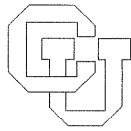


**Theonet:
A Connectionist Expert System
For Solar Flare Forecasting**

Richard Fozzard

CU-CS-442-89 August 1989



University of Colorado at Boulder

DEPARTMENT OF COMPUTER SCIENCE

ANY OPINIONS, FINDINGS, AND CONCLUSIONS OR RECOMMENDATIONS EXPRESSED IN THIS PUBLICATION ARE THOSE OF THE AUTHOR(S) AND DO NOT NECESSARILY REFLECT THE VIEWS OF THE AGENCIES NAMED IN THE ACKNOWLEDGMENTS SECTION.

THEONET:
A CONNECTIONIST EXPERT SYSTEM
FOR SOLAR FLARE FORECASTING

by

RICHARD LANE FOZZARD

B.S., Stanford University, 1978

M.S., Stanford University, 1980

A thesis submitted to the
faculty of the Graduate School of the
University of Colorado in partial fulfillment
of the requirements for the degree of
Master of Science
Department of Computer Science
1989

This thesis for the Master of Science degree by


Richard Lane Fozzard

has been approved for the

Department of

Computer Science


by



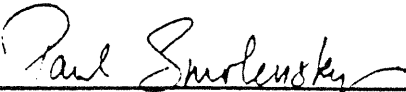
Gary Bradshaw



Clayton Lewis



Michael Mozer



Paul Smolensky

Date _____

Fozzard, Richard Lane (M.S., Computer Science)

A Connectionist Expert System For Solar Flare Forecasting

Thesis directed by Assistant Professor Gary Bradshaw

The Space Environment Laboratory in Boulder has collaborated with the University of Colorado to construct a small expert system that forecasts solar flares, called THEO. It performed as well as a skilled human forecaster. I have constructed *TheoNet*, a three-layer back-propagation connectionist network that learns to forecast flares. *TheoNet* is trained using an on-line database of historical flares which THEO also uses as its major source of information. Tests can be performed by drawing additional data from the database.

Performance was measured with signal detection techniques used in forecasting, instead of the sum-squared error usually associated with connectionist networks. The network's ability to learn and generalize is studied as a function of the number of hidden units and compared with a method equivalent to traditional regression analysis.

Performance was found to be relatively independent of the phase of the solar cycle used for training, suggesting that prediction techniques do not need to refer to the current phase of the sun. It was also independent of many network parameters and other ways of presenting data for training, indicating that a very robust solution was found to the prediction problem.

Since a two-layer network can also learn the task (though with much more data and many more training iterations), it re-

mains to be shown whether any benefits arise from the use of three-layer backpropagation networks on problems of this sort.

Overall prediction accuracy is very good, comparable to the best human experts and the THEO system, as shown by signal detection measures. *TheoNet's* success suggests that a connectionist network can perform some part of knowledge engineering automatically.

ACKNOWLEDGEMENTS

The author would like to thank Jim Winkleman of NOAA for help in obtaining the solar data needed for training and testing. Dave Shaw, Pat McIntosh, Vern Derr, and Dick Grubb, also of NOAA, gave valuable discussions and background about THEO and the problems of forecasting. University of Colorado Symbolics and P3 gurus Andreas Lemke and Jonathan Bein eased the learning curve, and enabled me to create a powerful and interactive simulation environment. Planning sessions with Clayton Lewis and Mike Mozer (thesis committee members) generated goals, direction, and enough interesting ideas and questions to fill several doctoral theses. Paul Smolensky introduced me to the connectionist approaches, taught the Advanced Connectionism seminar that gave birth to *TheoNet*, and provided inspiration and ideas throughout the project.

The lion's share of thanks must go to Gary Bradshaw who was not only a founding father of the THEO expert system, but first had the inspiration that a connectionist network should be tried out on the same data. Gary's extensive background in machine learning and psychology gave invaluable guidance in the research process, and assistance with analyzing the results. He also supplied patience with all the last minute scrambling for publication and thesis defense, and above all, encouragement that we were really on to something.

CONTENTS

| | | |
|---------|---|----|
| CHAPTER | | |
| I. | INTRODUCTION | 1 |
| | Background | 1 |
| | Motivation | 3 |
| II. | TECHNICAL APPROACH | 6 |
| | Network Description | 6 |
| | Simulation Details | 9 |
| III. | RESULTS | 12 |
| | Hardware and Software Issues | 13 |
| | Data Presentation | 14 |
| | Techniques of Measuring Performance | 16 |
| | Comparisons with the Experts | 19 |
| | Training and Testing Effects | 21 |
| | Size of Training Set | 21 |
| | Multiple Day Chunks (Spacing Granularity) | 22 |
| | Phase of Solar Cycle | 23 |
| | Parameter and Architecture Sensitivity | 25 |
| IV. | CONCLUSIONS | 29 |
| | Summary of Results | 29 |
| | Future Work | 30 |
| | Final Remarks | 32 |
| | REFERENCES | 35 |

APPENDIX

| | | |
|----|---------------------------|----|
| A. | THEO DATABASE EXAMPLES | 37 |
| B. | SELECTED PERFORMANCE DATA | 40 |
| C. | SOURCE CODE | 42 |

CHAPTER I

INTRODUCTION

Can neural network learning algorithms let us build “expert systems” automatically, merely by presenting the network with data from the problem domain? I tested this possibility in the domain of solar flare forecasting, where traditional analysis methods have failed, but an expert system has been developed that is at least as good as the expert.

Background

Early work on predicting solar flares involved formal methods using multi-variant discriminant analysis and logistic regression. Less formal methods such as cluster analysis and the maximum entropy method were also tried. These methods generally showed no improvement over human forecasters and were often worse under some conditions [see survey in Sawyer 86 Ch. 4].

Knowledge-based expert systems attempt to capture the knowledge in a restricted domain of a human expert into a computer program and make this knowledge available to users with less experience. Such systems could be valuable as assistants to forecasters or as training tools. In the past four years, the Space Environment Laboratory (SEL) in Boulder has collaborated with the Computer Science and Psychology Departments at the University of

Colorado to construct a small expert system for solar flare forecasting incorporating the rules and strategies of Pat McIntosh, a senior solar physicist at SEL. The project convincingly demonstrated the possibilities of this type of computer assistance, which also proved to be a useful tool for formally expressing a forecast procedure, verifying its performance, and instructing novice forecasters. The system, named THEO (an OPS-83 production system with about 700 rules), performed as well as a skilled human forecaster using the same methods, and scored better than actual forecasts made at SEL in the period covered by the test data [Lewis and Dennett 1986, Shaw 89].

In recent years connectionist (sometimes called “non-symbolic” or “neural”) network approaches have been used with varying degrees of success to simulate human behavior in such areas as vision and speech learning and recognition [Hinton 1987, Leaky and Sejnowski 1988, Sejnowski and Rosenberg 1986, Elman and Zipser 1987]. Logic (or “symbolic”) approaches have been used to simulate human (especially expert) reasoning [see Newell 1980 and Davis 1982].

This has developed into a schism between symbolic and connectionist areas of research within the artificial intelligence/cognitive psychology community. The same problem has rarely been attacked by both approaches, leaving a great deal of uncertainty about the relationships between the two approaches. It is hardly my intent to debate the relative merits of the two paradigms. The intent of this project is to apply a connectionist learning technique (multi-layer back-propagation) to the same

problem, even the very same database, used in an existing successful rule-based expert system. There has been other work in connectionist or hybrid connectionist-symbolic expert systems [Mozer 87, Gallant 88, Goodman 89], but at this time we know of no current work where the connectionist system has been directly compared to a pre-existing symbolic system in terms of the performance measures in the problem domain.

Motivation

Forecasting (solar flare, weather, etc.), as described by those who practice it, is a unique type of informal reasoning within very soft constraints supplied by often incomplete and inaccurate data. Current knowledge of solar physics is inadequate to provide "perfect" forecasts, much as atmospheric physics has not been able to remove the guesswork from terrestrial weather forecasts [Sawyer 84]. The problem is one with inherent uncertainty. Good forecasters use a combination of knowledge, intuition, and rules of thumb to generate predictions. This type of problem bears many similarities to the methods doctors use to diagnose diseases and was the justification behind rule-based approaches such as MYCIN [Buchanan 84]. Thus it was felt that the expert system approach could be used on solar flare forecasting, and THEO was the result.

Solar flares are explosions in the corona of the sun. These explosions can sometimes be of tremendous proportions, releasing the energy equivalent of ten million hydrogen bombs within an hour. Flares are usually associated with active regions, or groups of sunspots, that appear on the sun [Wentzel 89]. Observations of ac-

tive regions are imperfect and error-prone - not all of the characteristics may be detectable or interpretable. Connectionist networks have been shown to exhibit a great deal of fault tolerance for these sorts of input data [see Hinton 81 and Rumelhart 86 for surveys]. It may also be that some of the reasoning involves pattern matching and dimensionality reduction of the different categories of data. This led to the hope that a connectionist network might be able to learn the necessary internal representations to cope with this sort of task.

The problem of flare forecasting, then, is seen to be one apparently needing logical (symbol manipulation) techniques as well as pattern recognition and a great degree of fault tolerance. This is what makes the problem domain ideally (though hardly uniquely) suited to examination by both approaches.

The goal of this work is to examine the practicality of working with a connectionist system to attack the sorts of problems thought only to be approachable with knowledge-based techniques. Must the knowledge embodied in the rules in an expert system be explicitly coded? Or can the equivalent be learned by a connectionist network? Can that network then actually generate good predictions?

In this work, I present a connectionist network with the same categories of solar data used by the THEO expert system. Pairing each presentation with actual flare occurrence data should give the network the potential of learning internal representations of the relationships between the data. Once this is done, the network's ability to predict flares from the associations it has learned

will be tested using a measure in common use for evaluating forecasting performance. This can then be used to compare it against other methods that have been used for flare forecasting at the SEL.

Of particular concern for the practical development of expert systems is how difficult good forecasting performance is to achieve. What sort of dependencies for learning are there on the distribution or types of data presented to *TheoNet*? The effects of training with data from different phases of the solar cycle and training with data clumped together in periods one or more weeks long will be studied. Also, experiments will be presented that investigate how many days worth of data are needed to do the learning.

Even the simple connectionist network used here has a number of possible variations on learning parameters and architectures. Many problems solved by connectionist systems have been critically dependent on these and there are as yet no well-accepted formal methods for determining the optimal ones. Building successful connectionist networks can involve a great deal of trial and error. I will look at how important the size of its hidden layer is to *TheoNet*'s performance and what this implies about the nature of the solar flare forecasting problem.

CHAPTER II

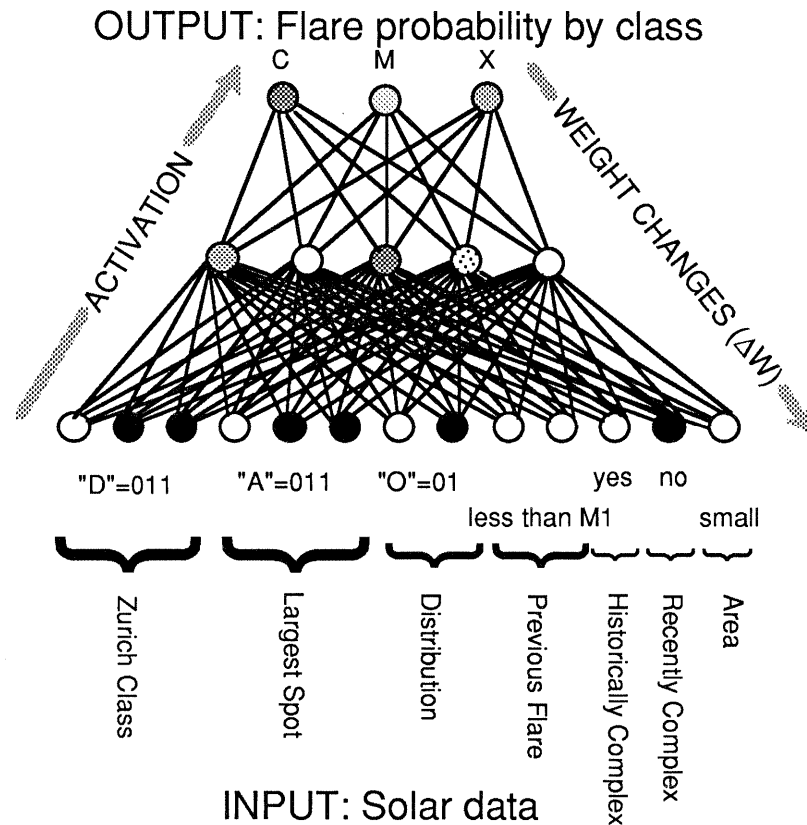
TECHNICAL APPROACH

Network Description

The *TheoNet* network model has three layers of simple, neuron-like processing elements called “units”. The layers are connected with feedforward links (see **Figure 1** for a diagram of the implementation of the network used for most of the testing). The large layer of 13 units at the bottom of the figure is the input layer and is clamped to a pattern that is a distributed representation of the solar data for a given day. For the central (“hidden”) and top (“output”) layers, each unit's output (called “activation”) is the weighted sum of all inputs from the units in the layer below:

$$y_j = \frac{1}{1 + e^{-x_j}} \quad \text{where: } x_j = \sum_i y_i w_{ji} - \theta_j \quad (1)$$

where y_i is the activation of the i th unit in the layer below, w_{ji} is the weight on the connection from the i th to the j th unit, and θ_j is the threshold of the j th unit. The weights are chosen from a uniform distribution in the range -1.0 to +1.0, but are allowed to vary beyond that range. Thus activation passes from bottom to the top of the network as shown by the arrow on the left.



