inference methods in the remainder of this section, first on a simulated dataset with a continuous dependent variable, then on the Iris dataset with a binary output.

6.2.2 EXAMPLE 1. Linear dependent variable.

Graphical and nomographical methods may provide insight into interactive relationships. Below, we’ve expanded `myfirst.net` (see tutorial 4.1) to include 200 cases. This simple dataset contains only 2 predictors of the duration of hospital stay: the patient’s age and cholesterol level. Using a statistical package, we have split data according to age (young vs. old) to created a binary predictor. We then superimpose the separately run linear regressions of cholesterol on hospital stay, for the two age groups:

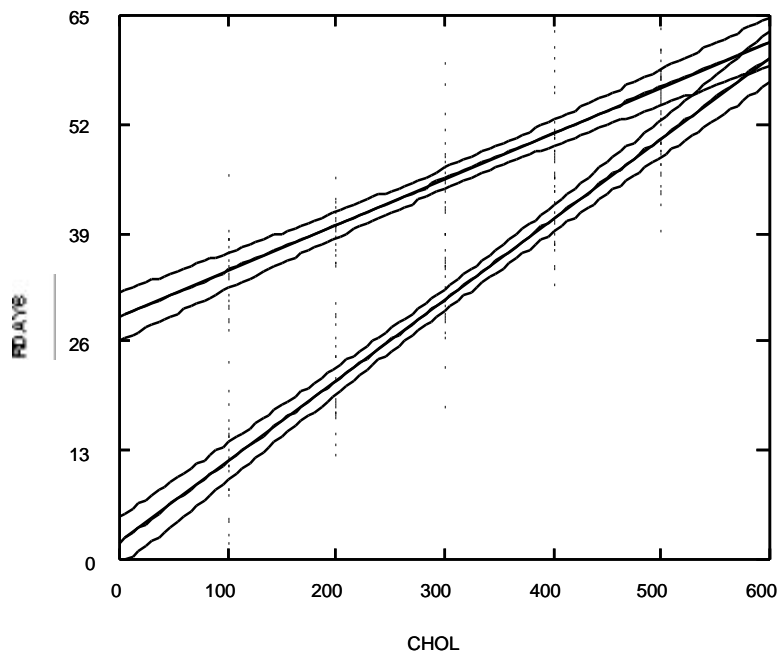


Figure 6.2.1.1 Superposition of two regressions of cholesterol on the number of days of hospital stay for myocardial infarction. Upper regression, older patients; lower regression, younger patients; surrounding each regression line are estimated 95% confidence bands.

In this synthetic example, it seems clear that patients with higher cholesterol end up staying longer in the hospital, whether young or old. Although the predicted hospital stay is shorter for younger patients, the slope (effect) of cholesterol is greater. That is, younger patients in general recover more quickly from a heart attack, but higher cholesterol elevates the risk of complications more strongly than in the elderly. Given these findings, one could generate a hypothesis such as that, in young patients, high cholesterol is a marker for other genetic tendencies (like hypertension and diabetes) that lead to complications (like infection, peripheral vascular disease, and stroke).

Unfortunately, datasets rarely contain only 2 variables, so that 2-way graphical exploration is of limited value. Looking at the model-fitting summary from the statistical package for a linear model that ignores the putative interaction, we have:

DEP VAR:	RDAY5	N:	200	MULTIPLE R:	0.911	SQUARED MULTIPLE R:	0.829
ADJUSTED SQUARED MULTIPLE R:	0.828	STANDARD ERROR OF ESTIMATE:	5.902				
VARIABLE P(2 TAIL)	COEFFICIENT	STD ERROR	STD COEF	TOLERANCE	T		
CONSTANT	8.286	1.064	0.000	.	7.787	0.000	
CHOL	0.076	0.003	0.755	1.000	25.647	0.000	
AGE	14.443	0.835	0.509	1.000	17.303	0.000	

Figure 6.2.1.2 Linear regression summary. No interaction is coded.

The common way to model a putative interaction in a GLM is to create a synthetic variable by multiplying the 2 predictors:

DEP VAR:	RDAY5	N:	200	MULTIPLE R:	0.934	SQUARED MULTIPLE R:	0.873
ADJUSTED SQUARED MULTIPLE R:	0.871	STANDARD ERROR OF ESTIMATE:	5.107				
VARIABLE P(2 TAIL)	COEFFICIENT	STD ERROR	STD COEF	TOLERANCE	T		
CONSTANT	2.010	1.198	0.000	.	1.678	0.095	
CHOL	0.097	0.004	0.964	0.500	26.750	0.000	
AGE	26.994	1.694	0.952	0.182	15.936	0.000	
CHOL_AGE	-0.042	0.005	-0.532	0.154	-8.191	0.000	

Figure 6.2.1.3 Linear regression summary. Interaction is coded as a new multiplicative variable.

Regarding **METHOD 1** (SIGN and SIGNIFICANCE of the EFFECT), we see that both linear models show a positive sign for cholesterol and age effects. That is, the higher cholesterol, and older age predict a longer duration of hospital stay after a heart attack. Both would be considered strongly significant, with very small 2-sided probabilities of type I error (false positive association).

The interaction effect (CHOL_AGE) in figure 6.2.1.3 is also significant, and a test of the nested difference in errors accounted for by the two models would show the reduction to be statistically significant. But now consider **METHOD 3** (MARGINAL EFFECTS of the PARAMETER VALUES)– for every 100 units increment in cholesterol, there is an age-adjusted prolongation of hospital stay of $100 \times 0.076 = 7.6$ days in the first model, and a modestly longer projection of 9.7 days according to the model that includes the interaction. Comparing younger (age=0) to older (age=1) groups, the two models predict cholesterol-adjusted stays of about 14 days versus 27 days. Clearly, ignoring the multiplicative interaction term now tremendously distorts the effect of age, and to a lesser degree that of cholesterol (although a difference of 2 days could be the difference between profit and loss for a health care system!).

The problem is that the the model with an interaction term no longer allows a single summative parameter to interpret.

Aside from graphical models, one could now use **METHOD 2** (PREDICTED VALUES of the MODEL). This information could be generated from a standard GLM statistical package or spreadsheet, by providing X values for the fitted regression formula. For example, for the linear models above:

Model without interaction term:

$$\text{Predicted stay} = 8.286 + 0.076 \cdot \text{CHOL} + 14.443 \cdot \text{AGE} \quad (6.2.1)$$

Model with interaction term:

$$\text{Predicted stay} = 2.01 + 0.097 \cdot \text{CHOL} + 26.994 \cdot \text{AGE} - .042 \cdot \text{CHOL_AGE} \quad (6.2.2)$$

Unfortunately, to get an output, need to supply values of *all* predictors– essentially, we lose the benefit of examining only the marginal effects, which are “controlled” for other variables. So, if we want to look at the stay as a function of cholesterol, we need to specify CHOL and AGE. If we took either AGE=0 or AGE=1, we are just plucking off the subgroup predictions. We can use the actual data for each cholesterol level, it is the same as taking the average age (0.5, since half the cases belong to each group). Here is what we get:

Linear model, no interaction. (based on 6.2.1)			
		(assumes AGE fixed at 0.5)	(assumes CHOL fixed at 300)
CHOL	DAYS	AGE	DAYS
100	23.1	Young	31.1
200	30.7	Old	45.5
300	38.3		
400	45.9		
500	53.5		

Figure 6.2.1.4 Predictions of duration of hospital stay, based on linear model without interaction.

Linear model, with interaction. (based on 6.2.2)			
		(assumes AGE fixed at 0.5)	(assumes CHOL fixed at 300)
CHOL	DAYS	AGE	DAYS
100	23.1	Young	31.1
200	30.7	Old	45.5
300	38.3		
400	45.9		
500	53.5		

Figure 6.2.1.5 Predictions of duration of hospital stay, based on linear model with multiplicative interaction.

Notice that the tables, above, show the same predictions. Because these are GLMs, we will get identical predictions for any model, including those with interaction terms. We could just as well have cross-tabulated the dataset and computed cell means, without ever fitting a

regression model. From a predictive standpoint, the benefit of using **METHOD 2** on a linear model is that we can *interpolate* for future predictions (for example, validating the model on an independent test set). In addition, it is useful to generate these predictive averages for comparison with ANN subgroup predictions— see figure 6.2.1.9)

Despite the three methods available, the inference problem remains because the GLM with an artificially-introduced interaction term does not provide a *single* summative parameter to interpret.

Artificial neural networks overcome this obstacle by allowing a natural interaction of effects. Only the original predictors are used— no recoded synthetic variables. But the price of this more mechanistically proper ANN model is the inclusion of intermediary parameters. Work at our center has provided the mathematics to marginalize over these intermediary parameters, to yield a single estimate of the effect of each predictor (see section 2.6).

Before exploring effects, it is important that the final model is well fitted and generalizable (see section 6.1). Thereafter, setting `NEffectBoots>0` in the NevProp3 .net file invokes Mean Effects computations (see also section 2.6). This module uses bootstrapping of the dataset to estimate confidence intervals. Bootstrapping the data without booted models (i.e., `NBoots` is set to 0) assumes the final model is “correct”— uncertainty can arise only from variability in the process of data sampling. If uncertainty in model development is relevant to the problem, it is necessary to set `NBoots>0`; this usually requires no extra NevProp3 computation, because `NBoots` should already have been generated to correct for optimistic bias (as in 6.1). NevProp3 will internally preserve and use these booted models and their corresponding data resamples to estimate the (broader) confidence intervals associated with simultaneous uncertainty in data sampling *and* model fitting.

Before proceeding, please review section 5.6.1.9, which describes the basic layout and terminology of NevProp3’s mean effects module— this will not be repeated here.

Continuing with the example above, we now run an ANN model with two hidden units, regularized by early stopping with AutoTrain. We first check the model fit and bias:

***** BOOTSTRAP RESULTS *****							

Number of boots: 50							
		(A)	(B)	(C)	(B-C)	A-(B-C)	(of B)
	Model:	FULL	BOOT	BOOT		CORR.	STD.
INDEX	Data:	FULL	BOOT	FULL	BIAS	INDEX	ERROR
^^^^^^		^^^^^	^^^^^	^^^^^	^^^^^	^^^^^	^^^^^
ASqEr		25.556	25.597	27.252	-1.654	27.210	2.7963
R2		0.873	0.873	0.864	0.008	0.865	0.0139
##### MEAN EFFECTS REPORT #####							
#####							

Figure 6.2.1.6 Model fitting summary for nonlinear ANN regularized by early stopping with AutoTrain.

The R^2 of 0.873 is identical to that in figure 6.2.1.3. This is the best any model *could* do, because the data represent an intentionally-simulated multiplicative interaction. The nonlinear ANN not only captured the multiplicative interaction, but the regularization prevented fitting of the noise (i.e., there was no substantial optimistic bias).

Now inspect the mean effects summary, below (see section 5.6.1.9 for interpretation of the format and terminology). This was generated by setting `NEffectBoots` to 1000 and `NBoots` to 50.

DEPENDENT VARIABLE (output) # 1 : "RDays" (nonbinary)							
BOOTSTRAPPED SAMPLES used to derive CI's (NBoots): 50							
SUB-BOOTSTRAPS used per MODEL (NEffectBoots): 1000							
VARIABLE*	Mean	95.% Confid. Interval				Nonlin-	Mean
	Effect	for mean effect				earity	Predict
^^^^^^	^^^^	^^^^^^^^^^^^^^^^^^^^				^^^^	^^^^
1. Chol	0.0734	(0.0700, 0.0770)				0.3562	38.255
Over all boot models		(0.0621, 0.0847)					

Percentile:	0.05	0.10	0.25	0.50	0.75	0.90	0.95
Mean Effect:	0.0703	0.0710	0.0721	0.0733	0.0747	0.0756	0.0763

L5 (500.)	0.0666	(0.0651, 0.0680)				0.0750	52.579
Over all boot models		(0.0408, 0.0923)					
L4 (400.)	0.0546	(0.0463, 0.0617)				0.4390	46.669
Over all boot models		(0.0046, 0.1047)					
L3 (300.)	0.0990	(0.0867, 0.1114)				0.4219	39.017
Over all boot models		(0.0592, 0.1388)					
L2 (200.)	0.0754	(0.0730, 0.0775)				0.0928	30.135
Over all boot models		(0.0421, 0.1087)					
L1 (100.)	0.0713	(0.0712, 0.0714)				0.0047	22.900

2. Age	29.073	(23.488, 34.948)				1.4951	38.255
Over all boot models		(18.720, 39.426)					

Percentile:	0.05	0.10	0.25	0.50	0.75	0.90	0.95
Mean Effect:	24.501	25.494	27.115	29.022	31.047	32.817	34.002

L2 (1.00)	53.056	(43.092, 63.376)				0.9671	45.501
Over all boot models		(34.274, 71.839)					
L1 (0.00)	5.0899	(4.7487, 5.3885)				0.3303	31.026

* L = level# (value at that level); Q# = quartile# (mean value).							
##### END OF MEAN EFFECTS REPORT							
#####							

Figure 6.2.1.7 Mean effects summary. For AGE, L1 (coded as 0) represents the younger group, L2 (coded 1) the older group.

First, compare the confidence intervals based only on data bootstrapping (those on the same line as the variable label and mean effect) to those based on both data and model bootstrapping (printed on the subsequent line). There is remarkably little expansion of the confidence intervals, increasing our confidence that this ANN was sufficiently complex to capture true effect nonlinearities, yet appropriately regularized to prevent overfitting. This supports the findings of the optimistic bias table in figure 6.2.1.6.

Regarding **METHOD 1** (SIGN and SIGNIFICANCE of the EFFECT), we see that both linear models show a positive sign for cholesterol and age effects. That is, the higher cholesterol, and older age predict a longer duration of hospital stay after a heart attack. Both would be considered strongly significant, with very small 2-sided probabilities of type I error (false positive association).

Next, compare the overall mean effect sign (**METHOD 1**) and marginal effect (**METHOD 3**) for each predictor in the ANN to that in the linear models. Here is an excerpted summary:

A. Linear model, no interaction.

VARIABLE	COEFFICIENT	STD ERROR	95% CONFIDENCE INTERVAL
CHOL	0.076	0.003	(0.070, 0.082)
AGE	14.443	0.835	(12.806, 16.080)

B. Linear model, multiplicative interaction.

CHOL	0.097	0.004	(0.089, 0.105)
AGE	26.994	1.694	(23.674, 30.314)
CHOL_AGE	-0.042	0.005	(-0.052, -0.032)

C. Nonlinear ANN model.

VARIABLE*	Mean Effect	95.% Confid. Interval for mean effect	Nonlin- earity
*****	*****	*****	*****
1. Chol	0.0734	(0.0700, 0.0770)	0.3562
2. Age	29.073	(23.488, 34.948)	1.4951

Figure 6.2.1.8 Comparison of overall effects of three models. For linear models, confidence intervals were computed as the coefficient $\pm 1.96 \times \text{STD_ERROR}$.

We know from the design of the dataset that, for either age group alone (or as a collapsed, single group) the relationship of cholesterol to hospital stay is purely linear (figure 6.2.1.1). This effect is properly represented in the simple linear model (.076) and in the ANN (.073), but is distorted in the linear model with interaction (.097) due to the introduction of the artificial variable CHOL_AGE.