1. Modified Zurich class (7 possible values: A/B/C/D/E/F/H)
2. Largest spot size (6 values: X/R/S/A/H/K)
3. Spot distribution (4 values: X/O/I/C)
4. Previous flare activity (less than M1 / M1 / more than M1)
5. Historically complex (yes/no)
6. Recently became complex on this pass (yes/no)
7. Area (small/large)

Figure 1. Five hidden unit implementation of *TheoNet*

A least mean square error learning procedure called *back-propagation* is used to modify the weights incrementally for each input data pattern presented to the network [Rumelhart 86]. This compares the output unit activations with the “correct” (what actually happened) solar flare activity for that day. This gives the weight update rule:

$$\Delta w_{ji}(t+1) = -\epsilon \nabla E(t) + \alpha \Delta w_{ji}(t) \quad (2)$$

where $\nabla E(t)$ is the partial derivative of least mean square error, ϵ is a parameter called the *learning rate* that affects how quickly the network attempts to converge on the appropriate weights (if possible), and α is called the *momentum* which affects the amount of damping in the procedure. The learning rate and momentum parameters used were 0.2 and 0.9, respectively, but these values were not critical; a wide range of values still resulted in good network convergence as well as prediction performance. The values used in the test reported here follow Hinton [1987], except that no weight decay was used. Weights were updated after each presentation of an input/output pattern.

The three output units at the top of **Figure 1** code for each of the three classes of solar flares to be forecasted. The three classes C, M, and X represent flares of increasing intensity one order of magnitude apart. Individual output activations correspond to the relative likelihood of a flare occurrence of that class within the next 24 hours (see the analysis below). The 13 input units at the bottom provide a distributed coding of the seven categories of input data that are currently fed into *TheoNet*; these are also given to the “default” mode of the expert system THEO¹. Three binary

¹The expert system THEO has two modes of operation: default and interactive. In default mode, it simply applies its rules to the raw solar data in the database; in interactive mode, the user can feed additional data into it. *TheoNet* did not use any of this additional data.

THEO default mode uses seven categories that are not

(on/off) units code for the seven classes of sunspots, two units code for spot distribution, and so on. The hidden units at the center mediate the transfer of activation from the input to the output units and provide the network with the potential of forming internal representations. A backpropagation network may have any number of hidden units. Most of the experiments described below used a hidden layer of five units.

Simulation Details

The P3 connectionist network simulator from David Zipser of University of California at San Diego's parallel distributed processing (PDP) group [Zipser 86] was used to implement and test *TheoNet* on Symbolics 36xx workstations. This simulator provides an object-oriented environment that allows users to create networks of nodes and their interconnections. It provided an interactive graphical interface (**Figure 2**) for working with the network simulation and allowed the use of Lisp code to execute activation and learning functions, compile statistics, etc.

A number of capabilities were added to allow for rapid construction and performance testing of variations of the network model and the data presented to it.

Multiple active sunspot regions may be present on any

input to *TheoNet*. This is for historical reasons; the original version of *TheoNet* had ten categories of data available to it that were used by an earlier version of THEO. Three of these are no longer used by the current THEO, which now uses 14 categories of data (seven old plus seven new).

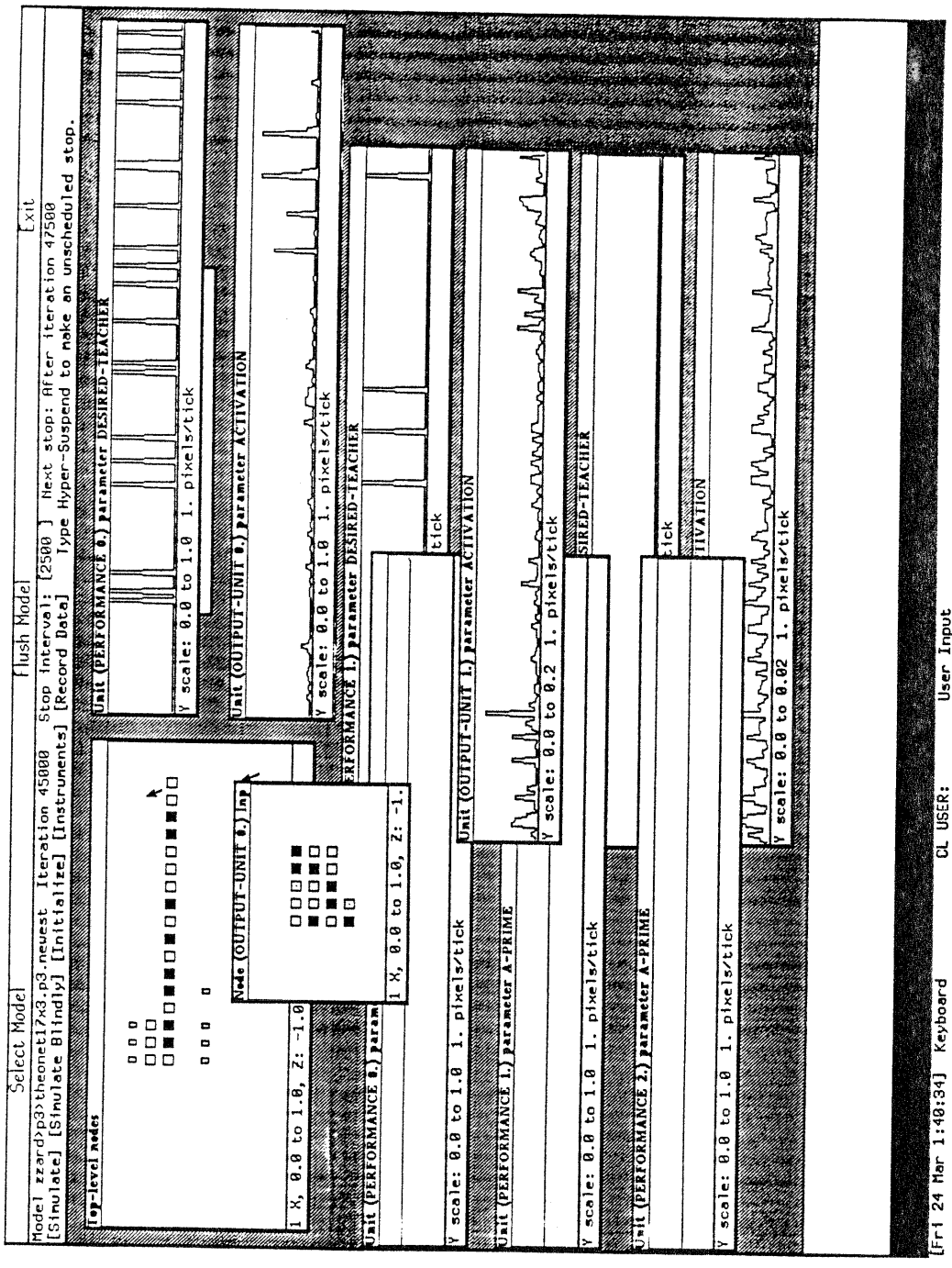


Figure 2. P3 simulator interactive graphical interface

given day. Individual sunspot region predictions were combined into whole-sun flare predictions since forecasters are mainly concerned with whether flares will occur, regardless of which region they come from. This was done by special-purpose units outside the actual network architecture using the simple probability rule:

$$P(\text{whole}) = 1 - \prod (1 - P(\text{region}_i)) \quad (3)$$

THEO used this combination rule, but also augmented the single procedure with additional rules about how nearby regions could interact to affect the overall whole-sun probability. *TheoNet* did not attempt exploit interactions in deriving whole-sun predictions. Results reported in this paper are drawn from the whole-sun predictions.

TheoNet directly reads the active region database files used by the THEO expert system. These files also contain the region-by-region and whole-sun predictions generated by THEO. This has allowed testing to include data up to the current date. (However, solar data and THEO's predictions have only been consistently available in computer-readable form since April 1987.)

Training or testing the model on any subset of the database may be interactively chosen from the P3 interface. Learning parameters and the number of hidden units used are also interactively variable.

The model provides the ROC a' performance measure (see the **RESULTS** section below) in “real-time” calculated after every complete presentation of the data set (“epoch”). The a' performance measure can be displayed continuously as the simulation is

running. These values are available for both THEO and *TheoNet* individual sunspot region and whole-sun predictions for direct comparisons of the two systems.

CHAPTER III

RESULTS

The main thrust of this work is to answer some questions about the practicality of using connectionist techniques to solve “real-world” problems: is *TheoNet* a “Connectionist Expert System” that actually works? And what did it take to get it to work?

Consequently I will discuss hardware and software particulars and the nature of the data the network was evaluated with, as well as the different ways that data was presented to it for both training and testing. Next I provide a justification and description of the signal detection performance measure used. *TheoNet's* prediction performance is then compared to the expert system THEO in both its default and interactive modes, a seven-day average prediction model, and human forecasters from the Space Environment Laboratory.

In determining how robust is *TheoNet's* ability to learn the art of forecasting solar flares, I will examine the performance effects of different methods of training and testing. For the same reason, I will look at how sensitive *TheoNet's* performance is to the common learning parameters (learning rate, momentum, and number of hidden units) of back-propagation networks.

Hardware and Software Issues

TheoNet is, at its core, a simple program. As previously mentioned, it is described in an object-oriented fashion in a simulator language. Ignoring the learning curve associated with the Symbolics, the essential code was written in just a few weeks. No “knowledge engineering” was done: there were no discussions with experts². The raw solar data was simply input to a generic back-propagation network and good performance was obtained with very minimal tweaking. Most of the coding effort (and compute time!) was involved with providing interactive control and real-time performance measures.

TheoNet was simulated using many different variations of learning parameters, input data protocols, and number of hidden units used. Using a Symbolics 3653, building a model would take about 2-3 hours, and individual simulations (train-then-test cycles) 30-60 minutes. The bulk of this compute time was involved with the calculation of performance measures; early versions without the performance calculations would run in roughly one-fourth the time. Time to make any individual prediction from an input data example was less than 100 milliseconds. The database of solar data and THEO predictions occupied about two megabytes. In all, several hundred simulations were run in the process of developing and testing the network.

²To this day, I know less about how to predict solar flares than a novice forecaster - in fact, I don't even know what most of the data categories even refer to.

Data Presentation

The network was trained and tested using raw solar data from the same database as used by the THEO expert system. An excerpt from the database file is included in **APPENDIX A**. The raw solar data and THEO predictions data are available from April 1987 through June 1989 - about 5000 patterns (input/output examples) by sunspot region. Another 4000 patterns, but not prediction data, is intermittently available from 1969, '78, '80, '82, and '86. The total number of days or *julians*³ in the database was 1625. All the major phases of the solar cycle (onset, peak, and minimum) were represented, though not contiguously or completely. Flare occurrence by class (C, M, X) for these periods is shown in **Figure 3** (the figure is compressed to omit the periods from the database that were not available).

The data are often inconsistent and noisy. Many examples have the same input pattern, and in many cases the same input would result in different flare results in the following 24 hours. These sorts of inconsistencies in the data make the job of prediction difficult to systematize and the solution inherently error prone. Thus the domain seems an ideal candidate for the soft-constraints nature of connectionist systems.

To fairly determine the network's actual ability to predict flares, it is important to test it on data not seen in training it - that

³SEL jargon for the sequential integers representing the number of days since Jan 1, 1900

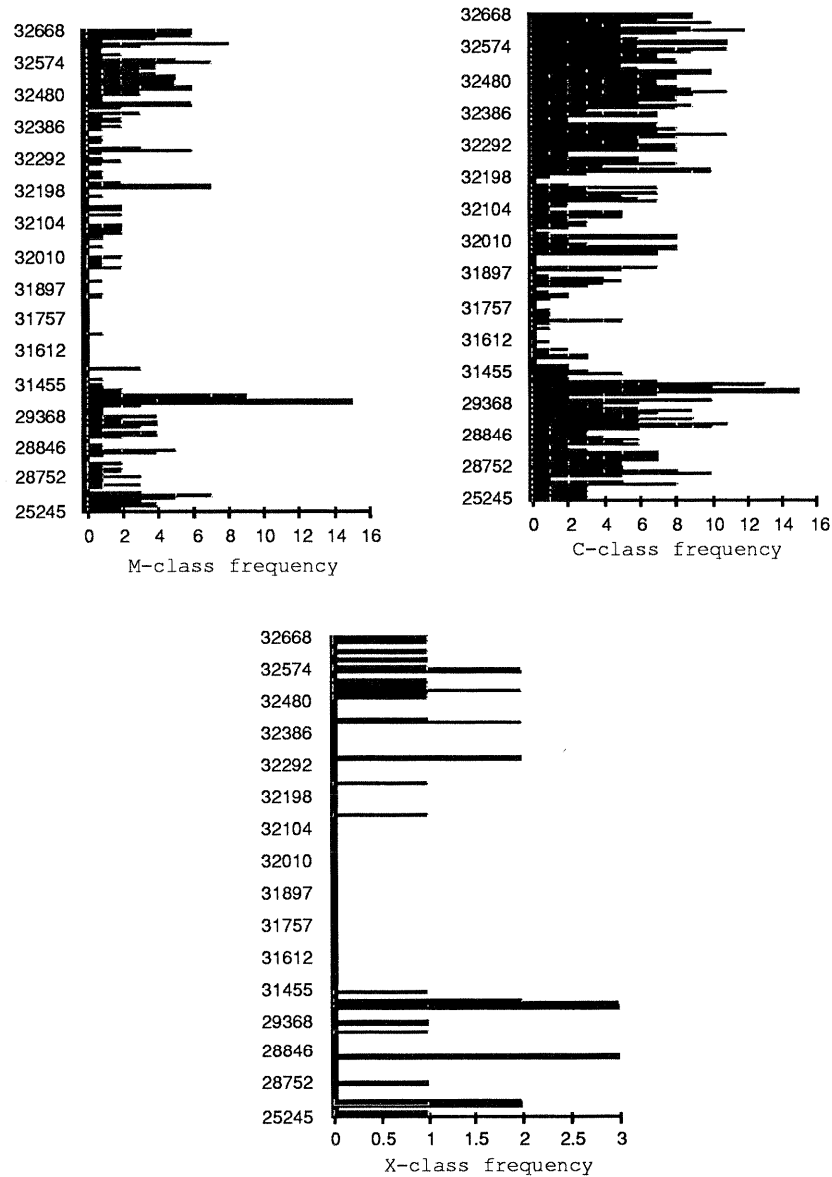


Figure 3. Flare frequency across THEO database

is, how well does it *generalize*?. Since the database is finite, the problem arises on how to divide this database into sets for training and sets for testing - and how might this affect measures of prediction performance?

Three basic methods were used in presenting data to the

the network. **Massed** protocols allow training on all julians within a certain range and testing on another. **Spaced** protocols divide the data into odd (testing) and even (training) clusters of julians. I coined a term to describe the size of these clusters: *spacing-granularity*. For example, a spacing granularity of 14 would assign 2 week groups of data alternately to the testing and training sets; a granularity of 1 would just assign data from odd-numbered julians to testing and even-numbers to training. **Distributed** protocols are a generated by choosing some number of julians in an evenly distributed fashion from the entire data set.

Techniques of Measuring Performance

Analyzing performance of an expert system is best done using measures from the problem domain. Forecasting problems are essentially probabilistic, requiring the detection of signal from noisy data. Thus forecasting techniques and systems are often analyzed using signal detection theory [Spoehr and Lehmkuhle 1982]. If the claim is to be made that *TheoNet* is an “expert system”, it needs to be evaluated using this technique.

An earlier simulation tracked a simple measure called *overall-prediction-error*. This was the average difference over one complete epoch of input patterns between the activation of an output unit and the “correct” value it was supposed to have. This is directly related to the sum-squared error used by the back-propagation method and is the common way of measuring performance of connectionist networks.

While the overall-prediction-error would drop quickly for

all flare classes after a dozen epoches or so, individual weights would take longer to stabilize. Oscillations were seen in weight values if a large learning rate was used. When this was reduced to 0.2 or lower (with a momentum of 0.9), the weights would converge smoothly to their final values.

Overall-prediction-error however, is not a good measure of performance. Since flares happen very rarely, a network that predicted no flare occurrence regardless of the state of the sun (a “just say no” network) would have a very low error measure. Something was needed to indicate whether the network was behaving differently when a flare was about to occur. This, in addition to the desire to use a common domain performance measure, was the reason for turning to signal detection measures.

The system was modified to calculate $P(H)$, the probability of a hit, and $P(FA)$, the probability of a false alarm, over each epoch. These parameters depend on the *response bias*, which determines the activation level used as a threshold for a yes/no response⁴. A graph of $P(H)$ versus $P(FA)$ gives the *receiver operating characteristic* or ROC curve. The amount that this curve is bowed away from a 1:1 diagonal is the degree to which a signal is being detected against background. A summary measure, which can be used to compare different ROC curves, is given by the area under the curve. This measure is known as a' . Guessing or chance performance results in an a' of .50. Forecasting skill is evidenced by a' values that exceed

⁴Even though both THEO and *TheoNet* have a continuous output (probability of flare and activation), varying the response bias gives a continuous evaluation of performance at any output level.

.50. This was the method used for measuring the performance of THEO [Lewis and Dennett 1986].

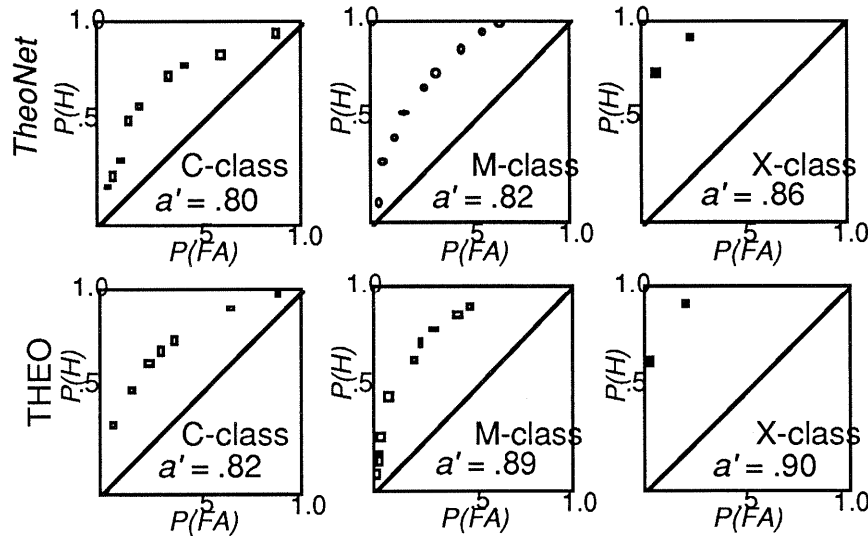


Figure 4. ROC performance curves of *TheoNet* and *THEO*

The network was exposed to the test data before and after training. The test data set was composed of data patterns that the network had not seen during training. After training, the probability of hits was consistently higher than that of false alarms in all flare classes (**Figure 4**). Given the limited data and very low activations for X-class flares, it may or may not be reasonable to draw conclusions about the network's ability to detect these - in the test data sets there would be roughly only 20-30 X-flares. As can be seen in later sections, the a 's for this class were much more variable than C and M classes.

Actual region and whole-sun performance figures from many of the different simulations are given in **APPENDIX C**.

Comparisons with the Experts

How does the prediction performance described here compare to the other methods in use for forecasting flares? Forecasting performance figures are available for the domain expert, other SEL forecasters, the seven day average model, and the interactive and default modes of the expert system THEO.

Both THEO and *TheoNet* generate flare probability predictions for individual sunspot regions and for the sun as a whole on a

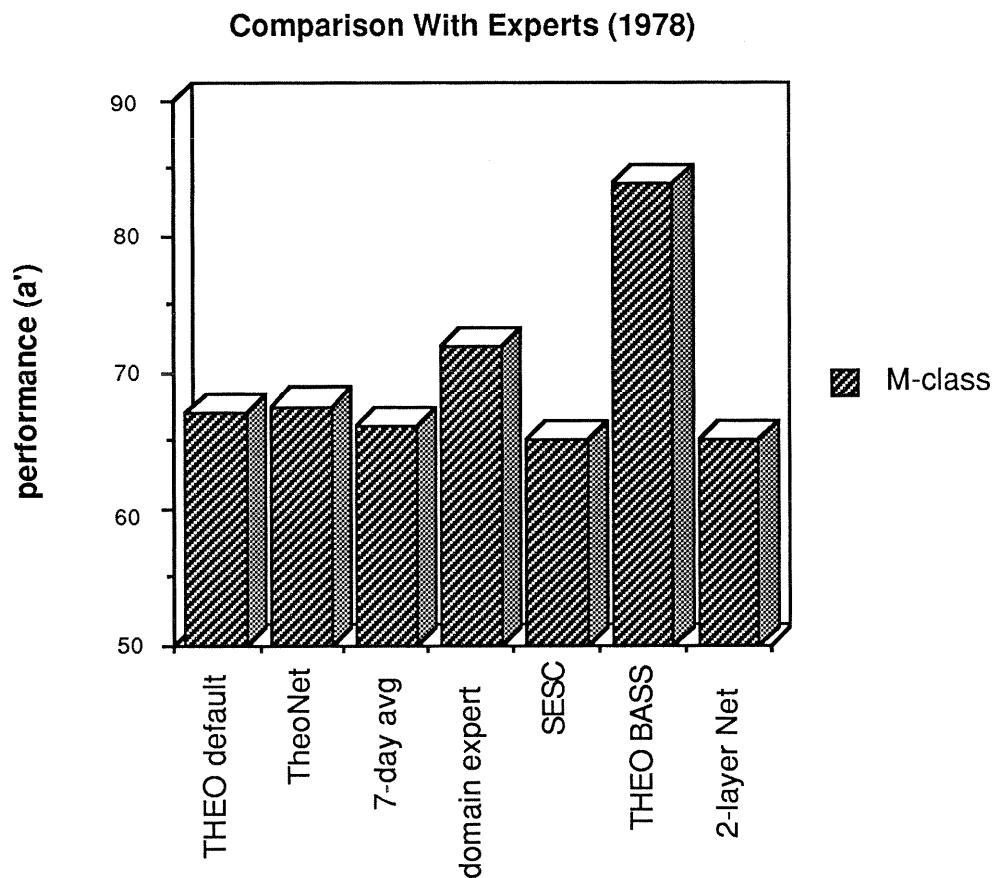


Figure 5. Comparison of prediction methods from 1978

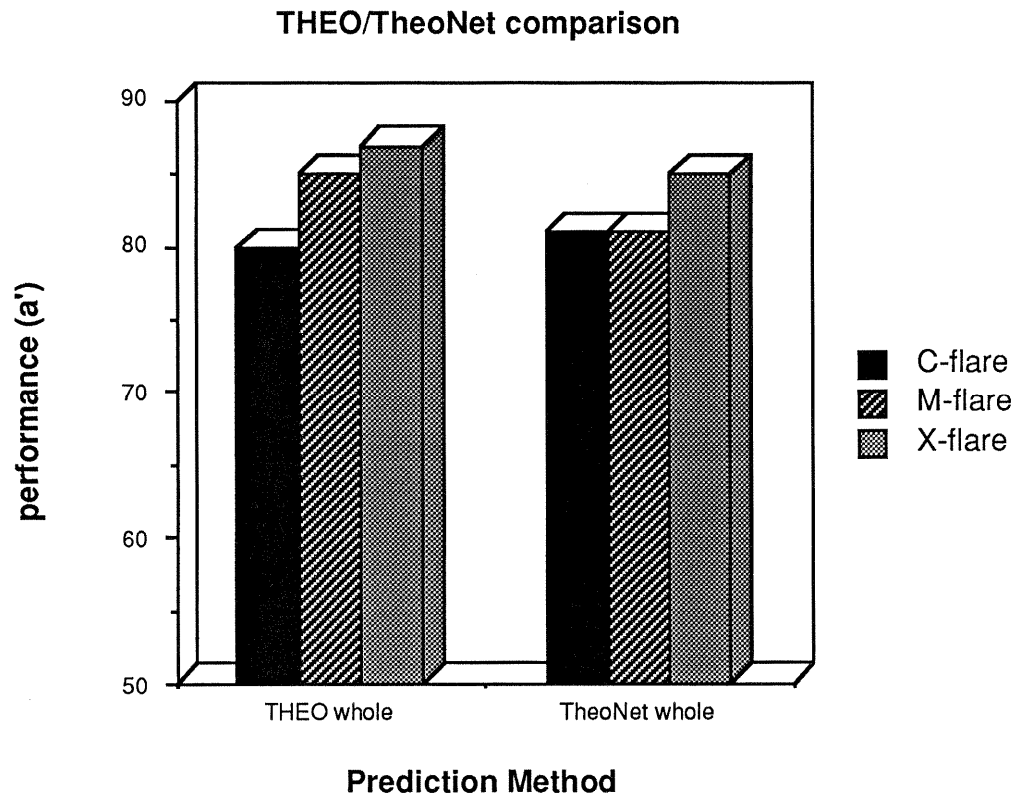


Figure 6. Comparison with expert system

particular day. The simulation was modified to read in THEO's predictions along with the raw solar data and calculate these a 's at the same time and in the same fashion as for *TheoNet*'s own predictions. Data for M-class whole-sun prediction for all methods is available for the 1978 onset period [Gary Bradshaw, personal communication], so it is this period used for the comparisons shown in **Figure 5**.

The 1978 onset period was considered a challenging period for flare prediction and all the methods performed comparably, with the exception of the THEO interactive mode. The improvement for this mode has been attributed to the ability of the

user to change pieces of suspect data [Bradshaw, personal communication] as well as to consider additional data.