Next, we know also that age dominates the nonlinear effect. Ignoring the nonlinearity, the simple linear model finds effect of about 14 days lengthened stay for the older group. The linear model with interaction splits the effect between AGE and CHOL_AGE, and is interpretable only by separately considering the 2 age groups (essentially, by reconstructing figure 6.2.1.1). The ANN, however, yields a mean effect, or “importance” of the age, of about twice that in the linear model. That is, compared with the corresponding simple linear model, the ANN results show that: (1) cholesterol is primarily a linear effect, and, (2) the linear and interactive effects of age result, overall, in a marginal increased length of stay of about 29 days.

The nonlinearity score (coefficient of variation of the mean effects across all cases—*unrelated* to the variance used to compute the confidence interval) likewise reflects these relationships. Because of the inherent mixing of effects in an ANN, there remains a small case-to-case coefficient of variation of about .36 for the effect of cholesterol. However, the ANN balances these variations across cases to achieve a net linear effect. For the effect of age, the ANN permitted a relative 4-fold greater nonlinearity. If there were additional predictors in the dataset, the cholesterol nonlinearity of .36 could serve as a reference value for comparison of the corresponding effects.

We can also explore the nonlinearities by examining the effects withing each quartile of cholesterol, and for each of the two age groups. Figure 6.2.1.7 clearly shows a very similar effect across quartiles of cholesterol with no systematic trends (as would be expected for a linear effect). On the other hand, the effect of age differs 10-fold between age groups, reflecting the fact that older age nonlinearly increases the hospital stay.

Now let's revisit **METHOD 2** of effect inference. The right-most column of the Mean Effects display (figure 6.2.1.7) shows the average value of the length of stay prediction, across cases contained in that level of the predictor. Figure 6.2.1.9 compares the average case predictions of the linear the ANN models.

	LINEAR	ANN		LINEAR	ANN
<u>CHOL</u>	<u>DAYS</u>	<u>DAYS</u>	<u>AGE</u>	<u>DAYS</u>	<u>DAYS</u>
100	23.1	22.9	Young	31.1	31.0
200	30.7	30.1	Old	45.5	45.5
300	38.3	39.0			
400	45.9	46.7			
500	53.5	52.6			

Figure 6.2.1.9 Predictions of duration of hospital stay, comparing linear and ANN models. For linear model, age is assumed fixed at .5 when cholesterol is predictor, and cholesterol fixed at 300 when age is predictor.

The ANN subset predictions closely match that of the linear models, despite the potential nonlinearities introduced by the hidden units (9 parameters rather than 3). Once again, this confirms that regularization was successful.

6.2.3 EXAMPLE 2. Binary dependent variable.

The format of the probabilistic model Mean Effects report is described in detail in section 5.6.1.9. Here, we draw inference on the effect of the predictors of species classification using the Iris dataset.

The inference methods from sections 2.5 and 2.6 that are applicable here are:

METHOD 1. SIGN and SIGNIFICANCE of the EFFECT (all models)

METHOD 3. MARGINAL EFFECTS of the PARAMETER VALUES (all models)

METHOD 4. PREDICTED PROBABILITIES given a set of cases (probability models)

METHOD 5. MARGINAL EFFECTS on the PROBABILITY (probability models)

Figures 6.2.3.1 and 6.2.3.2 show the NevProp3 Mean Effects Report for a linear and nonlinear ANN model, respectively. The results are based on the full and 100 bootstrapped models and, on each of those, 2000 bootstrapped estimates of mean effect (defined in section 2.6).

##### MEAN EFFECTS REPORT #####							
DEPENDENT VARIABLE (output) # 1 : "Species1" (binary) BOOTSTRAPPED SAMPLES used to derive CI's (NBoots): 100 SUB-BOOTSTRAPS used per MODEL (NEffectBoots): 2000							
VARIABLE*	Mean	Odds	95% Confid. Interval			Nonlin-	
Mean	Effect	Ratio	for Odds Ratio**			earity	
Predict	^^^^^^	^^^^^	^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^			^^^^^^	
1. SepLen	-0.1566	0.8550	(0.8378, 0.8723)			0.7983	
0.3258	Over all boot models		(0.2167, 3.3730)				

Percentile:	0.05	0.10	0.25	0.50	0.75	0.90 0.95	
Mean Effect:	-0.1728	-0.1697	-0.1635	-0.1565	-0.1501	-0.1434 -0.1394	

Q4 (7.29)	-0.1738	0.6732	(0.6539, 0.6920)			0.4760	
0.3198	Over all boot models		(0.0233, 19.486)				
Q3 (6.35)	-0.2576	0.8245	(0.8083, 0.8410)			0.2547	
0.3254	Over all boot models		(0.1472, 4.6166)				
Q2 (5.55)	-0.1629	0.9755	(0.9703, 0.9807)			0.7540	
0.3302	Over all boot models		(0.8082, 1.1775)				
Q1 (5.29)	-0.0262	1.0000				2.6958	
0.3317							
2. SepWid	-2.3986	0.0908	(0.0636, 0.1288)			0.8976	
0.3258	Over all boot models		(0.0060, 1.3665)				

Percentile:	0.05	0.10	0.25	0.50	0.75	0.90 0.95	
Mean Effect:	-2.6936	-2.6271	-2.5140	-2.3955	-2.2764	-2.1800 -2.1083	

Q4 (4.01)	-0.3771	0.1395	(0.0956, 0.2036)			2.7283	
0.2015	Over all boot models		(0.0102, 1.9114)				
Q3 (3.40)	-1.3127	0.2344	(0.1911, 0.2867)			1.4587	
0.2712	Over all boot models		(0.0464, 1.1846)				
Q2 (2.90)	-3.2221	0.7284	(0.6997, 0.7557)			0.6522	
0.3174	Over all boot models		(0.5263, 1.0082)				
Q1 (2.80)	-3.1152	1.0000				0.4965	
0.3256							


```

3.  PetLen      2.5553 | 12.875 ( 9.1515, 18.085 ) | 0.8430
0.3258
  Over all boot models      ( 1.0875, 152.44 )
-----
Percentile:    0.05    0.10    0.25    0.50    0.75    0.90    0.95
Mean Effect:  2.2658  2.3306  2.4383  2.5609  2.6786  2.7760  2.8412
-----
      Q4 ( 6.12)  2.8573 | 207808 ( 113742, 393423 ) | 0.5459
0.6126
  Over all boot models      ( 1.5169, 3.e+10 )
      Q3 ( 4.95)  4.8252 | 2277.6 ( 1563.9, 3330.5 ) | 0.2438
0.3263
  Over all boot models      ( 0.9235, 6.e+06 )
      Q2 ( 3.95)  3.3903 | 37.456 ( 29.559, 47.764 ) | 0.1573
0.1565
  Over all boot models      ( 0.6061, 2314.8 )
      Q1 ( 1.90)  0.1445 | 1.0000 | 3.0797
0.0033

4.  PetWid      4.8243 | 124.50 ( 45.721, 371.80 ) | 1.3556
0.3258
  Over all boot models      ( 3.7191, 4167.7 )
-----
Percentile:    0.05    0.10    0.25    0.50    0.75    0.90    0.95
Mean Effect:  3.9706  4.1433  4.4853  4.8242  5.2034  5.5297  5.7393
-----
      Q4 ( 2.25)  4.1988 | 35771. ( 9950.4, 139276 ) | 0.9123
0.6618
  Over all boot models      ( 45.889, 3.e+07 )
      Q3 ( 1.70)  13.938 | 244.05 ( 132.68, 463.29 ) | 0.5127
0.3128
  Over all boot models      ( 3.5245, 16899. )
      Q2 ( 1.20)  3.1272 | 3.4253 ( 2.9938, 4.0823 ) | 0.3985
0.0669
  Over all boot models      ( 0.4427, 26.500 )
      Q1 ( 0.42)  0.0297 | 1.0000 | 0.4614
0.0179

5.  ZRandom      0.0064 | 1.0064 ( 1.0056, 1.0073 ) | 0.8243
0.3258
  Over all boot models      ( 0.6511, 1.5557 )
-----
Percentile:    0.05    0.10    0.25    0.50    0.75    0.90    0.95
Mean Effect:  0.0057  0.0059  0.0061  0.0064  0.0067  0.0070  0.0071
-----
      Q4 ( 1.22)  0.0068 | 1.0161 ( 1.0141, 1.0182 ) | 0.8406
0.3261
  Over all boot models      ( 0.3585, 2.8797 )
      Q3 ( 0.27)  0.0078 | 1.0091 ( 1.0075, 1.0107 ) | 0.6194
0.3260
  Over all boot models      ( 0.5507, 1.8489 )
      Q2 (-0.26)  0.0055 | 1.0055 ( 1.0044, 1.0067 ) | 0.9680
0.3259
  Over all boot models      ( 0.6885, 1.4686 )
      Q1 (-1.25)  0.0056 | 1.0000 | 0.9160
0.3257

* L = level# (value at that level); Q# = quartile# (mean value).
** Odds ratio of mean effect , relative to lowest level or quartile.
##### END OF MEAN EFFECTS REPORT
#####

```

Figure 6.2.2.1 Mean effects summary. Linear model Iris (no hidden units) of four flower part measurements to predict species (species 1 vs species 2).

MEAN EFFECTS REPORT							
DEPENDENT VARIABLE (output) # 1 : "Species1" (binary)							
BOOTSTRAPPED SAMPLES used to derive CI's (NBoots): 100							
SUB-BOOTSTRAPS used per MODEL (NEffectBoots): 2000							
VARIABLE*	Mean	Odds	95% Confid. Interval			Nonlin-	
Mean	Effect	Ratio	for Odds Ratio**			earity	
Predict	^^^^^^	^^^^^	^^^^^^^^^^^^^^^^^^^^^^^^^^^^			^^^^^^	
1. SepLen	-1.7677	0.1707	(0.1707, 0.1707)			0.0000	
0.3335	Over all boot models			(0.0041, 7.1108)			

Percentile:	0.05	0.10	0.25	0.50	0.75	0.90 0.95	
Mean Effect:	-1.7677	-1.7677	-1.7677	-1.7677	-1.7677	-1.7677	

Q4 (7.29)	-1.7677	0.0290	(0.0290, 0.0290)			0.0000	
0.3000	Over all boot models			(0.0000, 50.812)			
Q3 (6.35)	-1.7677	0.1529	(0.1529, 0.1529)			0.0000	
0.3271	Over all boot models			(0.0029, 8.0383)			
Q2 (5.55)	-1.7677	0.6288	(0.6288, 0.6288)			0.0000	
0.3487	Over all boot models			(0.2362, 1.6735)			
Q1 (5.29)	-1.7677	1.0000				0.0000	
0.3550							
2. SepWid	-5.4314	0.0044	(0.0044, 0.0044)			0.0000	
0.3335	Over all boot models			(0.0000, 0.3888)			

Percentile:	0.05	0.10	0.25	0.50	0.75	0.90 0.95	
Mean Effect:	-5.4314	-5.4314	-5.4314	-5.4314	-5.4314	-5.4314	

Q4 (4.01)	-5.4314	0.0014	(0.0014, 0.0014)			0.0000	
0.2192	Over all boot models			(0.0000, 0.3175)			
Q3 (3.40)	-5.4314	0.0384	(0.0384, 0.0384)			0.0000	
0.2811	Over all boot models			(0.0026, 0.5673)			
Q2 (2.90)	-5.4314	0.5809	(0.5809, 0.5809)			0.0000	
0.3242	Over all boot models			(0.3709, 0.9098)			
Q1 (2.80)	-5.4314	1.0000				0.0000	
0.3319							

```

3.  PetLen      7.2284 | 1378.0 ( 1378.0, 1378.0 ) | 0.0000
0.3335
  Over all boot models      ( 4.5322, 418950 )
-----
Percentile:    0.05    0.10    0.25    0.50    0.75    0.90    0.95
Mean Effect:  7.2284  7.2284  7.2284  7.2284  7.2284  7.2284  7.2284
-----
      Q4 ( 6.12) 7.2284 | 2.e+13 ( 2.e+13, 2.e+13 ) | 0.0000
0.6546
  Over all boot models      ( 592.78, 6.e+23 )
      Q3 ( 4.95) 7.2284 | 4.e+09 ( 4.e+09, 4.e+09 ) | 0.0000
0.3180
  Over all boot models      ( 100.40, 1.e+17 )
      Q2 ( 3.95) 7.2284 | 3.e+06 ( 3.e+06, 3.e+06 ) | 0.0000
0.0752
  Over all boot models      ( 22.153, 3.e+11 )
      Q1 ( 1.90) 7.2284 | 1.0000 | 0.0000
0.0000

4.  PetWid     14.439 | 2.e+06 ( 2.e+06, 2.e+06 ) | 0.0000
0.3335
  Over all boot models      ( 383.64, 9.e+09 )
-----
Percentile:    0.05    0.10    0.25    0.50    0.75    0.90    0.95
Mean Effect:  14.439  14.439  14.439  14.439  14.439  14.439  14.439
-----
      Q4 ( 2.25) 14.439 | 3.e+11 ( 3.e+11, 3.e+11 ) | 0.0000
0.6497
  Over all boot models      ( 53527., 2.e+18 )
      Q3 ( 1.70) 14.439 | 1.e+08 ( 1.e+08, 1.e+08 ) | 0.0000
0.3362
  Over all boot models      ( 2029.6, 6.e+12 )
      Q2 ( 1.20) 14.439 | 77841. ( 77841., 77841. ) | 0.0000
0.0525
  Over all boot models      ( 103.62, 6.e+07 )
      Q1 ( 0.42) 14.439 | 1.0000 | 0.0000
0.0001

5.  ZRandom    -0.8971 | 0.4077 ( 0.4077, 0.4077 ) | 0.0000
0.3335
  Over all boot models      ( 0.0818, 2.0325 )
-----
Percentile:    0.05    0.10    0.25    0.50    0.75    0.90    0.95
Mean Effect: -0.8971 -0.8971 -0.8971 -0.8971 -0.8971 -0.8971 -0.8971
-----
      Q4 ( 1.22) -0.8971 | 0.1091 ( 0.1091, 0.1091 ) | 0.0000
0.3194
  Over all boot models      ( 0.0021, 5.7659 )
      Q3 ( 0.27) -0.8971 | 0.2563 ( 0.2563, 0.2563 ) | 0.0000
0.3333
  Over all boot models      ( 0.0224, 2.9337 )
      Q2 (-0.26) -0.8971 | 0.4109 ( 0.4109, 0.4109 ) | 0.0000
0.3403
  Over all boot models      ( 0.0836, 2.0200 )
      Q1 (-1.25) -0.8971 | 1.0000 | 0.0000
0.3518

* L = level# (value at that level); Q# = quartile# (mean value).
** Odds ratio of mean effect , relative to lowest level or quartile.
##### END OF MEAN EFFECTS REPORT
#####

```

Figure 6.2.2.2 Mean effects summary. Nonlinear ANN Iris model (3 hidden units) of four flower part measurements to predict species (species 1 vs species 2).

[IN PROGRESS]

&6.3 Convergent and Non-Convergent Regularization to Improve Model Generalization

6.3.1 Overview of Regularization

A statistical model is generated from only a sample of data. A model that generalizes well is one that predicts accurately on the larger universe of potential measurements from which the sample is drawn. Of course, that universe of data is never available; we are fortunate when we can collect further sizable samples, and more often must wait for accumulation over time.