Considerably better performance is obtained when testing on more recent data from the solar database (see **Figure 6**). Only figures for the expert system THEO and *TheoNet* are available for this comparison. The values shown are for testing over the period from April 1987 through June 1989. *TheoNet*'s performance was comparable over all three flare classes to that of THEO.

Training and Testing Effects

Size of Training Set

Did *TheoNet* need the thousands of days worth of data in the solar database to achieve good performance? To answer this I used *distributed* protocols (described above under **Data Presentation**) to present it with varying numbers of days worth of solar data for training. The results are shown in **Figure 7** below. The values graphed are the average of several trials with different initial connection weights and the arrows show the minimum and maximum values obtained.

Performance is essentially the same from the maximum of 800 days (half the data set) down to about 100 days. Below this, there is some noticeable drop off to a 's around .70. Measurements of training sets of below 25 days are not useful, since it is at this point that the probability of getting no X-class flares is nearly 50%. Obviously, the network needs a few examples of a flare occurrence to learn. Variance in obtained a 's also increased as the training set

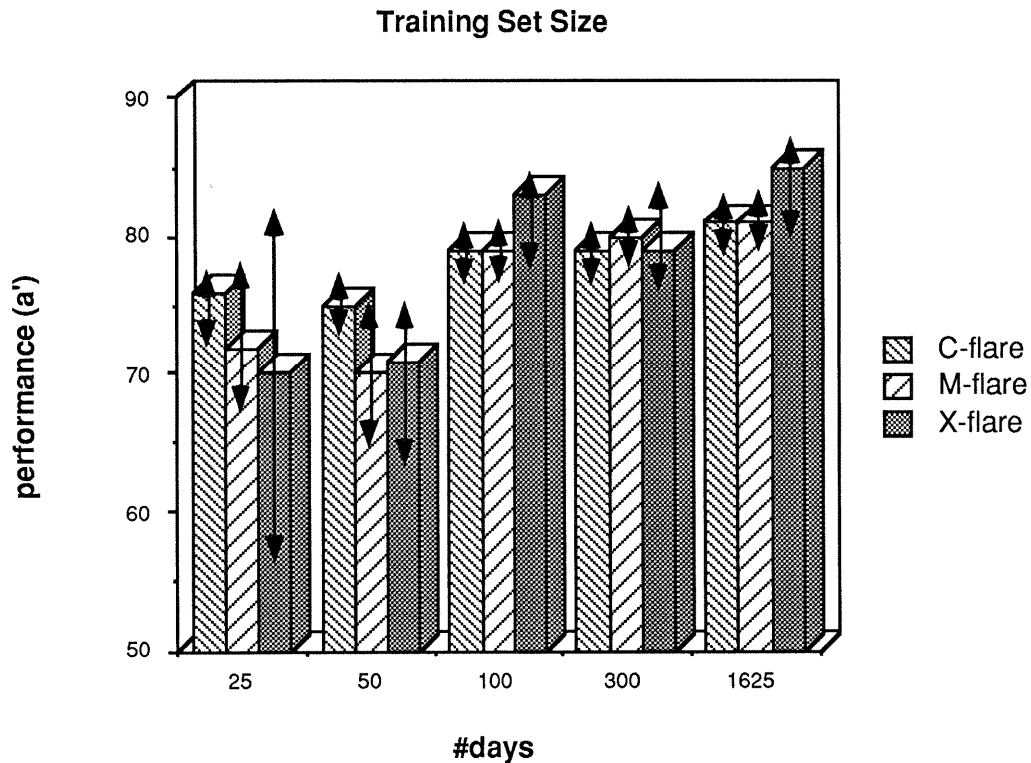


Figure 7. Effect of size of training set

size decreased.

Multiple-Day Chunks (Spacing Granularity)

Since the sun exhibits periodic behavior for 14-21 days at a time, SEL scientists felt training and testing should be done on alternating 14 or 21 day chunks (that is, a *spacing granularity* of 14-21). This is to answer the objection that the network was being tested on data possibly more similar to that on which it had been trained than if the test data is from a different two- to three-week period. However, this made no noticeable difference in the network's ability to learn and generalize over a wide range of granular-

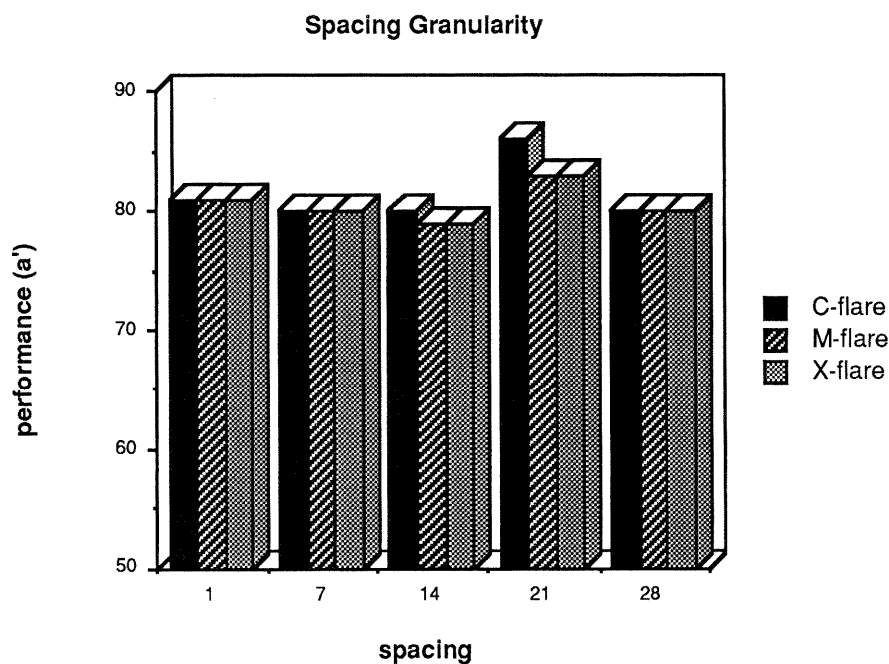


Figure 8. Effect of spacing protocols

ities (see **Figure 8**).

Phase of Solar Cycle

Does a given type of sunspot behave differently in the different phases of the 11 year solar cycle? If so, then training the network with data from only one phase might not lead to valid predictions in another. The previous tests used data from the entire database, covering most of the solar cycle, and so avoided this problem.

Based on sunspot count (which is correlated with, but not the same as, flare frequency), three main phases are recognized by solar scientists. **Minimum** for the quiescent period between activ-

ity peaks, **Onset** for the steeply rising phase, and **Peak** for the period of steady high activity. (These can be seen in the flare frequency diagrams of **Figure 3**.)

For this experiment, training sets consisted of *massed* protocols of 98 days from the 1980 peak (julian 29275-39372) 1986-87 minimum (31425-31522), and the 1989 onset (32500-32597). The protocols were limited to 98 days, since this was all the data that was available from the 1980 peak phase. These were then compared to a *distributed* protocol of 98 days from the entire cycle. All the resulting networks were tested on the entire database (**Figure 9**).

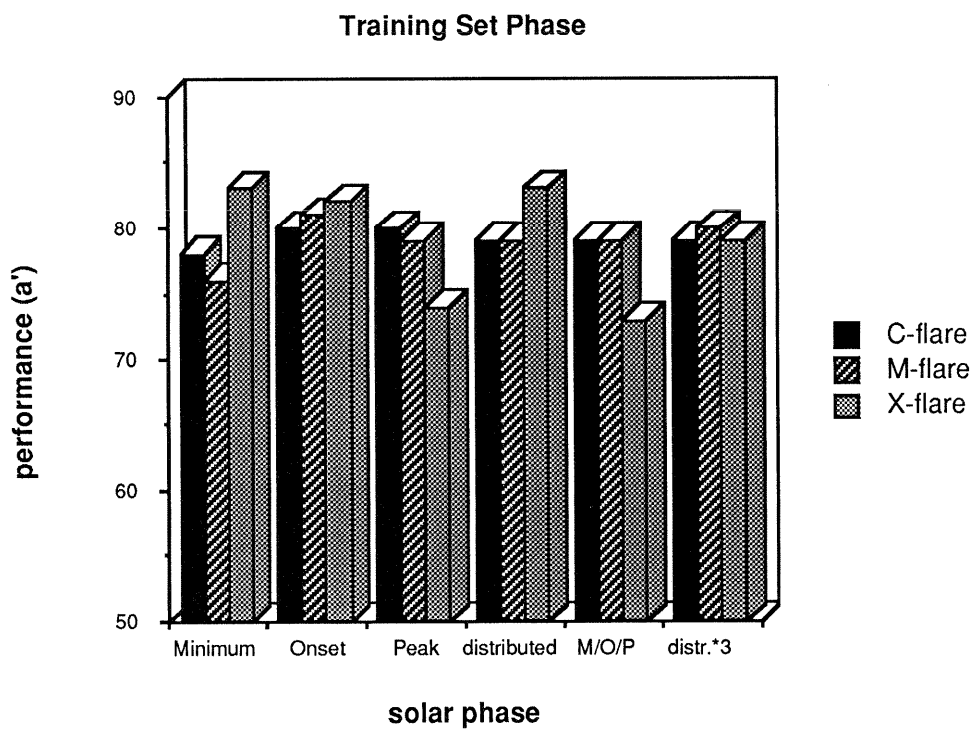


Figure 9. Effect of phase of training set

There were no apparent performance differences among any of the protocols. An additional experiment was done with the 294 day combination of the minimum, onset, and peak periods (M/O/P in Figure 6) as well as a distributed set of 294 days and even this comparison made no difference. So just three months of data is enough for *TheoNet* to learn flare prediction⁵.

Thus it seems reasonable to conclude that no careful (or “knowledgeable”) selection of data was required to successfully train *TheoNet*. Performance was good as long as enough different data patterns were presented. Flare prediction using the characteristics of a sunspot (from the database) thus is apparently independent of the solar phase.

Parameter and Architecture Sensitivity

Connectionist networks using the back-propagation learning algorithm have a number of different parameters that affect the ability to reduce error and to generalize. These affect the way the system searches for the minima of a multi-dimensional error surface. Different learning rates, momentums, and numbers of units in the hidden layer(s) can cause the network to overshoot or oscillate around local minima, or even miss them entirely. It is also possible to have error surfaces with multiple local minima, and the gradient descent procedure used by back-propagation can actually trap the network away from the best solution or global minimum.

⁵Of course, quiescent periods where no flares occur do not lead to useful predictions.

For these reasons, many problems are difficult and sensitive to the values of these parameters, or are even intractable. During development of the network different values of learning rates and momentums were tried. Barring ludicrous values outside of anything reported in the connectionist literature, the system did not seem sensitive at all to these parameters, except that a lower learning rate would simply require more epoches for convergence. For all the formal performance testing, the learning rate was 0.2 and the momentum was 0.9.