For reasons presented in section 6.1, we want to use *all* the available data to develop a model. Nonlinear ANNs present an opportunity to flexibly map the “true” underlying statistical relationships, but at the risk of overfitting noise or other irrelevant features. If the model is to perform well on future data samples, it is desirable that potential generalization performance be considered *during the optimization of the parameters*. This basically means that individual weights should only grow as large as necessary to match the true distribution of data— the larger a weight in an ANN, the more irregular the regression surface can become. Regularization, therefore, is the global process used to let each weight grow to an optimal size (and not more).

Regularization approaches fall into two very broad categories:

Nonconvergent methods indirectly affect optimization by causing it to cease before the criterion function would otherwise have converged. The nonconvergent method used in NevProp3 is to stop training based on a target derived from preliminary models developed by internal cross-validation (6.3.1, below).

Convergent methods that directly affect the criterion function. When residual errors become suitably asymptotic, the training process is halted (presumably at a functional minimum that reflects good generalization). Convergent methods used in NevProp3 include fixed penalty (“weight decay”) and Bayesian regularization (6.3.2, below).

6.3.2 Nonconvergent Regularization (Early Stopping)

We use the term “nonconvergent regularization” in narrow way, to refer to methods in which training is stopped before the criterion function would otherwise have, regardless of the method used to stop the training. Thus it is synonymous with “stopped training,” or more precisely, “early stopping.” Others²³ have used the term nonconvergent in a very broad way, to mean any method in which the training process itself has an effect on the regularization terms and their constants— essentially any procedure except fixed decay procedures.

In theory, stopped training methods can be quite effective. Wang et. al.²⁴ showed that stopped training methods were always consistent, whereas decay methods may not be. They showed that there is an optimal stopping time which guarantees generalization for a given network prior to convergence of the network on the training set.

The fact that an optimal stopping point exists does not mean that a stopped training procedure will find it. Amari et. al.²⁵ showed that stopped training produced some gain in generalization error if the optimal stopping point were accessed. A common method is to withhold and use part of the training data to estimate out-of-sample performance—and stop training at the point at which residual error begins to rise on the held-out set. But the exact point at which that occurs depends on the split between training and testing sets (high variance). Furthermore, withholding some of the data means that the entire training set is not used for the final model.

In NevProp3, we've addressed these issues by using a 2-part approach to early stopping, using a function we call "AutoTrain" (see 7.6.4). In phase I, the dataset is split apart multiple (`NSplits`) times (see 7.6.5). A model is fitted and tested for each split, and the training-set error at which the test set began to rise is stored (the "target"). In phase II, the complete data set is reconstituted, and a single, final model fitted. Training is stopped when the error criterion value drops to the mean value of the targets across the `NSplits` models. The motivation here is that while cross-validation procedures demonstrate relatively high variance, they are relatively unbiased; this serves the purpose of achieving an unbiased point-estimate of the mean target. In practice, 5-10 `NSplits` of the data in phase I usually suffices to create a cumulatively stable target for fitting the final model. Examples are given in tutorials 4.1 and 4.2.

6.3.3 Convergent Regularization (Fixed Penalty &ARD)

As discussed in section 2.2, the flexibility of ANN models is well-suited to address the Bayesian-motivated concern that distributions of parameters be considered. In fact, the fixed "weight decay" penalty used empirically in the ANN community corresponds to a penalty on large weights at the tails of a Gaussian weight distribution. By favoring smaller weights, the ANN regression surface is less curvy (more regular), and is therefore unlikely to be extruded to fit outliers and noise.

Maximum likelihood optimization seeks to find the set of parameters that maximize the joint likelihood of the data. In practice, the logarithm of this likelihood function is used, which translates into a goal of minimizing an additive criterion function. If we wish to implement a penalty based on the sizes of the weights, we may represent the hyperparametrized cost function as follows:

$$COST = \beta E_D + \alpha E_W \quad (6.3.3.1)$$

where E_D is the error component contributed by the likelihood of the data, and E_W is that contributed by the weight penalty.

As mentioned above, the simplest assumption is that small weights are desirable, and that the prior distribution is roughly Gaussian (centered at zero). A global weight penalty would then be a squared-error function, such as:

$$COST = \beta E_D + \alpha \left[\frac{1}{2} \sum_i w_i^2 \right] \quad (6.3.3.2)$$

Gradient optimization methods use the *derivative* of the COST function with respect to each weight (w_j) to determine the update in that weight. In the derivative of the simple

penalty formulation above, the other weight terms will drop out, leaving the same “fixed” β multiplying each weight.

In NevProp3, the fixed penalty formulation is implemented as the `WeightDecay` (see 7.5.10). Because a fixed penalty ignores the role played by individual weights in determining the curvature of the regression surface, we do not recommend it as a regularization technique. Set to a very low fraction, however, the fixed penalty can be allowed to enter only when weights are growing extremely high during model fitting: in an ANN with sigmoidal activation functions, the use of a small fixed penalty can prevent a massive weight along an individual input from overwhelming other inputs and thereby locking up the sigmoid’s upper or lower asymptotic extremes. Used in this fashion, the small fixed weight decay serves as an “outlier penalty” independent of the particular shape of the prior distribution of a weight.

In order to automate the process of assigning tailored penalties to individual weights, we need to make some assumptions about prior distributions of the weights and the residual error distribution of the likelihood function. Under the assumption of Gaussian residual error (reasonable for unbounded regression, but questionable for classification) and weight distributions, Mackay²⁶ has elaborated a computationally feasible method of recursively estimating individual weight penalties.

[IN PROGRESS]

&6.4 Variable Selection— Effects of Individual Predictors (ARD)

[IN PROGRESS]

&6.5 Prediction & Imputation Using ANN-kNN mode

In most dataset analyses, the goal is to generate a properly fitted model. The model’s parameters and effects may be of principle interest, or accurate prediction may be the objective. In either circumstance, the presence of data cases with missing elements forces the analyst to decide how to squeeze as much information as possible out of the observed data, without increasing bias or variance into the model. This subject is presented in more detail in sections 2.7 and 5.3.3.

If model-building is the main objective, it may not be necessary to explicitly know what values were imputed (substituted) for missing elements in the process model-fitting. On the other hand, making predictions on new data with missing elements requires a method of imputation.

In any event, an appropriate method should be used deal with missingness. Discarding cases with missing elements, or substituting unconditional means or medians is fraught with hazard (see 2.7). Substituting regression or nearest neighbor-based imputations is better, because these techniques use some information about the joint distribution of missing and observed data. Ideally, if the “true” joint distribution of missing and observed data was known, its parameters could be used to eliminate or directly impute missing elements. Alternatively, Bayesian subjective assumptions could be used to

generate distributions of such parameters. Unfortunately, except in artificial, simple, or constrained models, the joint distribution cannot be accurately expressed *a priori*.

The next best way to reconstruct the joint likelihood of missing and observed data is called Expectation-Maximization (EM). EM is an iterative approach to estimating the parameters of the joint likelihood. Under EM, a parametric form of joint likelihood must be put provided; the tentative parameters are used to generate expected values for the missing elements. These imputations are then used to update the likelihood function, which is, in turn used to update the expected values, and so on, until convergence. Many recent papers support the notion that the general EM formulation converges reliably to a maximum of the likelihood function. The price is computational intensity, which increases with the fraction of missing information.

For complex datasets, however, how does one find a suitable joint likelihood? Because of their parametric flexibility, artificial neural networks (ANNs) present a novel opportunity to use EM with missing data. We propose that an ANN be generated that creates the joint likelihood by use of a nearest neighbor principle. The basic approach is as follows:

1. Create an ANN architecture such that the input variables are also the output variables. For instance, if there were 3 predictors, there would be 3 new outputs as well. For the moment, ignore any dependent variables. Include a large number of intervening hidden units (appropriate regularization, such as AutoTrain, will limit overfitting the model).
2. Standardize the data, so that predictors are scaled similarly.
3. For each data record, find it's nearest (k) neighbor(s) by Euclidian distance. In this step, missing values are initialized using unconditionally imputed means or medians of the observed cases for each variable. (Alternatively, it is possible to use only complete cases as a starting point, later incorporating those with missing elements. This will be demonstrated below). For instance, let record i be represented by the vector $\{x_1, x_2, x_3, y\}_i$, where the first three elements are the predictor values, and the last element is an outcome measure. Consider a specific case, $\{2.9, 0.9, 1.7, 1\}$. To find its single nearest neighbor (assume $k=1$), scan the dataset for the record vector $\{x_1^*, x_2^*, x_3^*, y^*\}$ that produces the minimum value when tested by the formula $(2.9 - x_1^*)^2 + (0.9 - x_2^*)^2 + (1.7 - x_3^*)^2$. Let this nearest neighbor be $\{2.8, 0.7, 1.9, 1\}$. The dependent variable here is ignored, because we're only concerned with the joint distribution of predictors.
4. Construct a new dataset, consisting of k "extended" record(s) of each case. An extended record is created by appending, to each original record, its nearest neighbor vector. In our example, $\{2.9, 0.9, 1.7, 1\}$ would be replaced by $\{2.9, 0.9, 1.7, 1, 2.8, 0.7, 1.9, 1\}$. If there were k nearest neighbor of interest, the original vector would be replicated k times, each one extended by appending one of the k neighboring vectors.
5. Start the iteration by maximizing (M step) this network as if it were the final model (actually, just a little progress in optimization would suffice for the purposes of the EM algorithm). In our example, the three inputs (original vector) would be used to predict three outputs (nearest neighbor vector). This ANN model now contains a representation of the joint likelihood of the data, for the following reasons. Given an input record, the model estimates the nearest neighbor vector. In the limit of contiguous data points, the nearest neighbor converges to the index point. Therefore any characteristic (including values for missing elements) of the index can be approximated by that of the neighbor.

[IN PROGRESS]

&6.6 Ensemble of experts (adaptive weighting of predictions)

[IN PROGRESS]

&6.7 Interactions Among Predictors (separate NevGA application)

There are many plausible physiological interactions in science. For example, consider common risk factors for coronary heart disease. All other factors being equal, the mortality risk for an individual with both diabetes and dyslipidemia is greater than that predicted by adding the increased risk of the two disease independently. Many other examples can be found in texts on linear regression. Clearly, if we are interested in drawing (potentially causal) inference on the associations in our regression models, we must include such biologically plausible interaction effects.

As presented in section 6.2, a benefit of ANN over GLM models is that interactive effects among predictors are modeled a natural way. Using appropriate regularization and variable selection procedures, the final model should generalize well to future data. Given the complexity of the ANN model, however, how can the analyst identify which interactions are meaningful?

To address this question, we developed a separate program, called NevGA (for Nevada Genetic Algorithm) that uses (Darwinian) selection principles to efficiently (and automatically) sort through the myriad possible interactions among predictors. When documentation is complete, NevGA will appear in on the same anonymous FTP host as NevProp3.

Briefly, we propose that interactions among predictors can be identified by searching the distribution of prediction differences of a properly-regularized ANN and a GLM . This is because any significant differences in predictive performance must be due to predictor and/or effect non-linearity generated by the hidden units (i.e., interactions). Because a suitable differentiable objective function needed for gradient-based optimization is lacking, the space of differences in predictions among the two models is searched by a genetic algorithm. The GA is used to select those combinations of predictors which, if modeled multiplicatively in a GLM, would likely result in a statistically significant improvement in accuracy. As an example, a standard clinical dataset is studied, with the finding of a single 4-way interaction that accounted for twice the log likelihood, with greater statistical significance, than a textbook treatment using multiple two-way interaction terms in the model.

[IN PROGRESS]

CHAPTER &7. SETTINGS Reference

This chapter is intended as a reference for NevProp3 users. A more succinct summary may be found in the file, **Describe.net**, provided with the software. This file is actually a version of **Iris.net**, heavily commented to describe the meaning and options of each setting. This is intended to serve as a quick on-line reference during the operation of NevProp3.

&7.1 Defaults and bare-minimum Network (.net) file

DEFAULT SETTINGS. Default settings are assigned internally by NevProp3 for all but two settings: **Ninputs** and **Noutputs**. These must be explicitly declared by the user. These settings are needed to configure the network, and to parse lines of data (because a single case are allowed to extend over several lines). Default values are described in detail in section 7.2, below, and in the supplied file, **Describe.net**.

At the beginning of each NevProp3 run, the values of all settings are reported. This allows the user to confirm the values. Also, the display is formatted so that the text may be copied as-is and pasted to create a new **.net** file.

BARE-MINIMUM .net FILE. Assuming the default values for the settings are acceptable, a very minimal **.net** file could look something like the following. Here, it is assumed that the training data are placed in an external file called **baremin.trn** (alternatively, training data could be appended within the **.net** file, below a line containing the keyword, **DATA**).

```
##### Description #####
# "baremin.net" Minimal specification of settings.

Ninputs 4          Noutputs 1

ReadTrainFile    baremin.trn

##### End of baremin.net #####
```

The user has supplied the data file **baremin.trn** containing 150 cases, each with 5 variables: 4 columns of input (predictors), then a single column of outputs (dependent variable). Of course, the user would be accepting the default settings and architecture, as indicated at the beginning of the NevProp3 run:

```
##### Starting NevProp NP3r1
#####
FYI: OutputUnitType not in .net file -- set to 3 (auto).
!!!! Nhidden (number of hidden units) was not given in .net file
      Initializing to 2.
FYI: Connect was not explicitly set in .net file. Assuming...
      Connect 1 4          5 6
      Connect 5 6          7 7
FYI: FileFormat was not specified -- assuming v3 network file.
... SETTINGS were successfully read from "baremin.net".
... Training data was read from "baremin.trn".
```

```

FYI: InputColumns & OutputColumns were not individually specified--
     NVars will be set to the sum of Ninputs and Noutputs (5).
... SEED=846016433 using lrand48(),srand48()
... Read in 150 TRAIN CASES [random 75 cases (50%) to a holdout
subset]
# DATA FILE SETTINGS
  ReadTrainFile      baremin.trn
  NHeaders 0         IDColumn NO
  StandardizeInputs 1 SaveStandWts NO ImputeMissing discard
  InputColumns 0
  OutputColumns 0
  NVars 5            ShuffleData YES
# REPORTING SETTINGS
  DescribeVars YES
  NBoots 0            NEffectBoots 0
  CalccIndex YES      ScoreThreshold 0.5
  OutputStatVars 0
# CONNECT CALLS
  Connect 1 4 5 6
  Connect 5 6 7 7
# CONFIGURATION SETTINGS
  Ninputs 4           Nhidden 2           Noutputs 1
  kNN 0               lofN NO
  HiddenUnitType 1    OutputUnitType 3
  /* OutputUnitTypes assigned: 0 */
  WeightRange 0.001
# TRAINING SETTINGS
  TrainCriterion 3     /* TrainCriterion assigned: 0 */
  BiasPenalty NO       WeightDecay -0.001
  OptimizeMethod 1     SigmoidPrimeOffset 0
  QPMaxFactor 1.75     QPModeSwitchThreshold 0
  Stochastic NO        LearnRate 0.01     SplitLearnRate NO   Momentum
0.01
# BEST-BY-HOLDOUT SETTINGS
  PercentHoldout 50.00
  AutoTrain YES        MinEpochs 200      BeyondBestEpoch 2
  NSplits 5            SepBootXVal YES
# RELEVANCE DETERMINATION SETTINGS
  UseARD NO            WhenARD Auto        ARDTolerance 0.05   ARDFreq 25
  GroupSelection Input BiasRelevance NO    ARDFactor 1

```

&7.2 NP File Settings

7.2.1 ResultsFile (.res)

Format: **ResultsFile file_name[.res]**

Values: **file_name** any string, with or without the **.res** extension.