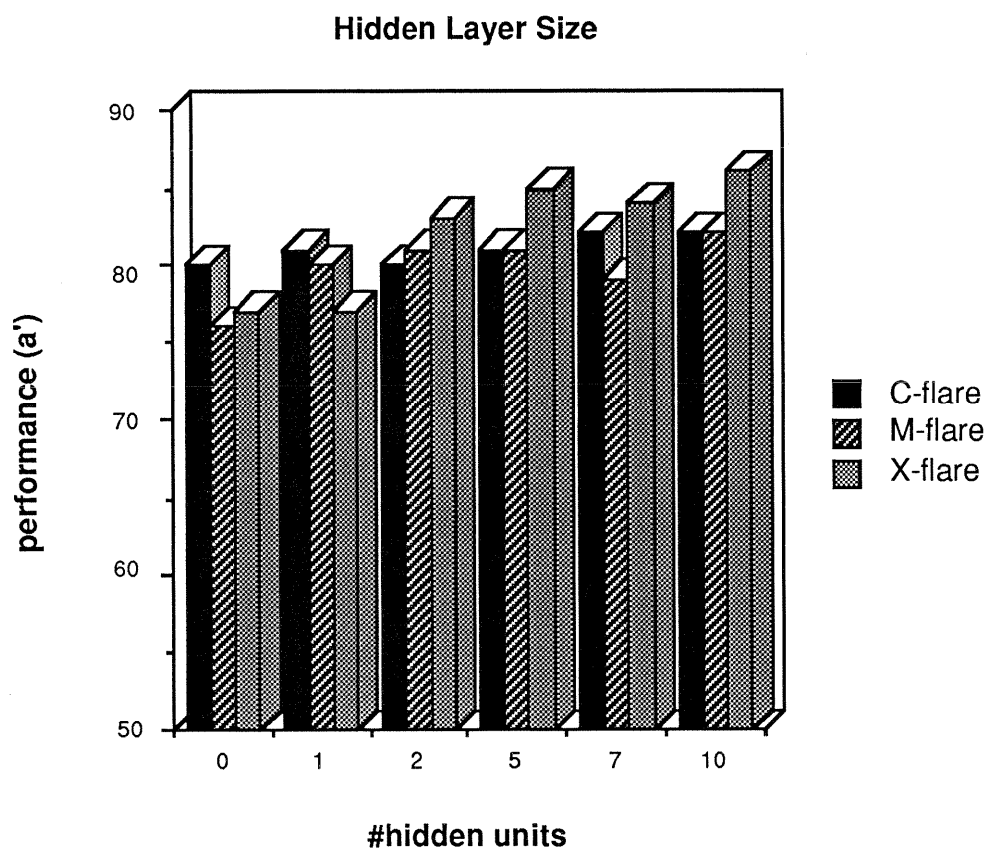


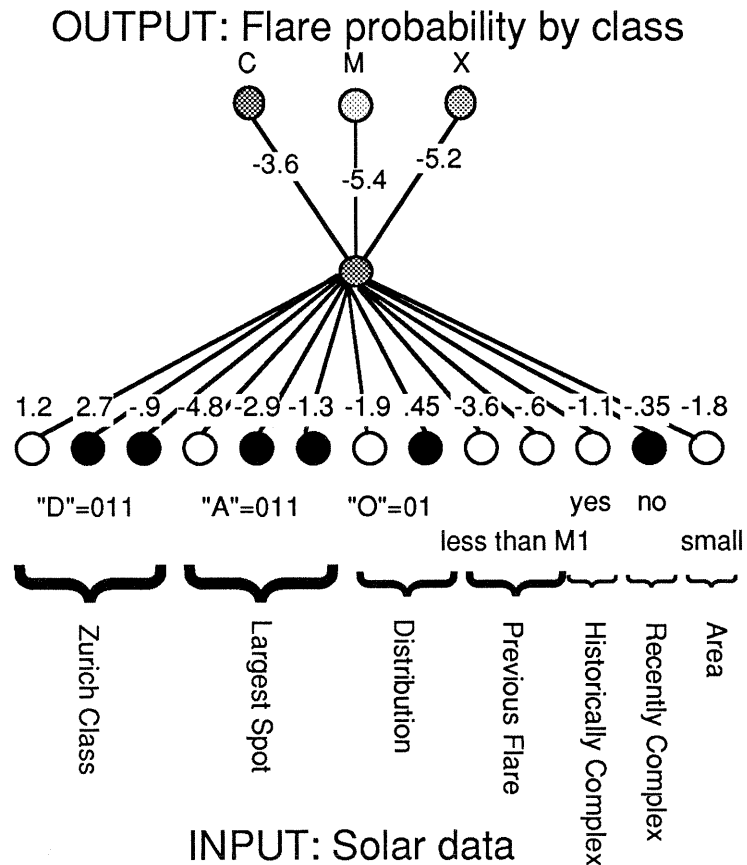
Figure 10. Effect of hidden layer size

The hidden units are where the crucial representations are formed that perform the mapping from input data to output activations (predictions for *TheoNet*). Networks with 1-10 hidden units were constructed, then trained and tested using the simple odd/even (*spaced-granularity* = 1) protocol across the entire database. In addition a simple two-layer network with no hidden units was built and tested in the same way. These are compared in **Figure 10**.

The most surprising result here is that *it did not matter how many hidden units were used!* A single hidden-unit network performed as well as the standard five unit or a ten unit one. This was only performed using the entire database, and leaves open many questions about other possible differences.

Another obvious question is if anything can be concluded from the weights in the simple single hidden-unit network. **Figure 11** shows the weights obtained after a training run on the single hidden-unit network. They are somewhat difficult to interpret due to the arbitrary binary distributed representations. An informal analysis showed that the **Largest Spot** McIntosh classification and **Previous Day** categories are prominent and **Recent** is hardly used, but it is apparent the network is paying significant attention to all the categories of data. Any other interpretations from this result remain unexplored.

The simple two layer (0 hidden units) network was also able to learn the task, but required an order of magnitude (on the order of hundreds of epoches instead of tens) more iterations before weights settled. In preliminary tests (see the data in



1. Modified Zurich class (7 possible values: A/B/C/D/E/F/H)
2. Largest spot size (6 values: X/R/S/A/H/K)
3. Spot distribution (4 values: X/O/I/C)
4. Previous flare activity (less than M1 / M1 / more than M1)
5. Historically complex (yes/no)
6. Recently became complex on this pass (yes/no)
7. Area (small/large)

Figure 11. Connection weights of a 1 hidden-unit network

APPENDIX C), it was also unable to learn X-class flare prediction on smaller samples of the data set (massed or distributed protocols). There is not, however, sufficient data here to draw further conclusions.

CHAPTER IV

CONCLUSIONS

Summary of Results

The connectionist approach used by *TheoNet* was successful in learning to predict solar flares. Its performance was comparable to an expert system that outperforms forecasters at the Space Environment Laboratory. It needed exposure to only about three months of solar data taken from any part of the 11-year solar cycle or distributed throughout it in different fashions. The prediction performance generalized consistently well across the cycle. This also seems to indicate that solar flare prediction from sunspot characteristics is not dependent on the current phase of the solar cycle; forecasting is learnable in just three months by *TheoNet* (or, presumably, a novice forecaster!).

It did surprisingly did not matter for basic performance (other effects have not yet been examined) how many units the hidden layer contained, nor was the network sensitive to other learning parameters. This would lead to the conclusion that the problem has a relatively unconvoluted error surface in the space of possible solutions. Its lack of input data dependencies and small input data requirements reinforce the robustness of the solution it found.

It was possible for a two-layer network to solve the prediction problem, given enough data and simulation time. Since this is mathematically equivalent to a non-linear regression, this should indicate that the problem is simple enough for regression analysis to solve. Preliminary testing indicated that the two layer network needed large amounts of data from across the solar cycle - this could explain the failure of earlier attempts that may not have had enough solar data.

TheoNet's ability to learn quickly on just 100 days worth of data, to significantly outperform most human forecasters, and do as well as the senior domain expert indicate that it is hardly solving a trivial problem. Flare forecasting apparently requires a delicate balancing of inputs that humans find very difficult.

Future Work

Many questions remain to be answered. We would like to examine the internal representations using relevance measures [Mozer in press] or something similar to see if there is any relationship to the rules used in THEO. Without those interpretations, connectionist networks cannot easily offer the help and explanation facilities of traditional expert systems that are a fallout of the rule-writing process. And how useful is the network regarding new knowledge, maintenance, training, unusual cases, and explanation? The particular representations formed, if they show a relationship not noticed before, may be of interest to solar physicists in understanding the nature of flare production.

Since the network used the same categories of data input

to THEO, these data were known to be significant. We need to ask if the network can eliminate redundant or unnecessary categories. This is an important issue for the practical development of expert systems. Conversely, a “choking auto-encoder”⁶ network that forces a non-redundant encoding of the input data could be used to determine the importance of any redundancy to the network's predictive ability. These investigations may reveal further differences between the two and three-layer models.

Both THEO and human experts have rules for dealing with interactions between sunspot regions to generate a whole-sun flare prediction⁷. How might this be done with a connectionist architecture? The problem is especially challenging, since this involves time and space dependent relationships and a variable number of regions. Some interesting connectionist architectures may be needed to do this.

Possibly most intriguing is the question of how the network will respond to the use of newly available data. The SEL is looking to obtain new categories of data about the magnetic and H-alpha wavelength characteristics of active regions. Traditional rule-based systems are a static codification of human knowledge at the time when the expert is interviewed. Learning systems such as

⁶ A three-layer network with the output constrained to the same values as the input, and the minimum number of hidden layer units needed to achieve a very low error.

⁷ *TheoNet's* current whole-sun prediction is, as described earlier, just a simple combination of region probabilities done “outside” the network.

TheoNet represent potentially more flexible and responsive solutions when domain knowledge is dynamic.

Final Remarks

Two particularly intriguing prospects are raised by the results of this research. The first is that if a connectionist network can perform the same task as a rule-based system, then a study of the internal representations constructed by the network may give insights to the “microstructure” of how knowledge can be represented in an implicit way. This is the same knowledge delineated at a higher level of description by the rules in an expert system. Our hope is that the two paradigms may eventually come to complement and support each other in cognitive science research.

The second prospect is more of an engineering nature and is that of dramatically easing the knowledge engineering job. Witness the current explosion of expert system technology in the marketplace today. Yet for all their glamor, expert systems have usually proved time consuming and expensive to implement. The “knowledge-engineering” step of interviewing experts and transferring their knowledge explicitly to rules that work successfully together has been the most difficult and expensive part, even with advanced knowledge representation languages and expert system shells.

TheoNet has shown that at least in this instance, a standard back-propagation network can quickly and automatically learn those necessary representations and interactions (rules?) needed to do the same sort of reasoning. Development of a functioning

prototype of THEO (one of the quicker developments of a usable expert system) required many man-months, while *TheoNet* needed only about a week using a simple simulator. In addition, THEO requires on the order of a minute to process a single prediction while the network requires only a few milliseconds, thus promising better performance under real-time conditions.

It is tempting to claim that connectionist networks represent the lazy man's way to expert systems. But we must look at the sort of problem first - failing to do this has been the cause of many disappointed expectations in expert systems and AI.

What is it that characterizes a problem that makes it a candidate for a connectionist solution? As with expert systems, it should not be amenable to traditional methods (which are usually cheaper and more efficient) and for which there is little or no formal description. Connectionist networks also need to have the conditions of the problem (input data) representable as a vector of discrete or continuous numbers. Though work is being done to get connectionist systems to do formal symbol manipulation, the most successful and straightforward implementations have been where there has been available large amounts of data in relatively few categories. Problems with soft constraints and approximate or probabilistic answers are a strength of connectionist approaches. The interesting problems will be the ones where the interrelationships between data categories and the problem solution are not known or poorly understood.

Solar flare prediction is a problem that represents the intersection of the families of problems appropriate for expert sys-

tems and for connectionist networks. This research has supported the belief mentioned in the introduction that it would be a good problem domain for examining the different approaches to problem solving. Similar domains need to be examined to determine if the ease and robustness of *TheoNet*'s solution can be reproduced. And to demonstrate the significance of connectionist approaches over traditional methods (i.e. regression), a problem domain needs to be found that requires a three-layer network.

Weather forecasting and other large multi-variate systems may represent analogous problems where we may further investigate the ability of connectionist methods to achieve that delicate balancing of the conditions of a problem that characterize the people we call "experts" and the programs we call "expert systems".

REFERENCES

- Bruce Buchanan and Edward Shortliffe, *Rule-Based Expert Systems*, Reading, MA, Addison-Wesley, 1984
- Randall Davis "Expert Systems: Where Are We? And Where Do We Go From Here?", *The AI Magazine*, Spring 1982
- J.L. Elman and David Zipser *Discovering the Hidden Structure of Speech* ICS Technical Report 8701, University of California, San Diego
- Richard Fozzard, Gary Bradshaw, Lou Ceci, "A Connectionist Expert System That Actually Works", in *Advances in Neural Information Processing Systems*, David Touretzky, Ed., Morgan-Kaufmann, 1989
- Stephen Gallant, "Connectionist Expert Systems", *Communications of the ACM*, February 1988
- Rodney Goodman, John Miller, Padhraic Smyth, "An Information Theoretic Approach to Rule-Based Connectionist Expert Systems", in *Advances in Neural Information Processing Systems*, David Touretzky, Ed., Morgan-Kaufmann, 1989
- Geoffrey Hinton and James Anderson, *Parallel Models of Associative Memory*, Hillsdale, NJ, Lawrence Erlbaum Associates, 1981
- Geoffrey Hinton "Learning Translation Invariant Recognition in a Massively Parallel Network", in *Proc. Conf. Parallel Architectures and Languages Europe*, Eindhoven, The Netherlands, June 1987
- Sidney Lehky and Terrence Sejnowski, "Neural Network Model for the Cortical Representation of Surface Curvature from Images of Shaded Surfaces", in *Sensory Processing*, J.S. Lund, ed., Oxford 1988
- Clayton Lewis and Joann Dennett "Joint CU/NOAA Study Predicts Events on the Sun with Artificial Intelligence Technology", *CUEngineering*, 1986
- Michael Mozer, "RAMBOT: A Connectionist Expert System That Learns by Example.", In M. Caudill & C. Butler (Eds.), *Proceedings of the IEEE First Annual International Conference on Neural Networks*, San Diego: IEEE Publishing Services, 1987
- Michael Mozer and Paul Smolensky, "Using relevance to reduce network size automatically", *Connection Science*, vol. 1, in press

- Allen Newell "Physical Symbol Systems", *Cognitive Science* 4:135-183, 1980
- David Rumelhart, Geoffrey Hinton, and R.J. Williams, in *Parallel Distributed Processing*. Rumelhart, McClelland, Eds., Cambridge, MA, Bradford books, 1986
- Terrence Sejnowski and C.R. Rosenberg, *NETtalk: A parallel network that learns to read aloud* Technical Report 86-01 Dept. of Electrical Engineering and Computer Science, Johns Hopkins University, Baltimore MD
- C. Sawyer, J.W. Warwick, and J.T.Dennett, *Solar Flare Prediction* Boulder, CO, Colorado Associated University Press, 1986
- Dave Shaw "Theophrastus", *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*, Denver 1989
- K.T. Spoehr and S.W. Lehmkuhle "Signal Detection Theory" in *Visual Information Processing*, Freeman 1982
- Donat G. Wentzel, *The Restless Sun*, Washington, DC, Smithsonian Institution, 1989
- David Zipser and D.E. Rabin "P3: A Parallel Network Simulating System", in *Parallel Distributed Processing*. Rumelhart, McClelland, Eds., Cambridge, MA, Bradford books, 1986