Default: (if **ResultsFile** is not specified, no results file is created)

Note: If the extension **.res** is not provided, it is assumed (the results file will have the **.res** extension). An alternative way to create a log is to redirect standard output to a file (on platforms that support this). Will over-write any existing file with the same name (and indicate so in the standard output).

Details: If a record of NevProp3 processing is desired, use of this setting will save an ASCII file log, in the same directory as NevProp3, with essentially the same results as that sent to standard output (usually, the terminal screen). A date-time stamp is included at the beginning and end of each run.

Sample: **ResultsFile Iris.res**
ResultsFile Iris (actual log file will be **Iris.res**)

7.2.2 SaveTrainPrdFile (.ptr)

Format: **SaveTrainPrdFile file_name[.ptr]**

Values: **file_name** any string, with or without the **.ptr** extension.

Default: (if **SaveTrainPrdFile** is not specified, no training predictions file is created)

Note: If the extension **.ptr** is not provided, it is assumed (the training predictions file will have the **.ptr** extension). Will over-write any existing file with the same name (and indicate so in the standard output).

Details: If declared, at the end of model training, NevProp3 will pass the complete training dataset through the current weights. If a training hold-out subset was specified (see 7.6.1), the active weights are those obtained at the epoch where minimal error was found on the training hold-out subset. If NevProp3 was trained on the entire training dataset, with or without AutoTrain (see 7.6.4), then the active weights are those at the final epoch.

Each line of the training prediction file will contain a row label ("SEQU"), and, corresponding to each output, predictions ("PRED#") based on the active weights, and the target output values ("TRUE#") copied from the training data. (If an output was dichotomous, but not valued as -.5,.5 or 0,1, and **OutputUnitType** 3 was specified, then rescaled 0,1 predictions and target value are saved to the file.)

If **NBoots** is >0, lower and upper 95% confidence limits for each case's prediction will be saved into two additional columns for each output, based on predictions across all bootstrapped models. For a linear output unit, the confidence limits are generated from the standard deviation of the predictions. For a logistic output unit, the confidence limits are generated from the standard deviation of the logits of the probabilities, which are then converted back to the probability scale using the inverse logit function. For instance, if **NBoots** was 100, each training case would be passed through each of the 100 booted models. The standard deviation is computed for the 100 predictions made for each case. The confidence limits are generated by subtracting and adding 1.96 times the case's standard deviation, to the full-model prediction (and inverting the logit, if the output was logistic).

Sample: **SaveTrainPrdFile Iris.ptr**
SaveTrainPrdFile Iris (actual file name will be **Iris.ptr**)

7.2.3 SaveTestPrdFile (.pts)

Format: **SaveTestPrdFile file_name[.pts]**

Values: **file_name** any string, with or without the **.pts** extension.

Default: (if **SaveTestPrdFile** is not specified, no testing predictions file is created)

Note: If the extension **.pts** is not provided, it is assumed (the training predictions file will have the **.pts** extension). Will over-write any existing file with the same name (and indicate so in the standard output).

Details: If declared, and if **ReadTestFile** is specified, NevProp3 will pass the testing dataset through the current weights. If a training hold-out subset was specified (see 7.6.1), the active weights are those obtained at the epoch where minimal error was found on the training hold-out subset. If NevProp3 was trained on the entire training dataset, with or without AutoTrain (see 7.6.4), then the active weights are those at the final epoch.

Each line of the training prediction file will contain a row label ("SEQU"), and, corresponding to each output, predictions ("PRED#") based on the active weights, and the target output values ("TRUE#") copied from the training data. (If an output was dichotomous, but not valued as -.5,.5 or 0,1, and **OutputUnitType** 3 was specified, then rescaled 0,1 predictions and target value are saved to the file.)

If no training was performed, but a **ReadWeightsFile** was specified, those weights will be used to generate the predictions (in this way, validation predictions may be created only on the weight set ultimately determined to define the best model).

At present, confidence limits for test file predictions *cannot* be generated, because NevProp3 does not externally save the bootstrapped models created from the training data.

Sample: **SaveTestPrdFile Iris.pts**
SaveTestPrdFile Iris (actual file name will be **Iris.pts**)

7.2.4 SaveTrainImputFile (.itr)

Format: **SaveTrainImputFile file_name[.itr]**
Values: **file_name** any string, with or without the **.itr** extension.
Default: (if **SaveTrainImputFile** is not specified, no file with imputed training data is created)
Note: If the extension **.itr** is not provided, it is assumed (the training predictions file will have the **.itr** extension). Will over-write any existing file with the same name (and indicate so in the standard output).
Details: If declared, NevProp3 will save a copy of the current training dataset, with any missing values imputed according to **ImputeMissing** (see 7.3.6), or the Expectation-Maximization kNN method (see 6.5 and 7.4.8). This may be useful to confirm the nature of the imputation, or to use the identically-imputed dataset in another modeling program or for other purposes.
Sample: **SaveTrainImputFile Iris.itr**
SaveTrainImputFile Iris (file name will be **Iris.itr**)

7.2.5 SaveTestImputFile (.its)

Format: **SaveTestImputFile file_name[.its]**
Values: **file_name** any string, with or without the **.its** extension.
Default: (if **SaveTestImputFile** is not specified, no file with imputed testing data is created)
Note: If the extension **.its** is not provided, it is assumed (the training predictions file will have the **.its** extension). Will over-write any existing file with the same name (and indicate so in the standard output).
Details: If declared, NevProp3 will save a copy of the testing dataset, with any missing values imputed according to **ImputeMissing** (see 7.3.6), or the Expectation-Maximization kNN method (see 6.5 and 7.4.8). This may be useful to impute a dataset for other purposes.
Sample: **SaveTestImputFile Iris.its**
SaveTestImputFile Iris (file name will be **Iris.its**)

7.2.6 SaveWeightsFile (.wts)

Format: **SaveWeightsFile** **file_name[.wts]**

Values: **file_name** any string, with or without the **.wts** extension.

Default: (if **SaveWeightsFile** is not specified, no weights file is created)

Note: If the extension **.wts** is not provided, it is assumed (the weights file will have the **.wts** extension). Will over-write any existing file with the same name (and indicate so in the standard output).

Details: If declared, NevProp3 will save the current weights to the file specified. Each line of the weights file will contain a pair of numbers indicating the connection between units (unit zero is the bias, or intercept unit), followed by the corresponding weight. Following the connection section, the mean value(s) of the output variable(s) is listed; this information will be used by NevProp3 to estimate the R2 statistic on validation (**.tst** file) data. Finally, if ARD is active (see 7.7), additional information about the hyperparameters is reported (for user inspection only— will be ignored if the weights file is subsequently read in for another NevProp3 run).

If a training hold-out subset was specified (see 7.6.1), the current weights are those obtained at the epoch where minimal error was found on the training hold-out subset. If NevProp3 trained on the entire training dataset, with or without AutoTrain (see 7.6.4), then the current weights are those at the final epoch.

Sample: **SaveWeightsFile Iris.wts**

SaveWeightsFile Iris (actual file name will be **Iris.wts**)

7.2.7 ReadWeightsFile (.wts)

Format: **ReadWeightsFile** **file_name[.wts]**

Values: **file_name** any string, with or without the **.wts** extension.

Default: (if **ReadWeightsFile** is not specified, no weights file is read in)

Note: If the extension **.wts** is not provided, it is assumed (the weights file must have the **.wts** extension). The **.wts** file must have been generated by a model whose structure is identical to that of the current model specified.

Details: If declared, NevProp3 will read the weights in the file specified. This may be performed to continue training from the established point in weight-space, or to pass a testing (validation) dataset through the model without further training (by setting the maximum number of epochs to zero when calling NevProp3).

Sample: **ReadWeightsFile Iris.wts**

ReadWeightsFile Iris (actual file name must be **Iris.wts**)

&7.3 DATA File Settings

7.3.1 ReadTrainFile (.trn)

Format: **ReadTrainFile file_name[.trn]**

Values: **file_name** any string, with or without the **.trn** extension.

Default: (if **ReadTrainFile** is not specified, training data must be appended in the **.net** file)

Note: The actual testing data file *must* have the extension **.trn**; if not provided, it is assumed. The training file must reside in the same directory as NevProp3 (or in the same directory as an alias linked to the executable NevProp3 kept in another directory). A specified **.trn** file will be ignored if the keyword **DATA** appears within the **.net** file (because NevProp3 will be expect appended training data, not an external file).

Details: NevProp3 will look in the named file for headers (if specified– see 7.3.3) and data. The first column of data may be a row label (see 7.3.4). Subsequent values must be numeric, using integer, floating point, or exponential notation (e.g., the following are all read identically: 5, 5.000, and 5E0). Values may be separated by any number of spaces or tabs. A row may continue onto any number of lines– this is helpful when many predictors are used. NevProp3 knows the number of total variables to read in, and so will move across as many lines as necessary to complete each data vector.

Sample: **ReadTrainFile Iris.trn**
ReadTrainFile Iris (actual file name must be **Iris.trn**)

7.3.2 ReadTestFile (.tst)

Format: **ReadTestFile file_name[.tst]**

Values: **file_name** any string, with or without the **.tst** extension.

Default: (if **ReadTestFile** is not specified, no testing data will be used)

Note: The actual testing data file *must* have the extension **.tst**; if not provided, it is assumed. The testing file data must reside in the same directory as NevProp3 (or in the same directory as an alias linked to the executable NevProp3 kept in another directory).

Details: If declared, NevProp3 will look in the named file for testing data to be passed only once, at the end of an NevProp3 run, for the purpose of validating an established model. Summary statistics will be reported, and predictions saved, if requested (see 7.2.3). If an existing weights file is also read (see 7.2.7), it is not necessary that a model actually be trained during the current session– just specify zero for the number of epochs to be trained when calling NevProp3.

NevProp3 will look in the named file for headers (if specified— see 7.3.3) and data. The first column of data may be a row label (see 7.3.4). Subsequent values must be numeric, using integer, floating point, or exponential notation (e.g., the following are all read identically: 5, 5.000, and 5E0). Values may be separated by any number of spaces or tabs. A row may continue onto any number of lines— this is helpful when many predictors are used. NevProp3 knows the number of total variables to read in, and so will move across as many lines as necessary to complete each data vector.

Sample: **ReadTestFile Iris.tst**
ReadTestFile Iris (actual file name must be **Iris.tst**)

7.3.3 NHeaders

Format: **NHeaders argument**

Values: **argument** integer value = 0

Default: 0

Details: NevProp3 expects to find **NHeaders** header-lines before the start of actual rows containing data. This applies to data appended in the **.net** file, or to data contained in the **.trn** and **.tst** files). Headers have three uses: (1) for the user to provide strings of descriptive information about the dataset (which will be reported atop each NevProp3 run), and (2) for NevProp3 to parse in search of variable names for display purposes when either **DescribeVars** (see 7.8.1), **NEffectBoots** (see 7.8.5), or **kNN** (7.4.8) is invoked. When parsing to find variable names, NevProp3 starts with the last word (tokens separated by spaces, tabs, and/or newline characters) of the last header and works backwards until it has read in the number of words corresponding to the number of variables in use (determined by **NVars** as per 7.3.8). Therefore, the *final* header(s) should contain the variable column labels.

Sample: **NHeaders 0** (no lines of text precede first row of data)
NHeaders 3 (3 lines of text precede first row of data)

7.3.4 IDColumn

Format: **IDColumn argument**

Values: **argument** YES [Y, TRUE, T, 1]
argument NO [N, FALSE, F, 0]

Default: NO

Details: It is often important to keep track of the identity of the particular row of data. If **IDColumn** is YES, NevProp3 will retain the first column as the label for each row. The label may be any alphanumeric string. The label will be reproduced when prediction files are saved (7.2.2, 7.2.3).

Sample: **IDColumn YES** (first column of data is alphanumeric row label)
IDColumn NO (first column of numeric data for computation)

7.3.5 StandardizeInputs

Format: **StandardizeInputs argument**

Values: **argument** 0, 1, or 2

Default: 1

Note: Affects only training data. Changes only temporary NevProp3 array – original **.trn** file is not modified.

Details: The initial randomization of weights will not be effective if the training predictor variables are on different scales. If the raw predictors are already on the same scale, set **StandardizeInputs** to 0. Otherwise, setting **StandardizeInputs** to 1 will linearly transform the training data predictors to mean of zero and units of standard deviation. If standard deviations differ substantially among predictor variables (as is usually the case if binary predictors are included), it is preferable to set **StandardizeInputs** to 2, which linearly transforms the values of all predictor variables to lie in the range of -0.5 to +0.5; in this case, note that the mean will not be zero unless the mean of the raw predictor was centered between its minimum and maximum values. When weights are saved, they are inversely transformed to reflect the original scale of the variables; this allows validation predictions to be made using the weights and the **.tst** file.

Sample: **StandardizeInputs 0** (no modification of training data)
StandardizeInputs 1 (to mean zero, units of s.d.)
StandardizeInputs 2 (to range of -0.5 to +0.5)

7.3.6 ImputeMissing

Format: **ImputeMissing argument**

Values: **argument** 0 [DISCARD], 1 [MEDIAN], 2 [MEAN], or 3 [RANDOM]

Default: DISCARD

Note: Affects training and testing data.
Affects stored NevProp3 arrays– original files are not modified.

Details: Missing elements in the file must be indicated by either: (1) a period(**.**) separated from other values on the line with spaces or tabs, or, (2) the word **NA**. The appropriate choice depends on the mechanism for missingness, and the objective of the analysis (see 5.3.3).

Another approach is to use NevProp3's ANN-implementation of a k-nearest neighbor algorithm (see 7.4.8). If kNN is selected, it is necessary to initialize the values for missing elements by specifying **ImputeMissing** as either **MEDIAN**, **MEAN**, or **RANDOM**.

Sample: **ImputeMissing DISCARD**
(discard entire cases with one or more missing elements)
ImputeMissing MEDIAN
(replace missing elements with variable median)
ImputeMissing MEAN
(replace missing elements with variable mean)
ImputeMissing RANDOM
(replace missing with randomly-chosen nonmissing values)

7.3.7 ShuffleData

Format: **ShuffleData argument**
Values: **argument YES** [Y, TRUE, T, 1]
argument NO [N, FALSE, F, 0]
Default: YES
Note: Affects only training data.
Affects stored NevProp3 array– original **.trn** file is not modified.
Details: It is generally advisable to randomize the initial ordering of data, because internal training dataset splitting (7.6.1) is frequently used. On occasion, the user may want to pre-arrange the order of cases to be segregated during the splitting, and will want to set **ShuffleData** to NO.
Sample: **ShuffleData NO** (do not randomize the training data array)
ShuffleData 1 (randomize after reading in training data)

7.3.8 NVars

Format: **NVars argument**
Values: **argument integer value = 2**
Default: sum of **NInputs** + **NOutputs** settings
Details: **NVars** need only be specified when the user wishes to *subset* the variables in a training or testing dataset by specifying the settings **InputColumns** (see 7.3.9) and **OutputColumns** (see 7.3.10). **NVars** is the *total* number of data-containing variable columns in the training or testing file (excluding **IDColumn**). This number is needed for NevProp3 to parse the data file, since the actual sum of **NInputs** + **NOutputs** used may be less than the total number of data variables available.
Sample: **NVars 7**
(use only a total of 7 variables as predictors and dependent, variables, although more than 7 are present in the dataset)

7.3.9 InputColumns

Format: `InputColumns argument argument ...`

Values: `argument` integer value(positive, negative, or zero)

Default: 0 (means none declared – use first `Ninputs` columns as default)

Details: NevProp3 will use as inputs (predictor) variables only those in the positive-numbered (training and testing data file) columns listed after `InputColumns`. Alternatively, NevProp3 will use all data columns *except* those in the negatively-numbered columns listed after `InputColumns`. In both cases, counting of column numbers begins *after* the `IDColumn`, if present.