APPENDIX A

THEO DATABASE EXAMPLES

The following is a excerpt of the database file used by the THEO expert system to generate flare predictions.

```
region same  lat long extnt max area C H mag spots area large  c m1
m2  x class
ver
R regions record
  region  SESC region number
  same    region number previous solar rotation
  lat     solar latitude
  long    solar longitude
  extnt   extent in degrees
  max     maximum spot class (may be from previous rotation if same
> 0
  area    maximum area (may be from previous rotation if same > 0
C         historically complex (derived quantity)
H         y: thre region did NOT become class H during this rota-
tion
  mag     magnetic classification
  spots   spot count
  area    current area in millionths
  large   area of largest spot in the region
  c       count of C flares
  m1     count of M1 flares
  m2     count of M2-9 flares
  x       count of X flares
  class   McIntosh classification
F flares
  region  SESC region number
  pairs:  Time in HHMM Flare classification
          (times and subclass not known for 1969, constructed from
flare
          count)
```

This file can have anything typed up here, but nothing in data area below -

- also no blank lines.

```
region same  lat long extnt max area C H mag spots area large  c m1
m2  x class
ver
```

```
ddmmyy = 120269 julian = 25245
```

```
R 581 0 N13 W066 13 E 70 n y bet 11 70 10 0 0
0 0  ESI
1 0
R 585 0 N17 E013 2 A 2 - y a_p 1 2 2 0 0
0 0  AXX
1 0
R 586 0 S11 E077 2 A 2 - y a_p 1 2 2 0 0
```

```

0 0 AXX
1 0
ddmmyy = 130269 julian = 25246

F 581 1000 C3.0

R 581 0 N12 W077 8 E 70 y y bet 4 50 8 1 0
0 0 CSO
1 0
R 585 0 N17 E010 7 D 60 - y bet 17 60 10 0 0
0 0 DSO
1 0
R 586 0 S12 E066 6 C 20 - y bet 8 20 5 0 0
0 0 CSO
1 0
ddmmyy = 140269 julian = 25247

R 585 0 N17 W012 8 D 170 - y bet 24 170 15 0 0
0 0 DSO
1 0
R 586 0 S13 E052 7 D 80 - y bet 19 80 10 0 0
0 0 DAO
1 0
R 587 0 N09 W014 2 A 6 - y bet 3 6 1 0 0
0 0 AXX
1 0
R 588 0 N02 E053 1 A 1 - y a_p 1 1 1 0 0
0 0 AXX
1 0
ddmmyy = 150269 julian = 25248

R 585 0 N17 W026 8 D 170 - y bet 12 130 10 0 0
0 0 DAO
1 0
R 586 0 S13 E037 7 D 100 - y bet 11 100 4 0 0
0 0 DAO
1 0
R 587 0 N09 E030 1 A 6 - y a_p 1 1 1 0 0
0 0 AXX
1 0
R 589 0 N27 E039 1 A 1 - y a_p 1 1 1 0 0
0 0 AXX
1 0

```

The following is an excerpt from the database of THEO's generated predictions. Most use the simple default mode (though there is no way to tell if the interactive mode was used), and most do not have region interactions.

| date | time | reg1 | reg2 | intr | C | M | X |
|-------|------|------|------|------|----|----|---|
| days | hhmm | | | | | | |
| 32203 | 2400 | 4956 | 0 | 0 | 2 | 0 | 0 |
| 32203 | 2400 | 4957 | 0 | 0 | 80 | 8 | 2 |
| 32203 | 2400 | 4958 | 0 | 0 | 80 | 8 | 2 |
| 32203 | 2400 | 4959 | 0 | 0 | 2 | 0 | 0 |
| 32203 | 2400 | 4960 | 0 | 0 | 30 | 0 | 0 |
| 32203 | 2400 | 4961 | 0 | 0 | 2 | 0 | 0 |
| 32203 | 2400 | 4956 | 4957 | 1 | 90 | 8 | 2 |
| | | | | | | | |
| 32203 | 2400 | 4961 | 4957 | 2 | 99 | 10 | 2 |
| 32203 | 2400 | -2 | -1 | -1 | 99 | 10 | 2 |
| | | | | | | | |
| 32203 | 2400 | 0 | 0 | 0 | 99 | 17 | 4 |

Note that the whole sun forecast used only the compound interaction (99 10 2) and regions 4958(80 8 2), 4959(-), and 4960(-)=(99 17 4)

APPENDIX B

SELECTED PERFORMANCE DATA

| size | C-reg | M-reg | X-reg | C-wh | M-wh | X-wh |
|------|-------|-------|-------|------|------|------|
| 25 | 72 | 70 | 81 | 76 | 72 | 70 |
| 25 | | | | 72 | 67 | 56 |
| 25 | 75 | 76 | 90 | 77 | 78 | 82 |
| 25 | 72 | 73 | 80 | 77 | 74 | 63 |
| 50 | 76 | 74 | 77 | 75 | 70 | 71 |
| 50 | | | | 76 | 65 | 63 |
| 50 | 79 | 77 | 82 | 77 | 75 | 74 |
| 100 | 81 | 82 | 84 | 79 | 79 | 83 |
| 100 | 81 | 85 | 93 | 78 | 78 | 76 |
| 100 | 81 | 86 | 94 | 79 | 79 | 81 |
| 300 | 82 | 85 | 89 | 79 | 80 | 79 |
| 1625 | 82 | 85 | 84 | 81 | 81 | 85 |
| 1625 | 82 | 85 | 87 | 80 | 80 | 82 |
| 1625 | 80 | 84 | 88 | 82 | 79 | 84 |
| 1625 | 78 | 84 | 91 | 80 | 80 | 84 |
| 1625 | 82 | 84 | 92 | 82 | 82 | 86 |

| layer/train | C-reg | M-reg | X-reg | C-wh | M-wh | X-wh |
|-------------|-------|-------|-------|------|------|------|
| 2/all | 80 | 68 | 84 | 80 | 76 | 77 |
| 3/all | 80 | 84 | 88 | 80 | 80 | 84 |
| 2/78 | 70 | 61 | 31 | 78 | 65 | 48 |
| 3/78 | 81 | 86 | 92 | 79 | 81 | 85 |
| 2/onset | 80 | 70 | 28 | 79 | 76 | 46 |
| 3/onset | 82 | 86 | 86 | 80 | 81 | 82 |
| 2/min | 68 | 75 | 63 | 75 | 74 | 58 |
| 3/min | 78 | 80 | 90 | 78 | 76 | 83 |
| 2/distrib | 80 | 68 | 28 | 78 | 69 | 48 |
| 3/distrib | 81 | 85 | 93 | 78 | 78 | 76 |
| 2/all | 80 | 68 | 26 | 80 | 76 | 45 |
| 3/onset | 81 | 86 | 94 | 78 | 80 | 82 |
| 3/78 | 82 | 86 | 94 | 80 | 82 | 85 |
| 2/78 | 81 | 65 | 27 | 76 | 62 | 32 |
| 2/peak | 72 | 70 | 22 | 79 | 76 | 48 |
| 2/all | 80 | 67 | 45 | 80 | 77 | 44 |
| 3/peak | 81 | 85 | 68 | 80 | 79 | 74 |
| 3/all | 82 | 85 | 87 | 80 | 80 | 82 |
| 3/all | 80 | 84 | 88 | 82 | 79 | 84 |
| 3/all | 78 | 84 | 91 | 80 | 80 | 84 |
| 3/all | 82 | 84 | 92 | 82 | 82 | 86 |

| method | M-whole |
|---------------|---------|
| THEO default | 67 |
| TheoNet | 68 |
| TheoNet | 67 |
| 7-day avg | 66 |
| domain expert | 72 |
| SESC | 65 |
| THEO BASS | 84 |
| 2-layer Net | 65 |

| method | C-wh | M-wh | X-wh |
|---------------|------|------|------|
| THEO whole | 80 | 85 | 87 |
| TheoNet whole | 81 | 81 | 85 |

| #hiddens | C-reg | M-reg | X-reg | C-wh | M-wh | X-wh |
|----------|-------|-------|-------|------|------|------|
| 0 | 80 | 68 | 84 | 80 | 76 | 77 |
| 1 | 77 | 83 | 87 | 81 | 80 | 77 |
| 2 | 80 | 87 | 95 | 80 | 81 | 83 |
| 3 | 0 | 0 | 0 | | | |
| 4 | | | | | | |
| 5 | 82 | 85 | 84 | 81 | 81 | 85 |
| 6 | | | | | | |
| 7 | 80 | 84 | 88 | 82 | 79 | 84 |
| 8 | | | | | | |
| 9 | | | | | | |
| 10 | 82 | 84 | 92 | 82 | 82 | 86 |

| solar phase | C-reg | M-reg | X-reg | C-wh | M-wh | X-wh |
|-------------|-------|-------|-------|------|------|------|
| Minimum | 78 | 80 | 90 | 78 | 76 | 83 |
| Onset | 82 | 86 | 86 | 80 | 81 | 82 |
| Peak | 81 | 85 | 68 | 80 | 79 | 74 |
| distributed | 81 | 82 | 84 | 79 | 79 | 83 |
| M/O/P | 80 | 86 | 90 | 79 | 79 | 73 |
| distr.*3 | 82 | 85 | 89 | 79 | 80 | 79 |

| spacing | C-reg | M-reg | X-reg | C-wh | M-wh | X-wh |
|---------|-------|-------|-------|------|------|------|
| 1 | 82 | 85 | 84 | 81 | 81 | 85 |
| 7 | 82 | 85 | 87 | 80 | 80 | 82 |
| 14 | 82 | 85 | 81 | 80 | 79 | 83 |
| 21 | 83 | 85 | 85 | 86 | 83 | 87 |
| 28 | 82 | 86 | 56 | 80 | 80 | 82 |

APPENDIX C

SOURCE CODE

The following are complete listings of the P3 simulator model description and associated LISP functions used to create *TheoNet*.


```

;;; -*- Syntax: Zetalisp; Package: P3; Mode: Lisp; Base: 10; Lowercase: T; Default-character-style: (:fix
:roman :normal); -*-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Description:
;;;   TheoNet: Backprop network to learn solar flare forecasting as done by
;;;   the automatic mode of THEO expert system.
;;;
;;;   Theonet-whole has whole sun predictions in it.
;;;   Theonet2 has been reorganized to read in all data at once,
;;;   and then allow interactive control of test/train range
;;;   and masked/spaced protocols. Also add p3 init to random
;;;   weights and fix some bugs in a-prime generation.
;;;   Theonet3 adds region performance checks of THEO's corresponding
;;;   predictions. This adds more performance units
;;;   and changes to the pattern unit (and GETALLDATA) to read
;;;   these predictions from VERIFY.THEO
;;;   Theonet4 adds interactive disabling of hidden units
;;;
;;;
;;;   Gradient descent supervised learning algorithm for a
;;;   strictly layered network of semi-linear units
;;;   with a squashing function consisting of the tanh function offset and
;;;   scaled to lie between 0 and 1. In this version
;;;   every weight is assumed to be modifiable.
;;;   There is currently no weight decay.
;;;
;;;   NOTE: this is for the new expanded THEO database
;;;
;;; Auxiliary files required:
;;;   the GETALLDATA function and the THEODATA.NEW database file
;;;   VERIFY.THEO holds THEO's predictions by julian
;;;   theo.aux
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;these are the data sets for the net. Comment out the ones not needed
;(PLAN CONSTANT data-pairs = (cl-user::getalldata "munch:>fozzard>theodata.test"))
(PLAN CONSTANT data-pairs = (cl-user::getalldata "munch:>fozzard>theodata.new"))

(PLAN CONSTANT layer-size-list = '(13. 10. 3.))
(PLAN CONSTANT learn-rate = 0.2)
(plan constant momentum-factor = .9)

;;;;;;;;;;;;;;;;

(PLAN CONSTANT nbr-patterns = (length data-pairs))
(PLAN CONSTANT nbr-layers = (length layer-size-list))
(PLAN CONSTANT nbr-units = (apply #' + layer-size-list))
(PLAN CONSTANT nbr-input-units = (nth 0 layer-size-list))
(PLAN CONSTANT nbr-output-units = (nth (1- nbr-layers) layer-size-list))
(PLAN CONSTANT nbr-intermediate-units =
  (- nbr-units nbr-input-units nbr-output-units))
(PLAN CONSTANT layer-boundary-list = (make-layer-boundary-list layer-size-list))

(PLAN CONSTANT start-julian = (first (first (first data-pairs))))
(PLAN CONSTANT stop-julian = (first (first (nth (1- nbr-patterns) data-pairs))))
(PLAN CONSTANT theo-start = (loop for i from 0 below nbr-patterns do
  (if (>= (first (fourth (nth i data-pairs))) 0)
    (return (first (first (nth i data-pairs))))
    stop-julian)))

(PLAN CONSTANT theo-end = (let ((last-julian theo-start))
  (loop for i from 0 below nbr-patterns do
    (let ((this-julian (first (first (nth i data-pairs)))))
      (if (and (> (first (fourth (nth i data-pairs))) 0)
              (>= this-julian theo-start))
          (setq last-julian this-julian) last-julian)
        finally (return last-julian))))

;;;;;;;;;;;;;;;;

(unit type last-error
  PARAMETERS
  last-error-at
  learning-cycles
  enabled
  INPUTS
  (student array output-units)