Column arguments may be separated by any combination of spaces or commas.

To use this setting at other than the default, `OutputColumns` (see below) and `NVars` (see 7.3.8) must be explicitly declared. For use under `kNN` mode, see 7.4.8.

Sample: `InputColumns 2 4 5`
(use data columns 2, 4, and 5 as predictors)
`InputColumns -1 -3`
(use data columns 2, 4, and 5 as predictors,
assuming 5 total columns of data)

7.3.10 OutputColumns

Format: `OutputColumns argument argument ...`

Values: `argument` integer value = 1

Default: 0 (means none declared– use last `Noutputs` columns as default, except in `kNN` mode, where virtual outputs are created corresponding to each input)

Details: NevProp3 will use as output (dependent) variables only those in the positively-numbered (training and testing data file) columns listed after `OutputColumns`. Alternatively, NevProp3 will use all data columns *except* those in the negatively-numbered columns listed after `OutputColumns`. In both cases, counting of column numbers begins *after* the `IDColumn`, if present.

Column arguments may be separated by any combination of spaces or commas.

To use this setting at other than the default, `InputColumns` (see above) and `NVars` (see 7.3.8) must be explicitly declared. For use under `kNN` mode, see 7.4.8.

Sample: `OutputColumns 1 3`
(use data columns 1 and 3 as the dependent variables)
`OutputColumns -2 -4 -5`
(use data columns 1 and 3 as predictors,
assuming 5 total columns of data)

7.3.11 SaveStandWts

Format: **SaveStandWts** *argument*

Values: **argument** YES [Y, TRUE, T, 1]
argument NO [N, FALSE, F, 0]

Default: NO

Note: Only affects scale of weight values saved to **.wts** file.

Details: If a **.wts** file will later be uploaded for prediction, the weights must be the scale of the original predictor variables; this is the reason for the default **SaveStandWts** NO. But if the weights will be saved only to inspected their relative magnitudes, setting **SaveStandWts** YES places the weights on a similar scale, *if* **Standardize** was set to 1 or 2 (see 7.3.5).

Sample: **SaveStandWts** NO (save weights on scale of original predictors)
SaveStandWts 1 (save weights on scale recoded by **Standardize**)

&7.4 CONFIGURATION Settings

7.4.1 Ninputs , Noutputs

Format: **Ninputs** *argument*
Noutputs *argument*

Values: **argument** integer value =1

Default: none– must be specified

Details: **Ninputs** specifies the number of input variables, and **Noutputs** the number of input variables, that NevProp3 should expect when parsing the data files, and creating the network architecture. When operating under **kNN** mode (i.e., when **kNN**>0; see 7.4.8), **Noutputs** should be set equal to the number of inputs (plus outputs, if desired) in the original file (this is because the target vector now consists of the same variables as the input, plus optionally the original output variables).

Sample: **Ninputs** 4 (four input variables)
Noutputs 1 (single output (dependent) variable)

7.4.2 Nhhidden

Format: **Nhhidden** *argument*

Values: **argument** integer value = 0

- Default: if not specified, one-half the number of inputs ($0.5 * N_{inputs}$, rounded to the nearest integer)
- Details: Specifying zero hidden units causes NevProp3 to create only direct connections from input to output units; that is, a generalized linear model. As (nonlinear) hidden units are added, nonlinearity in the effects is created with increasing flexibility. The default was chosen as a reasonable first representation in the reduction (50%) of dimensionality usually expected. Note that early stopping (see 7.6) prevents weights from being completely fit, reducing the final degrees of freedom (as if fewer completely-fit hidden units had been specified).
- Sample: **Nhidden 0** (generalized linear model)
Nhidden 3 (3 hidden units)

7.4.3 HiddenUnitType

- Format: **HiddenUnitType argument**
- Values: **argument** 0, 1, or 2
- Default: 1
- Note: All hidden units are created as the same type.
- Details: NevProp3 will create hidden units with an activation function determined by the argument:
- 0 linear
 - 1 symmetric logistic, range of -0.5 to +0.5
 - 2 asymmetric logistic, range of 0 to 1
- In general, it is preferable to use symmetric activations, because the initial weights are randomized about zero (see 7.4.7). Using linear activations creates a linear network, which will perform no better than omitting hidden units all together; this is because a linear combination of linear functions is itself a linear function. Asymmetric activations may be helpful if uploading weights created in another program that used this range.
- Sample: **HiddenUnitType 1** (symmetric activation, range of -0.5 to +0.5)
HiddenUnitType 2 (asymmetric activation, range of 0 to 1)

7.4.4 OutputUnitType

- Format: **OutputUnitType argument**
- Values: **argument** 0, 1, 2, or 3
- Default: 3
- Details: NevProp3 will create hidden units with an activation function determined by the argument:
- 0 linear (all output units)

- 1 symmetric logistic, range of -0.5 to +0.5 (all output units)
- 2 asymmetric logistic, range of 0 to 1 (all output units)
- 3 automatic (type determined by type of output variable—
also allows any values for dichotomous output variable)

In general, naturally bounded output (dependent) variables should be assigned a bounded activation range for proper scoring. Unbounded output variables should have a linear activation projecting to them (argument 0).

As a convenience, argument 3 causes NevProp3 to scan the dependent variable(s) to determine whether or not the variable is dichotomous. If it is 0,1 dichotomous (i.e., binary), an asymmetric logistic output unit is assigned (argument 2). For any other assigned pair of dichotomous values, NevProp3 internally reassigns the lower value to -0.5 and the upper value to 0.5, and used a symmetric logistic (argument 1). For nondichotomous output variabxles, linear activation (argument 0) is internally assigned. In the case of multiple output units, argument 3 (automatic) causes the appropriate type of activation function to be assigned to each unit. The actual internal assignments are shown in the display of settings at the start of the NevProp3 run.

Sample: **OutputUnitType** 3 (automatic assignment)
OutputUnitType 2 (all outputs must be 0,1 dichotomous)
OutputUnitType 1 (all outputs must be -.5,.5 dichotomous)
OutputUnitType 0 (outputs are assumed to be unbounded)

7.4.5 1ofN

Format: **1ofN** argument

Values: **argument** YES [Y, TRUE, T, 1]
argument NO [N, FALSE, F, 0]

Default: NO

Note: Applicable only when all output units are binary (dichotomous).

Details: When all output units are binary (with logistic activation appropriately set), the user may desire either (a) that all outputs sum to one (1-of-N classification), or (b) independent probability estimates for each output (M-of-N classification). Setting **1ofN** to YES invokes a normalizing function that forces outputs to sum to one during optimization, analogous to the Softmax concept.

Sample: **1ofN** NO (output variables' probabilities are independent)
1ofN 1 (output variables' probabilities sum to 1)

7.4.6 Connect

Format: **Connect** argumentF1 argumentF2 argumentT1 argumentT2

Values: **argumentxx** positive integer value

Note: Connections must always be made in the forward direction: input units must be connected to hidden or output units; hidden units must be connected to (a) output units, or (b) other hidden units positioned at least one connection link farther from the input unit. Any number of spaces or tabs may separate arguments, but the setting and all arguments must be contained in one line.

Default: (When no **Connect** is specified.) All inputs are connected fully to all hidden units as a single layer, and all hidden units are connected fully to all output units. If no hidden units are specified (i.e., **Nhidden 0**), a linear model is formed.

Details: F1 and F2 denote indexes of the beginning and end of the range From which connections are made (from each unit within that range), and, T1 and T2 denote indexes of the beginning and end of the range To which connections are made (to each unit within that range).

The index begins with 1 for the first input unit (zero is reserved for the automatically assigned bias unit), and ends with the sum (Ninputs + Nhidden + Noutputs) for the last output unit.

If a single unit is the origin (or terminus) of a connection, its index should be duplicated, so that there are always 4 arguments following each **Connect** call.

NevProp3 can connect any subset of inputs to any subset of hidden units, and directly to any subset of outputs (so called "shortcut," or skip connections). Likewise, any subset of hidden units can be connected to other hidden units or outputs (provided that the connections are always in a forward direction). In this way, arbitrary layering and transformations can be designed.

Any number of **Connect** calls may be specified in a **.net** file.

Sample: **Connect 1 4 5 5**
(GENERALIZED LINEAR MODEL: If Ninputs is 4, Nhidden 0, and Noutputs 1, this corresponds to a generalized linear model.)

Connect 1 4 5 7
Connect 5 7 8 8

(SINGLE FULLY-INTERCONNECTED HIDDEN LAYER: If Ninputs is 4, Nhidden 3, and Noutputs 1, this corresponds to first connecting each of units 1, 2, 3 & 4 [the inputs] to each of units 5, 6, & 7 [the hidden units], and then connecting these in turn to unit 8 [the output unit]. This would be also be the default configuration for this network if no **Connect** calls were specified.)

Connect 1 4 5 7
Connect 5 7 8 8
Connect 1 4 8 8

(SHORTCUTS: Same as in the previous sample, but "shortcut" links been established from each input to all outputs; this

sometimes speeds the training of a model with minimal nonlinearity in the true effects.)

```
Connect 1 4      5 7
Connect 5 7      8 9
Connect 8 9     10 10
```

(MULTIPLE LAYERS: If Ninputs is 4, Nhidden 5, and Noutputs 1, this corresponds to connecting each of units 1, 2, 3 & 4 [the inputs] to each of hidden units 5, 6, & 7, and then connecting these in turn to unit hidden units 8 & 9, and these in turn to the output unit, 10.)

```
Connect 1 1      5 7
```

(ISOLATED NONLINEAR TRANSFORMATION: Connects the first input unit to 3 hidden units, which will accomplish a nonlinear transformation of the variable without introducing any interactions with the effects of other input variables. This is analogous to spline regression with 3 knots. Units 5-7 would require additional **Connect** calls to specify their connectivities to other hidden or output units.)

7.4.7 WeightRange

Format: **WeightRange** *argument*

Values: **argument** numeric value = 0.0

Default: 0.001

Details: If existing weights are not read in from a **.wts** file (see 7.2.7), NevProp3 will assume they are to be initialized by selection from a uniform random range between \pm **WeightRange**. The **WeightRange** should be small relative to the scale of the input variables to prevent the early fitting of random effects.

Sample: **WeightRange .1** (weights initialized between \pm 0.1)
WeightRange 1E-2 (weights initialized between \pm 0.01)

7.4.8 kNN

Format: **kNN** *argument*

Values: **argument** integer value = 0

Default: 0 (or not specified) If **kNN** is >0, NevProp3 operates in kNN mode.

Note: This is a parametrized version of the k-nearest neighbor algorithm, which may require many more hidden than input units (up to a maximum of the number of cases, if there is no patterning of the cases in data-space).

Details: kNN is a specialized mode of operation. NevProp3 first identifies, for each case in the data set, its k (= the value of **kNN**) nearest neighbors (by Euclidean distance). Next, these k neighbors become output

targets for the corresponding inputs. Thus, NevProp3 trains a kNN-conditional model that approximates the unconditional distribution of data. If specified using **OutputColumns**, dependent variable(s) is(are) predicted simultaneously.

The original data cases are replicated to create k copies, each assigned to one of the k nearest neighbors as targets. For example, consider an original data set containing 100 cases. If **kNN** is 3, the 3 nearest neighboring cases are assigned to an identical copy of each reference case, generating a total of 300 cases to train the model. If there were originally 4 input and 1 output (dependent) variables, the internally created kNN model would have 4 inputs, and 4 or 5 outputs (4 corresponding to the input variables, plus, optionally, the original output variable).

By default, all input variables will be used, unless a subset is specified using the **InputColumns** setting. NevProp3 creates virtual output columns for these targets (which are actually the same variables as specified by **InputColumns**), so no **OutputColumns** need be specified (optionally, **OutputColumns** may be set to 0). However, if the user desires to simultaneously predict a dependent target variable which is not itself a predictor, that variable's index should be specified in the **OutputColumns** setting.

If input variables contain missing elements, these elements are initialized according to the **ImputeMissing** setting (see 7.3.6), and updated during training by a neural network version of Expectation-Minimization. If **SaveTrainImpFile** was specified in the **.net** file, an imputed train file (**.itr**) is saved— see 6.5 and 7.2.4.

Optionally, the model may be applied to new data. Either immediately after the model is fit, or later by uploading previously saved weights, specified test (**.tst**) data containing missing values can be imputed and saved as an imputed test file (**.its**)— see 6.5 and 7.2.5. A dependent variable may simultaneously be predicted (whether or not the **.tst** file contains input variables with missing elements).

Sample: **kNN 0** (kNN mode disabled)
kNN 3 (enter kNN mode, using 3 nearest neighbors)

7.4.9 Expert

Format: **Expert argument.ptr argument.pts argumentID**

Values: **argument.ptr** name of a file with predictions, for training
argument.pts name of a file with predictions, for validation
argumentID arbitrary string naming the expert, for reporting

Note: Any number of **Expert** calls may be specified in a **.net** file.

Any number of spaces or tabs may separate arguments, but the setting and all arguments must be contained in one line.

In the present version, only a single output variable (prediction) is allowed. The prediction may lie on any scale (of course, all experts should be using same scale for their predictions).

Default: None. If an **Expert** call is made, NevProp3 operates in ANN-Gated Ensemble mode.

Details: Ensemble is a specialized mode of NevProp3 operation. Each **Expert** call describes the sources of data, and the name of the expert. In Ensemble mode, NevProp3 accesses the original data (**.trn** file), and the files declared for each **Expert**. The objective is to find an optimal weighting (or, “gating”) for each **Expert** for each case (rather than a fixed weight), under the assumption that experts differ in prediction accuracy in different domains of data-space.

Sample:

Expert	SmithTrn.ptr	SmithTst.pts	Dr_Smith
Expert	GLMTrn.ptr	GLMTst.pts	GLM
Expert	ANNTTrn.ptr	ANNTst.pts	ANN
Expert	TREETrn.ptr	TREETst.pts	TREE

(A model to weight the predictions four “experts” will be generated; the **.ptr** files will be used to train the NevProp3 gating model, and the **.tst** files will be used once to validate the model. Here, Dr_Smith may be a human expert’s subjective score assignment, GLM the predictions of a generalized linear model, ANN those of an artificial neural network model, and TREE those of a decision tree algorithm [all of which have been determined earlier]. Predictions in the **.ptr** and **.pts** files of the four experts must be derived from identical data [which needs to be provided to NevProp3 via the **ReadTrainFile** setting, see 7.3.1]).

7.4.10 EnsemblePrior

Format: **EnsemblePrior** argument

Values: argument 0, 1, or 2

- 0 No internal “prior probability expert”
- 1 External & internal experts; weights must sum to 1
- 2 External & internal experts; only external weights must sum to 1

Default: 0

Note: Meaningful only when operating in ANN-Gated Ensemble mode.

The sum of all external experts’ weights always equals 1 (because Ensemble is a probabilistic model), each weight lying in the range 0-1. The weight assigned to the prior may also be forced to be probabilistic (**EnsemblePrior** 1), or be allowed to float (**EnsemblePrior** 2) to correct for conversely biased but reliable external experts.

Details: When operating ANN-Gated Ensemble mode (see 6.6 and 7.4.8), setting **EnsemblePrior** 1 or 2 will cause NevProp3 to create an internal expert. This “expert” prediction is a constant value equal to

the prior probability of the outcome (computed internally using the output variable). With **EnsemblePrior 1**, the weights of the external experts can be interpreted relative to the importance of the weight of the internal prior probability. That is, only those experts contributing more information than the prior itself will have weights significantly greater than zero. (The magnitude of the external experts' weights may still be compared with one another.) With **EnsemblePrior 2**, only the comparison of weights among external experts is meaningful, because the prior now acts only as a bias correction.

Sample: **EnsemblePrior 0** (use only external experts in ensemble)
EnsemblePrior 1 (internal expert predicting prior)

&7.5 TRAINING (optimization) Settings

7.5.1 TrainCriterion

Format: **TrainCriterion argument**

Values: **argument 0, 1, 2, or 3**

- 0 mean square error (MSE) for all outputs
- 1 MSE, modified by hyperbolic arctan, for all outputs
- 2 log likelihood (cross entropy) for all outputs
- 3 automatic

Default: 3

Details: The training criterion function is also known as the error, residual, loss, or objective function. The use of the MSE criterion (**TrainCriterion 0**) assumes a roughly Gaussian distribution of residual error. This is commonly assumed for linear (unconstrained) output variables, for which the **OutputUnitType** should also be set to 0 (see 7.7.4). This assumption is not ideal for dichotomous (binomial) outputs, so the full maximum (log) likelihood (**TrainCriterion 2**) is preferred. An older alternative to log likelihood was an ad hoc transformation of the error using a hyperbolic arctangent (**TrainCriterion 2**), which is available but rarely useful.

It is generally recommended to leave **TrainCriterion 3**, which has the following effects:

- (a) If all output (dependent) variables are dichotomous, cross entropy criterion is used.
- (b) If any of the output (dependent) variables are *not* dichotomous, MSE is used as a common scale.

Sample: **TrainCriterion 1** (MSE criterion, for unbounded outputs)
TrainCriterion 3 (automatic assignment)

7.5.2 OptimizeMethod

Format: **OptimizeMethod** *argument*

Values: **argument** 0, 1, 2, or 3

- 0 pure gradient descent, constant **LearnRate**
- 1 pure gradient descent, globally adaptive **LearnRate**
- 2 **LearnRate** adapts locally for each individual weight
- 3 quickprop (variation on a second-order method)

Default: 1

Note: All methods are initialized at the value of **LearnRate**.

Quickprop's first 2 steps use method 0 to establish changes in gradient, and thereafter only use **LearnRate** setting when defaulting to gradient descent (see 7.5.5).

Details: It is generally not advisable to use a constant **LearnRate** (method 0), because the objective function will not smoothly converge at a local minimum if forced to continue jumping. Globally adaptive **LearnRate** (method 1) is reliable, although slow; with each weight update, the global learn rate is incremented or decremented a small amount (set in the code file `epsilon.c`) depending on whether the previous step resulted in an improvement or worsening of the criterion function. Locally adaptive **LearnRate** (method 2) increments a weight's **LearnRate** only if the global criterion decreased *and* the partial gradient in that weight's dimension remained negative (that is, method 2 does use information about local curvature, not only first-order gradient). Quickprop (developed by Scott Fahlman <sef+@cs.cmu.edu>) makes an even stronger assumption about curvature: for each weight's dimension, the curvature is considered to be locally parabolic (quadratic), so each change in the gradient and weight uniquely determine the bottom of the current parabola. Quickprop tries to jump towards that minimum (see 7.5.5, 7.5.6, for control of that jump). This differs from Raphson-Newton and quasi-Newton methods (which also assume quadratic curvature) in that each weight dimension is considered independently, as if the Hessian matrix were diagonalized, thereby eliminating the computation effort needed for matrix inversion.

Sample: **OptimizeMethod** 1 (globally adaptive gradient descent)
OptimizeMethod 3 (quickprop)

7.5.3 LearnRate

Format: **LearnRate** *argument*

Values: **argument** numeric value > 0.0

Default: 0.01

- Details: During optimization, the weights are changed in proportion to the gradient of error criterion; the factor of proportionality is the **LearnRate**. See 7.5.2 and 7.5.5 for discussions of how the **LearnRate** is utilized under the several options for optimization. The default given is usually adequately slow for starting any optimization method. However, if early unstable oscillations in criterion value are noted on the NevProp3 display, consider reducing the **LearnRate** by sequential orders of magnitude.
- Sample: **LearnRate 0.001** (with each update, make weight magnitude changes 10 times slower than under the default setting)

7.5.4 SplitLearnRate

- Format: **SplitLearnRate argument**
- Values: **argument YES** [Y, TRUE, T, 1]
argument NO [N, FALSE, F, 0]
- Default: NO
- Details: If **SplitLearnRate** is YES, the **LearnRate** (see 7.5.3) for each weight is divided by the number of weights feeding forward in parallel into the distal (hidden or output) unit. The effect is to slow down movement over the criterion surface in that weight's dimension. **SplitLearnRate** causes the slowing to be proportional to the potential degrees of freedom affecting the input to the unit. This setting is relatively ineffectual with **OptimizeMethod** methods 2 and 3 (see 7.5.2), in which the learning rate varies with each weight.
- Sample: **SplitLearnRate NO** (accept **LearnRate** for all weights)
SplitLearnRate 1 (divide **LearnRate** by number of parallel weights)

7.5.5 QPModeSwitchThreshold

- Format: **QPModeSwitchThreshold argument**
- Values: **argument numeric value > 0.0**
- Default: 0.0
- Note: Active only when optimizing by quickprop (**OptimizeMethod 3**).
- Details: The parabolic curvature assumption of quickprop will be imprecise if the criterion-surface above weight-space is relatively flat. Therefore, if the absolute change in a weight at the last update was less than or equal to the value of **QPModeSwitchThreshold**, optimization will revert to gradient descent (with a learning rate determined by the **LearnRate** [7.5.3]). The percent of weights that were updated by gradient descent is indicated at each reporting interval in the NevProp3 display to standard output.

Sample: `QPModeswitchThreshold 0.1` (use quickprop as long as weight magnitude changed by at least 0.1)

7.5.4 QPMaxFactor

Format: `QPMaxFactor argument`

Values: `argument` numeric value > 0.0

Default: 1.75

Note: Active only when optimizing by quickprop (`OptimizeMethod 3`).

Details: `QPMaxFactor` is the maximum factor by which a weight's magnitude will be allowed to change relative to the previous change in weights (see 7.5.3). In our experience, the default appears relatively robust. Serves as a safeguard against wild jumps induced by strong unidimensional parabolic curvature.

Sample: `QPMaxFactor 1.75` (at any weight update, maximum jump in the magnitude of the weight is factor of 1.75 times last jump)

7.5.7 Stochastic

Format: `Stochastic argument`

Values: `argument` YES [Y, TRUE, T, 1]
`argument` NO [N, FALSE, F, 0]

Default: NO

Note: Automatically disabled in quickprop mode (not found to be stable).

Details: If `Stochastic` is YES, weights are updated after each case is presented, and the entire dataset is randomly permuted after each full pass. `Stochastic` contributes a random walk component to the search for a minimum of the criterion function, widening the exploration of the surface, and possibly jumping out of small local minima.

Sample: `Stochastic NO` (update weights after full pass through data)
`Stochastic 1` (update after each case, permute after full pass)

7.5.8 SigmoidPrimeOffset

Format: `SigmoidPrimeOffset argument`

Values: `argument` numeric value > 0.0

Default: 0.0

Details: `SigmoidPrimeOffset` is added to the activation function derivative, as a heuristic to maintain some backpropagation of error signals even when a logistic activation function is saturated along

one of its tails (where its derivative would otherwise be near zero). Use of a **SigmoidPrimeOffset** may speed up learning in some problems, but may also induce instabilities, because it perturbs the gradient descent. Consider its use if, despite smaller **LearnRate** and greater **WeightDecay**, weights rapid grow exponentially (indicating that they may be trying to overcome the influence of logistics locked-up prematurely along their tails).

Sample: **SigmoidPrimeOffset 0.1** (at any weight update, maximum jump in the magnitude of the weight is factor of 1.75 times last jump)

7.5.9 Momentum

Format: **Momentum argument**

Values: **argument** numeric value = 0.0

Default: 0.01

Note: **Momentum** is not used in quickprop (**OptimizeMethod 3**), except when it defaults to gradient descent.

Details: **Momentum** is the fraction of the previous change in a weight that is added to the next change (in addition to the amount dictated by backpropagation). The effect is to exponentially smooth, or dampen, update-to-update oscillations along the error criterion surface. **Momentum** is, therefore, an indirect form of regularization; it may stabilize optimization while larger learning rates are employed. Typical range of useful values is 0.01 to 0.99; a value 0.1 to 1 times the **LearnRate** is often a stable choice.

Sample: **Momentum 0.1** (to each weight's update, add 0.1 times the magnitude of its last weight change)

7.5.10 WeightDecay

Format: **WeightDecay argument**

Values: **argument** negative numeric value = 0.0

Default: -0.001

Note: **Momentum** is not used in quickprop (**OptimizeMethod 3**), except when it defaults to gradient descent.

Details: **WeightDecay** is the fixed fraction of the weight's magnitude subtracted at the next weight update. From a frequentist statistical viewpoint, this is a heuristic to prevent weight magnitude from becoming excessive unless the data reinforces the growth. From a Bayesian viewpoint, it expresses the belief that the distribution of weights should be Gaussian, centered about zero. In any event, **WeightDecay** induces a regularization penalty. That is, by punishing weight growth, the final multivariate regression function has less hyper-curvature (hence lower variance). Typical range of

useful values is -0.001 to -0.9; a stable selection is often one-tenth of the **LearnRate**.

An adaptive (but computationally intensive) penalty is available using ARD (see 7.7).

Sample: **WeightDecay -0.01** (subtract 0.1 times the magnitude of each weight from its next update)

7.5.11 BiasPenalty

Format: **BiasPenalty argument**

Values: **argument** YES [Y, TRUE, T, 1]
argument NO [N, FALSE, F, 0]

Default: NO

Details: Weights multiplying the bias constant feed into all hidden and output units, to permit optimal rotation of weight-dimension axes during optimization. If you have some reason to believe that the regression surface should pass through the weight-space origin (that is, the distribution of these effects should center about zero), **BiasPenalty** should be changed to YES.

Sample: **BiasPenalty NO** (bias weights are not penalized)
BiasPenalty 1 (bias weights are penalized— regression surface is expected to pass through the weight-space origin)

&7.6 BEST-BY-HOLDOUT Settings

7.6.1 NHoldout & PercentHoldout

Format: **PercentHoldout argument**

Values: **argument** numeric value = 0.0

Default: 50 if AutoTrain YES; 0 if AutoTrain NO
- or -

Format: **NHoldout argument**

Values: **argument** integer value = 0

Default: not defined (depends on number of training cases)

Note: Only one of these settings should be defined in a given **.net** file.

Details: **PercentHoldout**, or, alternatively, **NHoldout**, specifies how many (if any) training cases should be segregated into a subset. Cases to subset are drawn from the bottom of the training dataset.

This subset will not be used for direct optimization, but rather to assess the predictive accuracy of the model during the process of optimization. A character-based graphic display reports the pattern of error on the subset. The value of the best subset-determined error criterion-per-case is used subsequently if **AutoTrain** is YES (see 7.6.4).

Sample: **PercentHoldout** 30 (30% of cases, drawn from the bottom of the training data set, are held-out as a subset to assess model performance during optimization with the remaining cases)
PercentHoldout 0 (no cases are held out)

7.6.2 MinEpochs

Format: **MinEpochs** argument

Values: argument integer value = 1

Default: 200

Details: NevProp3 optimization will proceed *at least* **MinEpochs** epochs (iterations) through the training dataset before stopping (up to the maximum number of iterations specified on initiating the program). **MinEpochs** is primarily useful when **PercentHoldout** or **NHoldout** is invoked, because NevProp3 is then keeping track of the 10 epochs with the lowest error on the held back training subset—during early epochs of training, oscillations usually occur that could appear as a local minimum. Use of a reasonable **MinEpochs** (usually 50-200) will force NevProp3 to continue optimization and forward checking for a true functional error.

Sample: **MinEpochs** 50 (optimize at least 50 epochs before early stopping as determined by training subset error minimum)

7.6.3 BeyondBestEpoch

Format: **BeyondBestEpoch** argument

Values: argument numeric value = 0.0

Default: 1.5

Details: If early stopping is invoked (by a positive **PercentHoldout** or **NHoldout**), NevProp3 will continue *at least* a factor of **BeyondBestEpoch** times the best epoch found so far (as determined by the error on the held back training subset). This is to help ensure that NevProp3 optimizes a sufficient number of epochs to be reasonably certain the minimum found was true functional error minimum for that model. **BeyondBestEpoch** results in a dynamic target; that is, if a new “best” error is found while searching beyond the old best, the target for stopping the search is updated to reflect the **BeyondBestEpoch** factor times the new best error. Typical

range is 1.5 to 2, assuming a reasonable **MinEpochs** was specified to avoid early oscillations in the error criterion (see 7.6.2).

Sample: **BeyondBestEpoch 2.0** (optimize forward to the epoch number that is twice that of the current best, as determined by training subset error minimum)

7.6.4 AutoTrain

Format: **AutoTrain** *argument*

Values: **argument** YES [Y, TRUE, T, 1]
argument NO [N, FALSE, F, 0]

Default: YES

Note: Available only if **PercentHoldout** or **NHoldout** is specified.

Details: Early stopping, if invoked by a positive **PercentHoldout** or **NHoldout**, will result in the discovery of the best set of weights determined by minimum error criterion on the training subset. Saving or using these weights for prediction is suboptimal, because the predictive information in the held out subset did not directly contribute to the model optimization. By setting **AutoTrain** to YES, the weights above are not used for the final model. Rather, NevProp3 discovers the mean error criterion per case on the training optimization subset at the epoch at which the held out subset error was best (phase I). In phase II, training is restarted (from the same initial weights) using *all* training cases— optimization now stops when the error criterion reaches or exceeds the mean error criterion per case discovered in phase I.

Sample: **AutoTrain NO** (for early stopping, do not recombine training data subsets for second phase of optimization)
AutoTrain 1 (perform second phase of optimization using all training cases, stopping at best error discovered in phase I)

7.6.5 NSplits

Format: **NSplits** *argument*

Values: **argument** integer value = 1

Default: 5

Details: When **AutoTrain** is active, the entire training set is used for optimization in phase II (see 7.6.4), stopping when the error criterion reaches the mean error criterion per case discovered in phase I cross-validation. To lessen the sampling variation that results from a single cross-validation split, perform multiple (**NSplits**) cross-validations to obtain an average error-per-case target for phase II. Typical values are 1-10 (less than 5 is dubious; greater than 10 does not seem to improved average).