```

```

(teacher array output-units)
(enable)
METHOD
(cond ((minusp (read-unit-parameter enabled))
      (loop with e = (read-unit-parameter learning-cycles)
            for i from 0 below (input-dimension-n student 1)
            do (cond ((and (≥ (read-input (teacher i)) .5)
                          (< (read-input (student i)) .5))
                    (set-unit-parameter last-error-at e ))
                  ((and (≤ (read-input (teacher i)) .5)
                          (> (read-input (student i)) .5))
                    (set-unit-parameter last-error-at e ))
                )
      )
      (SET-OUTPUT (prediction-error i)
                  (abs (- (READ-INPUT (teacher i))
                        (READ-INPUT (student i))))))
      )
(set-unit-parameter learning-cycles
  (+ 1
    (read-unit-parameter
     learning-cycles)))
))
(set-unit-parameter enabled (read-input enable))
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(UNIT TYPE control-unit
PARAMETERS
  currently-enabled          ; 0 ==> gen. pattern
                            ; i ==> propagate act'n at
                            ;         layer i-1
                            ; -i ==> propagate deltas at
                            ;         layer i-1

  learning-rate
  (do-test-run = 0)         ;initially dont do a test run
OUTPUTS
  (enable ARRAY i)         ; -1 means propagate act'n
                            ; >0 means learn using value
                            ;   as rate
  (do-testing-run LINE PARAMETERS
   (OUTPUT = nil))         ; to tell pattern-generator
                            ; whether to
                            ; use training (nil) or testing
                            ; (t) patterns
METHOD
(let* ((last-enabled (READ-UNIT-PARAMETER currently-enabled))
      (top-layer (1- (OUTPUT-DIMENSION-N enable 1)))
      (next-enabled (cond ((= last-enabled top-layer) (- top-layer))
                        ((= last-enabled -2) 0)
                        (t (1+ last-enabled))))
      )
  (SET-UNIT-PARAMETER currently-enabled next-enabled)
  (SET-OUTPUT (enable (abs last-enabled)) 0)
  (SET-OUTPUT do-testing-run (not (zerop (READ-UNIT-PARAMETER do-test-run))))
  (SET-OUTPUT (enable (abs next-enabled))
              (if (minusp next-enabled)
                  (READ-UNIT-PARAMETER learning-rate) -1))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(UNIT TYPE pattern-unit
PARAMETERS
  whole-c-so-far           ;#c-flares so far this julian
  whole-m-so-far           ;#m-flares so far this julian
  whole-x-so-far           ;#x-flares so far this julian
  (last-index = -1)
  data-patterns-array      ;held in array to hide from p3 popup
  number-of-patterns
  (iterations-per-epoch    ;for setting p3 blind iterations
   = (// (* 6 (1+ nbr-patterns)) 2)) ;//2 since spaced is t initially
  (this-julian = 0)
  (test-start = start-julian) ;<julian>
  (test-end = stop-julian)   ;<julian>
  (train-start = start-julian) ;<julian>
  (train-end = stop-julian)  ;<julian>
; (spaced-test-train = t)    ;<t,nil> whether to alternate

```



```

      (whole-m (if new-julian actual-m ;if new day, use this region
                (if (pluss actual-m) actual-m ;else if new flare
                    temp-whole-m-so-far))) ;else old
      (whole-x (if new-julian actual-x ;if new day, use this region
                (if (pluss actual-x) actual-x ;else if new flare
                    temp-whole-x-so-far))) ;else old
      (theo-c (let ((xx (/ (nth 0 theo-pattern) 100.0))) (if (< xx 0) -1 xx)))
      (theo-m (let ((xx (/ (nth 1 theo-pattern) 100.0))) (if (< xx 0) -1 xx)))
      (theo-x (let ((xx (/ (nth 2 theo-pattern) 100.0))) (if (< xx 0) -1 xx)))
    )
    ;end of let*

(loop for i from 0 below (OUTPUT-DIMENSION-N pattern 1)
      do (SET-OUTPUT (pattern i) (nth i input-pattern)))
(SET-OUTPUT (theo 0) theo-c)
(SET-OUTPUT (theo 1) theo-m)
(SET-OUTPUT (theo 2) theo-x)
(SET-UNIT-PARAMETER theo-c-prediction theo-c)
(SET-UNIT-PARAMETER theo-m-prediction theo-m)
(SET-UNIT-PARAMETER theo-x-prediction theo-x)
;
(SET-UNIT-PARAMETER spaced-granularity theo-c)
(SET-OUTPUT (teacher 0) actual-c)
(SET-OUTPUT (teacher 1) actual-m)
(SET-OUTPUT (teacher 2) actual-x)
(when new-julian ;set previous julian teacher values
  (SET-OUTPUT (teacher 3) temp-whole-c-so-far)
  (SET-OUTPUT (teacher 4) temp-whole-m-so-far)
  (SET-OUTPUT (teacher 5) temp-whole-x-so-far))
;set theo performance-unit teacher values to -1 if no THEO prediction avail
(SET-OUTPUT (teacher 6) (if (= (first theo-pattern) -1) -1 actual-c))
(SET-OUTPUT (teacher 7) (if (= (second theo-pattern) -1) -1 actual-m))
(SET-OUTPUT (teacher 8) (if (= (third theo-pattern) -1) -1 actual-x))
(if (> current-index index) ;end of epoch
    (let* ((patterns (1+ (- current-index index)))
           (iters (* 6 patterns)))
      (SET-UNIT-PARAMETER
        iterations-per-epoch (if spaced (/ iters 2) iters))
        (SET-UNIT-PARAMETER last-start-index (1- index))
        (SET-UNIT-PARAMETER theo-predictions-end (1- index))
      ))
  (SET-UNIT-PARAMETER last-index index)
  (if (= (READ-UNIT-PARAMETER last-start-index) -1) ;havent found one yet
      (SET-UNIT-PARAMETER last-start-index (1- index))) ;found it!
  ;
  (if (= (READ-UNIT-PARAMETER theo-predictions-end) -1) ;havent found one yet
      (SET-UNIT-PARAMETER theo-predictions-end (1- index))) ;found it!
  ;
  (SET-OUTPUT last-index index)
  (SET-OUTPUT current-julian julian)
  (SET-OUTPUT current-region region)
  (SET-UNIT-PARAMETER whole-c-so-far whole-c)
  (SET-UNIT-PARAMETER whole-m-so-far whole-m)
  (SET-UNIT-PARAMETER whole-x-so-far whole-x)
  (SET-UNIT-PARAMETER this-julian julian)
  ))))
))))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(UNIT TYPE input-layer-unit
  PARAMETERS
    activation
  INPUTS
    (enable)
    (pattern-input)
  OUTPUTS
    (activation-output)
  METHOD
    (cond ((minusp (READ-INPUT enable)) ; propagate activation
          (let ((activity (READ-INPUT pattern-input)))
            (SET-UNIT-PARAMETER activation activity)
            (SET-OUTPUT activation-output activity)
          )))
)

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(UNIT TYPE intermediate-layer-unit
  PARAMETERS

```



```

      last-dw
      dwt))))))))))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(UNIT TYPE output-layer-unit
  PARAMETERS
    activation
    delta
    bias
    momentum
    (weights-initialized = nil) ;allows random weights on p3 init
  INPUTS
    (enable)
    (do-testing-run) ; to tell unit whether to
                    ; use training (nil) or testing (t) patterns
    (activation-input
      TERMINAL PARAMETERS
        weight
        terminal-index
        last-dw)
    (teacher)
  OUTPUTS
    (delta-times-wt-output ARRAY i)
    (activation-output)
  METHOD
    (let ((rate (READ-INPUT enable)))
      (without-floating-underflow-traps ;stupid kludge...

        (when (not (READ-UNIT-PARAMETER weights-initialized))
          (SET-UNIT-PARAMETER bias (random-weights))
          (FOR-TERMINALS j OF INPUT activation-input
            (SET-TERMINAL-PARAMETER
              (activation-input TERMINAL j) weight (random-weights)))
          (SET-UNIT-PARAMETER weights-initialized t))

        (cond ((minusp rate) ; propagate activation
              (let (
                  (activity)
                  (net-input (READ-UNIT-PARAMETER bias))
                )
                (FOR-TERMINALS i OF INPUT activation-input
                  (incf net-input (* (READ-TERMINAL-PARAMETER
                    (activation-input TERMINAL i) weight)
                    (READ-INPUT
                    (activation-input TERMINAL i))))))
                (setq activity (squash net-input))
                (SET-UNIT-PARAMETER activation activity)
                (set-output activation-output activity)
              ))

        ;; (zerop rate)

        ((and (plussp rate) (not (READ-INPUT do-testing-run)))
          ;propagate delta & change weights
          (let ((activity (READ-UNIT-PARAMETER activation))
                (desired (READ-INPUT teacher))
                (delta-value)
                (a (read-unit-parameter momentum)))
            (setq delta-value (* (- desired activity)
                                (squash-prime activity)))
            (SET-UNIT-PARAMETER delta delta-value)
            (SET-UNIT-PARAMETER bias (+ (READ-UNIT-PARAMETER bias)
                                         (* rate delta-value)))
            (FOR-TERMINALS i OF INPUT activation-input
              (let* ((index (READ-TERMINAL-PARAMETER
                (activation-input TERMINAL i) terminal-index))
                    (wt (READ-TERMINAL-PARAMETER
                    (activation-input TERMINAL i) weight))
                    (ldwt (READ-TERMINAL-PARAMETER
                    (activation-input TERMINAL i) last-dw))
                    (pre-synaptic
                    (READ-INPUT (activation-input TERMINAL i)))
                    (dwt (+ (* rate delta-value pre-synaptic) (* a ldwt))))

```

```

(SET-OUTPUT (delta-times-wt-output index)
             (* delta-value wt))
(SET-TERMINAL-PARAMETER
 (activation-input TERMINAL i) weight
 (+ wt dwt ))
(SET-TERMINAL-PARAMETER (activation-input TERMINAL i)
                        last-dw
                        dwt)))))))))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(UNIT TYPE whole-sun-unit
PARAMETERS
  (dummy-activation = 0.0)                ;!@#$! p3 bug was zeroing this
  (activation = 0.0)                      ;!@#$! p3 bug was zeroing this
  (prediction = 0.0)
  (product-so-far = 0.0)                  ;PI(1-region-pred[i]) so far
  (this-julian = 0)
INPUTS
  (enable)                               ;same as output units
  (output-activation)                    ;from output-units
  (current-julian)                       ;from pattern-unit
  (current-region)                       ;from pattern-unit
OUTPUTS
  (prediction)                            ;to performance-units
METHOD
  (if (plusp (READ-INPUT enable))         ;if output-units passing delta
      (without-floating-underflow-traps  ;stupid kludge...
        (let* ((prod-so-far (READ-UNIT-PARAMETER product-so-far))
               (temp-pred (- 1.0 prod-so-far))
               (julian (READ-INPUT current-julian))
               (new-julian (not (equal julian (READ-UNIT-PARAMETER this-julian))))
               (region-pred (READ-INPUT output-activation))
               (product (if new-julian (- 1.0 region-pred)
                             (* prod-so-far (- 1.0 region-pred))))
               )
          (SET-UNIT-PARAMETER this-julian julian)
          (SET-UNIT-PARAMETER product-so-far product)
          (when new-julian ;set prev julian prediction value
            (SET-OUTPUT prediction temp-pred)
            (SET-UNIT-PARAMETER prediction temp-pred))))))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(UNIT TYPE performance-unit ;for evaluating the performance of each output unit
; Unit's activity is compared against what actually
; occurred for different response biases to generate
; probabilities of hits & false alarms which in turn
; are graphed against each other; an area under the
; curve (a-prime) is calculated as a measure of the
; "signal detection" performance of the network.
; The last 3 units instantiated are for evaluating the
; performance of the whole-sun units.