Sample: **NSplits 10** (repeat phase I of Autotrain 10 times, each time using a different random split of the data; average the 10 error-per-case values as the target for phase II on all the data)

7.6.6 SepBootXVal

Format: **SepBootXVal** *argument*

Values: **argument** YES [Y, TRUE, T, 1]
argument NO [N, FALSE, F, 0]

Default: YES

Note: Meaningful only when **NBoots** is > 0 and **AutoTrain** is active.

Details: During sampling with replacement to construct booted datasets, it is possible for replicates of cases to fall into (what becomes) both subsets of the training data in phase I of **AutoTrain** (see 7.6.4). This will lead to overfitting, because optimization will proceed further in the presence of exact replicates in the held out subset. This phenomenon is prevented by specifying that **SepBootXVal** YES, forcing all replicates of a given case to be separated (segregated) into one or the other subset.

Sample: **SepBootXVal NO** (allow replicates across subsets in phase I)
SepBootXVal 1 (segregate case replicates in phase I)

&7.7 AUTOMATIC RELEVANCE DETERMINATION Settings

7.7.1 UseARD

Format: **UseARD** *argument*

Values: **argument** NO [0], QD [1], or FULL [2], or LastEpoch [3]

Default: NO

Note: FULL and LastEpoch ARD are computationally intense.

Details: **UseARD** QD or FULL causes NevProp3 to use ARD (proposed by David MacKay, see 6.3, 6.4), a dynamic optimization penalty to regularize the model. Thus, **UseARD** is a NevProp3 *hyperparameter*. Both QD and FULL versions normalize the penalty according to the magnitude of weights in a group (see 7.7.5). Additionally, the FULL version estimates and inverts a Hessian matrix every **ARDFreq** epochs (see 7.7.3) to estimate the effective degrees of freedom (number of well-determined parameters at that epoch).

UseARD LastEpoch is appropriate when regularization was *not* performed using ARD, but it is desired to compute, for the final model, summary estimates of variable relevance and number of well-determined parameters (effective degrees of freedom used by the model). This requires iterative computation of the hyperpenalty,

holding the weights constant. Iteration stops when the fractional change between successive hyperpenalties is less than **ARDTolerance** (7.7.3). To prevent infinite looping, the maximum number of iterations is set at 100*maxepochs specified for the optimization as a whole.

Sample: **UseARD QD** (use quick-and-dirty dynamic penalty)
UseARD FULL (use full matrix inversion to define penalty)
UseARD LastEpoch (at end of non-ARD regularization, compute variable relevances and number of well-determined parameters)

7.7.2 WhenARD

Format: **WhenARD argument**
Values: **argument** integer = 1, or AUTO
Default: AUTO
Note: Applicable only with **UseARD QD** or **FULL**.
Details: QD or FULL ARD will not commence until **WhenARD** epochs of training have been completed. AUTO means the program will use a heuristic algorithm to determine when to start ARD (see 7.7.3).
Sample: **WhenARD 100** (initiate ARD after epoch 100)
WhenARD AUTO (initiate ARD as determined by NevProp3)

7.7.3 ARDTolerance

Format: **ARDTolerance argument**
Values: **argument** numeric value = 0.0
Default: 0.05
Note: Applicable with either:
WhenARD QD or **FULL**, and **WhenARD AUTO**; or
WhenARD LastEpoch.
Details: With **WhenARD QD** or **FULL** and **WhenARD AUTO**, NevProp3 tracks the epoch-to-epoch change of all weights from the start of optimization. If, for each of 5 such consecutive changes in at least 50% of the cases, the increase in all weights is at least a factor of **ARDTolerance**, ARD (QD or FULL) will be initiated. The purpose of this check is to allow NevProp3 optimization to commit to a projection in weight space before attempting any (weight pattern-sensitive) dynamic regularization.
With **WhenARD LastEpoch**, ARD computation of the variable relevances and the number of well-determined parameters commences at the last epoch of optimization. Iteration stops when the fractional change between successive hyperpenalties is less than

ARDTolerance. Maximum no. of iterations allowed (to prevent infinite loops) is `MAX_ITERATION_FACTOR*maxepochs` (`MAX_ITERATION_FACTOR` is #defined in the code file `netTr.c`; presently defined at 1).

Sample: **ARDTolerance 0.10** (with **UseARD** QD or FULL, initiate ARD regularization when all weights consistently increased by at least 10% for 5 consecutive epochs; -or- when **UseARD** LastEpoch, stop iteration of the hyperpenalty when consecutive estimates are within 10%)

7.7.4 ARDFreq

Format: **ARDFreq** *argument*

Values: **argument** integer = 1, or AUTO

Default: AUTO

Note: Applicable only with **UseARD** FULL.

Details: FULL ARD will reestimate and invert its Hessian matrix every **ARDFreq** epochs of training. AUTO means the program will use the current NevProp3 display reporting frequency as the **ARDFreq**.

Sample: **ARDFreq 25** (reestimate Hessian every 25 epochs)
ARDFreq AUTO (reestimate Hessian at the report frequency)

7.7.5 GroupSelection

Format: **GroupSelection** *argument*

Values: **argument** INPUT or OUTPUT

Default: INPUT

Note: Applicable only with **UseARD** QD or FULL.

Details: **GroupSelection** determines whether direct (skip) connections from input units to output units are assigned to either the input group (INPUT) or output group (OUTPUT) for ARD penalties.

Sample: **GroupSelection INPUT** (skip connections assigned to corresponding input units' groups)
GroupSelection OUTPUT (skip connections assigned to corresponding output units' groups)

7.7.6 ARDFactor

Format: **ARDFactor** *argument*

Values: **argument** numeric value = 0 (slightly greater or less than 1.0)

Default: 1.0

Note: Applicable only with **UseARD** QD or FULL. Very large **ARDFactor** (= 2) will disrupt model fitting, because the resulting large penalties throw gradients in the direction opposite that of the previous update. Very small **ARDFactor** (close to zero) will nullify the penalty-based regularization.

Details: At each **ARDFreq** epoch, multiplies ARD-determined gradient penalties by **ARDFactor** . Could be used to correct the ARD estimates of Hessian-based penalties, for systematic bias induced by non-Gaussian curvature of error surfaces.

Sample: **ARDFactor 1.1** (increases the effective ARD-determined gradient penalty by 10% over Hessian-based estimate)
ARDFactor .95 (lessens the effective ARD-determined gradient penalty by 5% below Hessian-based estimate)

7.7.7 BiasRelevance

Format: **BiasRelevance** argument

Values: argument YES [Y, TRUE, T, 1]
argument NO [N, FALSE, F, 0]

Default: NO

Note: Applicable only with **UseARD** QD or FULL.

Details: Setting **BiasRelevance** YES causes NevProp3 to use include bias weights in the computation of the relative importance, or *relevance*, of input groups (see 6.4). The resulting summary then displays only relative importance of the predictor variables, including the bias (similar to the effect of predicting the mean for all cases). Because ARD relevance is an exploratory tool, the default is NO, to facilitate better relative comparison of the predictors, without respect to the bias.

Sample: **BiasRelevance YES** (include bias weights in the computations of the relative importance -- relevance -- of input groups)

&7.8 REPORTING Settings

7.8.1 DescribeVars

Format: **DescribeVars** argument

Values: argument YES [Y, TRUE, T, 1]
argument NO [N, FALSE, F, 0]

Default: YES

Details: Setting **DescribeVars** YES causes NevProp3 to compute and display, prior to the start of any training, the following descriptive

statistics on all input and output variables: number of cases, number of missing elements, mean, median, standard deviation, highest and lowest five values; frequency table for dichotomous and multilevel variables; and percentile table for continuous variables (NevProp3 scans each variable, and considers it continuous if more than 10 discrete values are present).

Descriptive statistics are useful for (1) ensuring the integrity of the dataset if transferred from another application or storage medium, (2) exploring new datasets for missingness, miscoding, and outlying data, (3) understanding univariate centers and distributions, (4) confirming that the proper variables were subsetted when using the **InputColumns** and **OutputColumns** settings, and (5) referring to the coding of data when interpreting the effects of predictors (see **NEffectBoots**, 7.8.5). After the first run on a given dataset, the user will probably want to set **DescribeVars** NO.

If a dataset header is present, NevProp3 will attempt to read in variable column labels, beginning with the end of the last header and moving backwards. That is, the user should place variable column labels in the last header(s). In the absence of any headers, NevProp3 will provide a sequentially numbered label for the variables.

Missing data is recognized by the presence of either ' .' (an isolated period), or the string 'NA' (case insensitive). These conventions are compatible with the output of most statistical packages.

Sample: **DescribeVars** NO (do not show descriptive statistics)

7.8.2 CalccIndex

Format: **CalccIndex** argument

Values: **argument** YES [Y, TRUE, T, 1]
argument NO [N, FALSE, F, 0]

Default: YES

Note: Applicable only for dichotomous output variables. If any output variable is not dichotomous, CalccIndex will automatically be reset to NO.

Details: **CalccIndex** YES causes NevProp3 to compute, for all dichotomous output variables, a nonparametric measure of discrimination called the c index, or area under the receiver operating characteristic curve. This summary statistic is 0.5 when the predicted probability for the cases is random with respect to their target outputs, and rises to a maximum of 1.0 if all cases with a one of the target values (e.g., 0) are predicted to have lower probabilities than all cases with a the other target (e.g., 1). If the c index is not of interest to the investigator, setting **CalccIndex** NO will save some computational overhead.

There is an analogous c index for non-dichotomous variables (although it's interpretation is not so clear), but we haven't got around to implementing it – sorry.

Sample: **CalccIndex NO** (do not show descriptive statistics)

7.8.3 **ScoreThreshold**

Format: **ScoreThreshold argument**

Values: **argument** numeric value = 0.0

Default: 0.5

Note: The computation of **ScoreThreshold** has been changed from that in NevProp2 for non-dichotomous outputs (no difference in results for binary outputs).

Details: **ScoreThreshold** is the portion of an output variable's range (maximum - minimum value) within which a prediction must fall to be classified as a correct. As optimization proceeds, predictions get closer to their targets, so that the fraction of training case predictions crossing the **ScoreThreshold** should continually improve. The NevProp3 display reports this conversely, as the average thresholded *error*; that is, the average, across all output variables, of their fractions of cases *not* within the **ScoreThreshold***range window of the targets. So the average thresholded error should continually decrease on training data.

For a binary output variable, the range is one, so that the **ScoreThreshold** itself is the range within which a prediction must fall to be classified as correct.

Sample: **ScoreThreshold 0.1** (classify as within-range of the target, any prediction within 0.1*range of that output variable)

7.8.4 **NBoots**

Format: **NBoots argument**

Values: **argument** integer value > 0

Default: 0

Details: When **NBoots** is > 0, NevProp3 will first train the full model as determined by the other settings. **NBoots** bootstrapped data sets (with the same total number of cases) will then be constructed, by sampling with replacement from the original training dataset. This results in "booted" datasets with some replicate cases, and some unrepresented cases. The entire training procedure used for the original data (including the same randomly initiated weights) is repeated for each booted data set. The resultant booted models are used to adjust the optimistically-biased predictive statistics of the full model (see chapter 5). The basic idea is that the booted data sets are fractally analogous to the original data, in the same way the original data set is a sample of the larger universe of such samples. Therefore, the relative deterioration in predictive goodness, from original to

booted model, should be similar to the deterioration found when the full model is applied to future samples. This adjustment procedure is intended to eliminate the need to saving part of the available data as a test set for final model validation.

Sample: **NBoots 50** (repeat entire training process on 50 bootstrapped datasets, and use the booted models to adjust predictive statistics of the full model)

7.8.5 NEffectBoots

Format: **NEffectBoots argument**

Values: **argument** integer value > 0

Default: 0

Details: It is often desirable to understand how individual predictors contribute to the predictions of a model— that is, to infer causes from effects. The “effect” of a predictor is the derivative of the output of a model with respect to an isolated change in the predictor. Unlike linear regression, where all other predictive variables may be held constant, the effect of a given variable is highly dependent upon the values of the other predictors due to the dense interactions in the neural network model. Therefore, in NevProp3, an effect is computed for *each* case, then averaged, to yield the “mean effect.” The standard deviation of effects among the cases is a measure of the nonlinearity of this effect. See chapters 2 and 6 for details.

If **NEffectBoots** is > 0, NevProp3 will first train the model as determined by the other settings. Second, the mean effects module is invoked, which computes, for each output, the “mean effect” and mean prediction for each variable. These statistics are shown for the variable as a whole, and for each unique level (for continuous variables, the data is broken into quartiles for this purpose). The percentile distribution of the variable’s mean effect is displayed, in order to convey a sense of the distribution of nonlinearity in the effect (the standard deviation of this distribution is also displayed as the “nonlinearity score”). Third, **NEffectBoots** booted data sets are generated from the original training set. Mean effects are computed for the predictors in each boot, which are used to generate a 95% confidence interval of the mean effects obtained from the full model. If **NEffectBoots** is < 200, the mean effects are used to calculate a standard deviation, which is used to derive a confidence interval. If **NEffectBoots** is = 200, the actual distribution of mean effects is used to form a confidence interval. The confidence intervals reflect variability expected in future samples, contributed by variability in composition of the data.

If **NBoots** is > 0, the procedure described above is applied to each of the **NBoots** bootstrapped datasets and models created earlier for adjusting optimistic bias of the summary statistics (see 7.8.4). (That is, for each of the **NBoots** bootstrapped data sets, **NEffectBoots**

are generated. This is computationally demanding.) The final confidence intervals computed, using all such effects, reflect variability expected in future samples, contributed by both data composition *and* model fitting.

Sample: **NEffectBoots 100** (compute mean effects for each predictor, with confidence limits)

7.8.6 OutputStatVars

Format: **OutputStatVars argument argument ...**

Values: **argument** integer value (positive, negative, or zero)

Default: 0 (include all output variables)

Details: When multiple output (dependent) variables are used, the summary statistics reflect the performance averaged across all such outputs. It may be desirable to know the performance of a specific output, or subset of the output variables (especially for the full kNN model—see 6.5).

Sample: **OutputStatVars 5** (Assuming it is one of several output variables, display summary statistics *only* for the fifth data variable in the training and/or testing data sets)

OutputStatVars -5 (Assuming it is one of several output variables, display joint summary statistics for all *but* the fifth data variable in the training and/or testing data sets)

CHAPTER &8. ERROR MESSAGES

There are basically 3 types of error messages. Most are self-explanatory.

1. **"FYI:..."** are "For Your Interest" advisories on settings that have been configured by default, about which the user should be aware. No acknowledgement is necessary.

Examples:

```
FYI: Connect was not set in .net file. Assuming...
      Connect 1 2      3 5
      Connect 3 5      6 6
```

EXPLANATION: In Connects statements are not present in the .net file, NevProp3 will default to connecting all inputs to all hidden units, and all hidden to all output units. If NHidden is zero, a linear model will be created.

2. **"!!!! Warnings..."** reflect potential conflicts in settings or computational conditions. Most are for the user's awareness, but some require a response.

Examples:

```
!!!! The term \"%s\" is not defined in NevProp. Please check the .net
      file.
```

EXPLANATION: Usually due to a misspelled setting name. Run continues under default setting, unless interrupted by the user.

```
!!!! Stochastic updating is not presently available in Quickprop
      mode.
```

```
!!!! That .wts file wasn't found; try again...
```

3. **"!!!! Fatal Errors..."** reflect major conflicts. Run is aborted.

Examples:

```
!!!! Check .net file for backwards connection.
```

EXPLANATION: The likely cause of this is backwards connections. If you want units and 2 both to be connected to 3 and 4, you must specify:

"Connect 1 2 3 4", rather than "Connect 3 4 1 2"

```
!!!! First data line has non-numeric chars: check data format.
```

EXPLANATION: Data format is not correct: check IDColumn setting (YES if first column of data is a label, else NO); check Headers setting (the number of headers before actual data starts); check Connect statement (format is 2 pairs of integers: start1 end1 start2 end2)

Appendix I

Cross-referencing of Common Statistical & Artificial Neural Network Jargon (Loosely) Corresponding Terms

Originally constructed by Warren S. Sarle saswss@unx.sas.com Sept 7, 1993

Modified/Expanded by David B. Rosen and Philip H. Goodman, 20 Oct 1993
(Errors should be assumed to be ours) <rosen@unr.edu> <goodman@unr.edu>

Particularly loose correspondences are marked by ~. Often we supply a definition using statistical terminology (or neural network terms in uppercase when these have their own entry).

Note that there are many subjects and terms in each of the two fields that have no obvious relationship to or counterpart in the other. In particular, many important types of neural network and statistical models are omitted here for this reason.

There is disagreement in the neural network literature on how to count total layers -- some people count inputs as a layer and some don't. So it is often preferable to specify the number of *hidden* layers instead.

Neural Network Jargon	Statistical Jargon
=====	=====
Neural networks (NN)	a class of flexible nonlinear regression and discriminant models, data reduction models, and nonlinear dynamical systems consisting of an often large number of UNITS interconnected in often complex ways, and often organized into LAYERS.
Unit, Node, Neuron	Simple computing element, sometimes nonlinear. Computes a single scalar value, sometimes representing a term in the model.
Layer	Functional grouping of UNITS calculating a vector quantity or sharing a common level in the organization of the NN model. Usually the connectivity of the network is symmetric with respect to permutation of UNITS within a layer, but not between layers. In a FEEDFORWARD NEURAL NETWORK, a layer is usually the largest subset of UNITS receiving input only from "previous" layers and sending its output only to "forward" layers.
Feedforward neural networks	neural networks whose UNITS (or LAYERS) can be ordered so that the calculations of each UNIT do not affect those of "previous" UNITS, so e.g. the model's predictions can be calculated directly and immediately from the independent

	variables and the parameters.
Recurrent neural networks	neural networks with arbitrary connectivity, so their state evolves iteratively or dynamically in time.
Multilayer feedforward NN	1) NN (usu. regression and discriminant models) comprising a hierarchical composition of models, each successive composition being represented by a LAYER whose regressors may themselves include derived quantities calculated by previous layers. 2) [loose usage] MULTILAYER PERCEPTRON
Multilayer perceptron (MLP)	Multilayer feedforward NN usually constructed from UNITS that each represent a generalized linear model, with inverse link function often a sigmoidal SQUASHING FUNCTION (such as logistic). With one HIDDEN LAYER, approximates ~~~Projection pursuit regression.
Cascade (Network Architecture)	Multilayer feedforward NN, usu. MLP, with one UNIT per HIDDEN LAYER, each connected to all previous units. UNITS usually added and trained successively, as in projection pursuit, rather than simultaneously/globally.
Cascade correlation (CasCor)	An/the original CASCADE model. Uses a residual-correlation criterion rather than least squares or max likelihood; primarily for discrimination.
Radial basis function (RBF)	Multilayer feedforward NN regression (or discriminant) model is sum of radially-symmetric (often gaussian) basis functions, each corresponding to a HIDDEN UNIT.
Normalized radial basis function (NRBF) network	Multilayer feedforward NN regression (or discriminant) model corresponding to conditionalizing a radial (often gaussian) mixture model for joint $p(yx)$. (Nonparametric version is ~GRNN, i.e. kernel regression.)
Squashing function	bounded function with infinite domain, often used as an ACTIVATION function.
Semilinear function	differentiable nondecreasing function
Connectivity, Architecture	Model, Class of model
Training, Learning, Adaptation	Parameter Estimation, Model fitting, Optimization
Supervised learning	Regression, Discriminant analysis (Use sample response values.)
Unsupervised learning	Principal components, Cluster analysis, Data reduction (Does not use sample response values as such.)
Training data	Sample data, estimation data, construction data, design data

Test data	(Cross)validation data, target data, holdout data
Pattern	Observation, Case
Reflectance pattern	an observation normalized to sum to 1
Binary(0/1), Bivalent(-1/1)	Binary, Dichotomous
Symbolic	(Unordered) Categorical
Input (Unit), Feature	Independent variable, predictor, regressor, explanatory variable, carrier, covariate
Input (loose usage)	Observation, Case
Net input (to a unit)	Quantity calculated by UNIT before applying ACTIVATION FUNCTION.
Output [Units or Layer]	Dependent variable(s) (observed or predicted value)
Output of network	Value predicted by model
Output of unit (or Layer)	Quantity calculated by UNIT (or LAYER)
Hidden Unit or Layer	Calculates intermediate quantities in the model.
Activation (of a Unit)	Value of any term or subcalculation in model.
Input (Unit) Activation	Value of independent variable(s).
Network Reponse, Output	Predicted value
Training values, Target values, True output values	Observed values, Responses
Shift register, (Tapped) (time) delay (line), Input window	Lagged variable
Errors	Residuals
Noise	Error term
Generalization	True performance of model on underlying population distribution (as opposed to apparent performance on estimation sample data)
Adaline (ADaptive LInear NEuron)	~Generalized linear model optimized by DELTA RULE.
Perceptron (no hidden units)	~Generalized linear model (GLIM). Originally with thresholded output and fit by perceptron learning rule.
Perceptron learning rule	Iterative algorithm using only a binary indicator of misclassification to fit a perceptron (no hidden units) for linear discrimination.
Activation function, Signal function, Transfer function	~Inverse link function in GLIM (but appears in hidden units in network as well)

Weights, Synaptic weights, Connection Strengths, Long-term Memory traces	Model parameters, (Regression) coefficients
~Bias weight	~Intercept parameter
~Shortcuts, Jumpers (direct connections from input to output)	~Main effects
Functional links	Interaction terms or transformations
Second-order network	Quadratic regression, Response-surface model
Higher-order network	Polynomial regression,
Instar, Outstar	differential equations for gated stochastic approximation of arithmetic mean or centroid
Delta rule, adaline rule, Widrow-Hoff rule, LMS rule	gradient descent (often stochastic) for training a linear perceptron/adaline (no hidden layer)
Generalized Delta Rule	Gradient descent (often stochastic) for training a multilayer network using BACKPROPAGATION.
Backpropagation	1) Computation of derivatives of multilayer network, starting with output layer and proceeding recursively back towards input layer. 2) GENERALIZED DELTA RULE 3) (loose usage) MULTILAYER FEEDFORWARD NN
Weight decay	Shrinkage of the parameter estimates (usu. towards zero) via a penalty term in the optimization.
Weigend-Rumelhart weight decay	Shrinkage of the parameter estimates towards a nonzero value via a specific penalty term in the optimization.
Stopped Training, Early Stopping, Nonconvergent Training, Cross-Validation [sic]	Shrinkage of the parameter estimates (towards their initial values near zero) by halting the iterative optimization before convergence, often according to a holdout data set criterion.
Jitter	random noise added to the inputs to shrink the estimates
Pruning, Brain damage	setting some coefficients to zero, model selection
~Probabilistic neural network	~Kernel density estimation and discriminant analysis
~General regression neural network	~Kernel regression
Competitive learning, Adaptive vector quantization	stochastic iterative algorithms for K-means cluster analysis

Learning vector quantization	a form of piecewise linear discriminant analysis using a preliminary cluster analysis
Counterpropagation	~K-means cluster analysis with the clusters used to predict some variables based on cluster assignments computed from other variables
Encoding, Autoassociation	Dimensionality reduction (Independent and dependent variables are the same)
Heteroassociation	Regression, Discriminant analysis (Independent and dependent variables are different)
Epoch	Iteration (a pass over the data set)
Continuous training, Stochastic training, Incremental training, On-line training, Pattern-wise update	Iteratively updating estimates one observation at a time via difference equations. ~Stochastic approximation, except that observations might be drawn many times (with or without replacement) from a fixed finite set of sample data.
Batch training, Off-line training,	Iteratively updating estimates only after each complete EPOCH, as in
Epoch-wise update	traditional nonlinear optimization algorithms

Appendix II:

Resources

“FAQ”— FREQUENTLY ASKED QUESTIONS ABOUT NEURAL NETWORKS

The latest version of the FAQ is available as a hypertext document, readable by any WWW (World Wide Web) browser, under the URL "ftp://ftp.sas.com/pub/neural/FAQ.html". This version is updated more frequently than the archived copies.

The FAQ posting is archived in the periodic posting archive on host rtfm.mit.edu (and on some other hosts as well). Look in the anonymous ftp directory "/pub/usenet/news.answers/ai-faq/neural-nets". The filenames are "part1", "part2", ... "part7". If you do not have anonymous ftp access, you can access the archive by mail server as well. Send an E-mail message to mail-server@rtfm.mit.edu with "help" and "index" in the body on separate lines for more information.

Here is the contents of the FAQ file:

===== Questions =====

Part 1: Introduction

- --- What is this newsgroup for? How shall it be used?
- --- What is a neural network (NN)?
- --- What can you do with an NN and what not?
- --- Who is concerned with NNs?
- --- How are layers counted?
- --- How are NNs related to statistical methods?

Part 2: Learning

- --- How many learning methods for NNs exist? Which?
- --- What is backprop?
- --- What are conjugate gradients, Levenberg-Marquardt, etc.?
- --- Why use a bias input?
- --- Why use activation functions?
- --- What is a softmax activation function?
- --- What is overfitting and how can I avoid it?
- --- How many hidden units should I use?
- --- How can generalization error be estimated?
- --- What are cross-validation and bootstrapping?
- --- Should I normalize/standardize/rescale the data?
- --- What is ART?
- --- What is PNN?
- --- What is GRNN?
- --- What about Genetic Algorithms and Evolutionary Computation?
- --- What about Fuzzy Logic?

Part 3: Information resources

- --- Good literature about Neural Networks?
- --- Any journals and magazines about Neural Networks?
- --- The most important conferences concerned with Neural Networks?
- --- Neural Network Associations?
- --- Other sources of information about NNs?

Part 4: Datasets

- --- Databases for experimentation with NNs?

Part 5: Free software

- --- Freely available software packages for NN simulation?

Part 6: Commercial software

- --- Commercial software packages for NN simulation?

Part 7: Hardware, etc.

- --- Neural Network hardware?

- --- Unanswered FAQs

=====

- ¹ We include "1" as the first term in \mathbf{X} to simplify later vector notation; the "1" is actually used to multiply a constant intercept(bias, threshold).
- ² McCullogh, P and Nelder, JA (1989). Generalized Linear Models (2nd ed.). London: Chapman and Hall.
- ³ Liao, TF (1994). Interpreting probability models. Sage University Paper series on Quantitative Applications in the Social Sciences, 07-101. Beverly Hills, CA: Sage.
- ⁴ Intrator O, Intrator N. Interpreting Neural-Network Models. Unpublished technical report, September 1993.
- ⁵ Efron B, Tibshirani RJ (1993). An Introduction to the Bootstrap. London: Chapman and Hall.
- ⁶ Rubin, D.B. (1978), ``Multiple Imputations in Sample Surveys -- A Phenomenological Bayesian Approach to Nonresponse,' in American Statistical Association, 1978, Proceedings of the Section on Survey Research Methods, 22-34
- ⁷ Little, R.J.A. and Rubin, D.B. (1987), Statistical Analysis with Missing Data, Wiley, New York.
- ⁸ Little, R.J.A. and Rubin, D.B. (1990), ``The Analysis of Social Science Data with Missing Values' in Modern Methods of Data Analysis, Eds. J. Fox & J.S. Long, 374-409, Sage Publications, Newbury Park.
- ⁹ Little, R.J.A. (1992), ``Regression with Missing X 's: A Review,' Journal of the American Statistical Association, 87, 1227-1237.
- ¹⁰ Little, R.J.A. and Schenker, N. (1995), ``Missing Data,' in Handbook of Statistical Modeling for the Social and Behavioral Sciences, Eds. G Arminger, C.C. Clogg & M.E. Sobel, 39-75, Plenum Press, New York.
- ¹¹ Haitovsky, Y. (1968), ``Missing Data in Regression Analysis,' Journal of the Royal Statistical Society, Ser. B, 30, 67-81.
- ¹² Bucks, S.F. (1960), ``A Method of Estimation of Missing Values in Multivariate Data Suitable for Use with an Electronic Computer,' Journal of the Royal Statistical Society, Ser. B, 22, 302-306.
- ¹³ Madow, W.G., Nisselson, H., and Rubin, D.B. (Eds.) (1983), Incomplete Data in Sample Surveys (Vols. 1-3), Academic Press, New York.
- ¹⁴ Rubin, D.B. (1987), Multiple Imputation for Nonresponse in Surveys, Wiley, New York.
- ¹⁵ Dempster, A.P., Laird, N.M. and Rubin, D.B. (1977), ``Maximum Likelihood from Incomplete Data via the EM Algorithm,' Journal of the Royal Statistical Society, Ser. B, 39, 1-38.

-
- ¹⁶ Tanner, M.A. and Wong, W.H. (1987), ``The Calculation of Posterior Distributions by Data Augmentation,' ' Journal of the American Statistical Association, 82, 528-550.
- ¹⁷ Geman, S. and Geman, D. (1984), ``Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images,' ' IEEE Transactions on Pattern Analysis and Machine Intelligence, 6, 721-741.
- ¹⁸ Shao J. Linear model selection by cross-validation. Journal of the American Statistical Association 1993;88:486-494.
- ¹⁹ Efron B. Estimating the error rate of a prediction rule: improvement on cross-validation. Journal of the American Statistical Association 1986;78:316-331.
- ²⁰ Pickard RR, Cooke. Cross-validation of regression models. Journal of the American Statistical Association 1984;79:575-583.
- ²¹ Efron B, Tibshirani R. An introduction to the bootstrap. NY, 1993; Chapman & Hall.
- ²² Harrell, FE Jr. Prediction outcomes: applied survival analysis and logistic regression. Contact the author for availability.
- ²³ Finnoff, W., Hergert, F., & Zimmermann, H. G. (1992). Improving model selection by nonconvergent methods. Neural Networks, 6, 771-783.
- ²⁴ Wang, C., Venkatesh, S. S., & Judd, J. S. (1994). Optimal stopping and effective machine complexity in learning. In Proceedings of the Conference on Neural Information Processing Systems, Denver, CO, Nov. 1993, printed Feb. 7, 1995.
- ²⁵ Amari, S., Murata, N., Muller, K.-R., Finke, M., & Yanh, H. (1995). Asymptotic statistical theory of overtraining and cross-validation. Univ. of Tokyo Tech. Report METR 06-95, obtained from archive.cis.ohio-state.edu/pub/neuroprose/amari/overtraining.ps.Z
- ²⁶ MacKay DJC. Probably networks and plausible predictions—a review of practical Bayesian methods for supervised neural networks. Technical report, 1995. mackay@mrao.cam.ac.uk