```

```

PARAMETERS
  a-prime
  (response-bias-initialized = nil)
  (bias-initialized = nil)
  (sum-flares = 0.0)
  (sum-no-flares = 0.0)
  (desired-teacher = 0)                ;from teacher input
  (previous-index = 999999)            ;for determining end of epoch
  (flares-per-epoch = 0)               ;max value of sum-flares
  (no-flares-per-epoch = 0)           ;max value of sum-no-flares
INPUTS
  (enable)                             ;will run at same time that pattern unit
; is running (so teacher is for last patterns)
  (teacher)                             ;gives actual flare occurrence info
  (do-testing-run)                      ; to tell unit whether to
; use training (nil) or testing (t) patterns
  (last-index)                          ;to tell units when epochs are over
  (activation-input)                    ;from corresponding output unit

```



```

(dummy-response ARRAY i                ;stupid kludge for LISP/P3 bug
  TERMINAL PARAMETERS (bias = 0.0))
(response ARRAY i                       ;response biases
  TERMINAL PARAMETERS (bias = 0.0))
(dummy-sum-hits ARRAY i                 ;stupid kludge for LISP/P3 bug
  TERMINAL PARAMETERS (bias = 0.0))
(sum-hits ARRAY i                      ;for each response bias
  TERMINAL PARAMETERS (bias = 0.0))
(dummy-sum-false-alarms ARRAY i        ;stupid kludge for LISP/P3 bug
  TERMINAL PARAMETERS (bias = 0.0))
(sum-false-alarms ARRAY i             ;for each response bias
  TERMINAL PARAMETERS (bias = 0.0))
(dummy-p-hits ARRAY i                  ;stupid kludge for LISP/P3 bug
  TERMINAL PARAMETERS (bias = 0.0))
(p-hits ARRAY i                       ;for each response bias
  TERMINAL PARAMETERS (bias = 0.0))
(dummy-p-false-alarms ARRAY i         ;stupid kludge for LISP/P3 bug
  TERMINAL PARAMETERS (bias = 0.0))
(p-false-alarms ARRAY i              ;for each response bias
  TERMINAL PARAMETERS (bias = 0.0))

```

OUTPUTS

METHOD

```

(let (
  (loc-sum-flares (READ-UNIT-PARAMETER sum-flares))
  (loc-sum-no-flares (READ-UNIT-PARAMETER sum-no-flares))
  (response-init (READ-UNIT-PARAMETER bias-initialized))
  (num-biases (INPUT-DIMENSION-N response 1))
  (a-sum 0.0) ;for doing a-prime
  (activity (READ-INPUT activation-input)) ;get it's activity
  (actual (READ-INPUT teacher)) ;get actual flare occurrence
  (index (READ-INPUT last-index)))

  (without-floating-underflow-traps ;stupid kludge...

    ;setup of the various response-biases to be used
    (if (not response-init)
      (let ((bias-step (do-divide
        1.0 (1- (INPUT-DIMENSION-N response 1)))))
        (loop
          with next-bias = 0.0
          for j from 1 below num-biases do ;all inputs,
            (SET-TERMINAL-PARAMETER (response j) bias
              (incf next-bias bias-step))
          finally
            (SET-TERMINAL-PARAMETER (response 0) bias 0.0)
            (SET-TERMINAL-PARAMETER (response 100) bias 1.0)
            (SET-UNIT-PARAMETER bias-initialized t)
            (SET-UNIT-PARAMETER previous-index 999999))))

      ;actual performance evaluation
      (cond ((and response-init ;if init done earlier,
        (minusp (READ-INPUT enable))) ;if starting patterns,
        (SET-UNIT-PARAMETER desired-teacher actual)
        (cond ((> (READ-UNIT-PARAMETER previous-index) ;at end of epoch
          index) ; since index drops only at end
          (loop for i from 0 below num-biases do
            (SET-TERMINAL-PARAMETER
              (p-hits i) bias
              (do-divide
                (READ-TERMINAL-PARAMETER (sum-hits i) bias)
                loc-sum-flares))
            (SET-TERMINAL-PARAMETER (sum-hits i) bias 0.0)
            (SET-TERMINAL-PARAMETER
              (p-false-alarms i) bias
              (do-divide
                (READ-TERMINAL-PARAMETER (sum-false-alarms i) bias)
                loc-sum-no-flares))
            (SET-TERMINAL-PARAMETER (sum-false-alarms i) bias 0.0)
          ) ;end loop
          (loop for j from 1 below num-biases do
            (setq a-sum
              (+ a-sum ;sum each vertical slice
                ;=rectangle+triangle
                (do-divide
                  (* (- (READ-TERMINAL-PARAMETER

```

```

                (p-false-alarms (1- j)) bias)
            (READ-TERMINAL-PARAMETER
             (p-false-alarms j) bias))
        (+ (READ-TERMINAL-PARAMETER
            (p-hits j) bias)
           (READ-TERMINAL-PARAMETER
            (p-hits (1- j)) bias)))
        2.0) ;div by 2 gives slice area
    ))
) ;end loop
(incf a-sum ;add in from max p(FA) up to 1.0
 (- 1.0 (READ-TERMINAL-PARAMETER (p-false-alarms 0) bias)))
(SET-UNIT-PARAMETER flares-per-epoch loc-sum-flares)
(SET-UNIT-PARAMETER no-flares-per-epoch loc-sum-no-flares)
(SET-UNIT-PARAMETER sum-flares 0.0)
(setq loc-sum-flares 0.0)
(SET-UNIT-PARAMETER sum-no-flares 0.0)
(setq loc-sum-no-flares 0.0)
(SET-UNIT-PARAMETER a-prime a-sum)

)) ;end cond at start of epoch
;for all of epoch
(cond ((> actual 0) ;if we have a flare,
      (SET-UNIT-PARAMETER sum-flares
        (1+ loc-sum-flares))
      (loop for k from 0 below num-biases do
            ;for each bias,
            ;if activity > this response-bias,
            ;increment corresponding sum-hits
            (if (>= activity
                (READ-TERMINAL-PARAMETER (response k) bias))
                (SET-TERMINAL-PARAMETER
                 (sum-hits k) bias
                 (1+ (READ-TERMINAL-PARAMETER
                     (sum-hits k) bias))))))
      ((= actual 0) ;else we dont have a flare
      (SET-UNIT-PARAMETER sum-no-flares
        (1+ loc-sum-no-flares))
      (loop for m from 0 below num-biases do
            ;for each bias,
            ;if activity > this response-bias,
            ;increment corresponding
            ;sum-false-alarms
            (if (>= activity
                (READ-TERMINAL-PARAMETER (response m) bias))
                (let ((sum-false-so-far (READ-TERMINAL-PARAMETER
                    (sum-false-alarms m) bias)))
                    (SET-TERMINAL-PARAMETER
                     (sum-false-alarms m) bias
                     (1+ sum-false-so-far)))
                ))) ;end all of epoch
      ;note that if actual < 0, then ignore this prediction. This would
      ;happen when there is no THEO prediction (teacher i) = -1.
      (SET-UNIT-PARAMETER previous-index index)
    )) ;end if doing patterns
  ))) ;end performance unit
)))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; UNIT instantiations
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(unit last-error
  of type last-error
  at (@ 2 -4 0)
  inputs
  (student array (typ 0 (1- nbr-output-units))
    lines at (@ typ 0 0))
  (teacher array (typ 0 (1- nbr-output-units))
    lines at (@ typ 0 0))
)

```

```

(UNIT control
  OF TYPE control-unit
  AT (@ 0 -2 0)
)

```

```

INITIALIZE
  (currently-enabled = 0)
  (learning-rate = learn-rate)
OUTPUTS
  (enable ARRAY (i 0 nbr-layers)
    LINES INITIALIZE (OUTPUT = (if (zerop i) -1 0)))
)

(UNIT pattern-generator
  OF TYPE pattern-unit
  AT (@ 1 -2 0)
  INITIALIZE
    (data-patterns-array =
      (cl-user::make-array 1 :initial-contents (list data-pairs)))
      ;this keeps p3 from trying to display the giant list in popup
    (number-of-patterns = nbr-patterns)
    (theo-predictions-end = theo-end)
  OUTPUTS
    (pattern ARRAY (i 0 (1- nbr-input-units)))
    (theo ARRAY (i 0 (1- nbr-output-units)))
    (teacher ARRAY (i 0 (1- (* 3 nbr-output-units))))
    LINES INITIALIZE (OUTPUT = 0)
  )
)

(UNIT input-unit
  ARRAY (input-unit-index 0 (1- nbr-input-units))
  OF TYPE input-layer-unit
  AT (@ input-unit-index 0 0)
)

(UNIT intermediate-unit
  ARRAY (int-unit-index 0 (1- nbr-intermediate-units))
  OF TYPE intermediate-layer-unit
  AT (@ (within-layer-index int-unit-index layer-boundary-list)
    (layer-number int-unit-index layer-boundary-list) 0)
  INITIALIZE
    (bias = (random-weights))
    (momentum = momentum-factor)
    (enabled = (if (< int-unit-index 5) 1 0)) ;turn on 1st 5 units
  INPUTS
    (delta-times-wt-input
      ARRAY (i 0 (1- (nbr-in-next-higher-layer
        int-unit-index layer-size-list
        layer-boundary-list))))
  OUTPUTS
    (delta-times-wt-output
      ARRAY (i 0 (1- (nbr-in-next-lower-layer
        int-unit-index layer-size-list
        layer-boundary-list))))
  )
)

(UNIT output-unit
  ARRAY (output-unit-index 0 (1- nbr-output-units))
  OF TYPE output-layer-unit
  AT (@ output-unit-index (1- nbr-layers) 0)
  INITIALIZE
    (bias = (random-weights))
    (momentum = momentum-factor)
  OUTPUTS
    (delta-times-wt-output
      ARRAY (i 0 (1- (nth (- nbr-layers 2) layer-size-list))))
  )
)

(UNIT whole-sun
  ARRAY (whole-sun-index 0 (1- nbr-output-units))
  OF TYPE whole-sun-unit
  AT (@ (+ 3 whole-sun-index) nbr-layers 0) ;put above output units
  INPUTS
    (output-activation)
;   (current-julian = 0)
)

(UNIT performance
;one each for C, M, X, and whole-sun C, M, X; and THEO C, M, X
  ARRAY (performance-unit-index 0 (1- (* 3 nbr-output-units)))
  OF TYPE performance-unit

```

```

AT (@ performance-unit-index (1+ nbr-layers) 0) ;put above whole-sun units
INITIALIZE
(a-prime = 0.5)
INPUTS
(dummy-response ARRAY (i 0 100))
(response ARRAY (i 0 100))
(dummy-sum-hits ARRAY (i 0 100))
(sum-hits ARRAY (i 0 100))
(dummy-sum-false-alarms ARRAY (i 0 100))
(sum-false-alarms ARRAY (i 0 100))
(dummy-p-hits ARRAY (i 0 100))
(p-hits ARRAY (i 0 100))
(dummy-p-false-alarms ARRAY (i 0 100))
(p-false-alarms ARRAY (i 0 100))
)

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; CONNECTIONS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

;last-error:

```

```

(CONNECT UNIT control OUTPUT enable nbr-layers
         TO UNIT last-error INPUT enable)

```

```

;pattern-generator:

```

```

(CONNECT UNIT control OUTPUT enable 0
         TO UNIT pattern-generator INPUT enable)

```

```

(CONNECT UNIT control OUTPUT do-testing-run
         TO UNIT pattern-generator INPUT do-testing-run)

```

```

;input-unit:

```

```

(loop for i from 0 below nbr-input-units
      do (CONNECT UNIT control OUTPUT enable 1
              TO UNIT input-unit i INPUT enable))

```

```

(loop for i from 0 below nbr-input-units
      do (CONNECT UNIT pattern-generator OUTPUT pattern i
              TO UNIT input-unit i INPUT pattern-input))

```

```

;whole-sun unit:

```

```

(loop for i from 0 below nbr-output-units
      do (CONNECT UNIT control OUTPUT enable nbr-layers ;run w/output passing delta
              TO UNIT whole-sun i INPUT enable)
      do (CONNECT UNIT pattern-generator OUTPUT current-julian
              TO UNIT whole-sun i INPUT current-julian)
      do (CONNECT UNIT pattern-generator OUTPUT current-region
              TO UNIT whole-sun i INPUT current-region)
      do (CONNECT UNIT output-unit i
              OUTPUT activation-output
              TO UNIT whole-sun i
              INPUT output-activation))

```

```

;performance:

```

```

(loop for i from 0 below (* 3 nbr-output-units) ;to performance units
      ;connect enable to have
      ;these run at same time as
      ;the pattern units.
      do (CONNECT UNIT control OUTPUT enable 0
              TO UNIT performance i INPUT enable)
      do (CONNECT UNIT control OUTPUT do-testing-run
              TO UNIT performance i INPUT do-testing-run)
      do (CONNECT UNIT pattern-generator OUTPUT teacher i
              TO UNIT performance i INPUT teacher)
      do (CONNECT UNIT pattern-generator OUTPUT last-index

```

```

        TO UNIT performance i INPUT last-index)
do (cond
  (>= i (* 2 nbr-output-units)) ;THEO performance units
  (CONNECT UNIT pattern-generator OUTPUT theo (- i 6)
    TO UNIT performance i INPUT activation-input))
  (>= i nbr-output-units) ;whole-sun performance-units
  (CONNECT UNIT whole-sun (- i 3) OUTPUT prediction
    TO UNIT performance i INPUT activation-input))
(t
  (CONNECT UNIT output-unit i
    OUTPUT activation-output
    TO UNIT performance i
    INPUT activation-input)))

;output-unit:

(loop for i from 0 below nbr-output-units
  do (CONNECT UNIT control OUTPUT enable nbr-layers
    TO UNIT output-unit i INPUT enable)
  do (CONNECT UNIT pattern-generator
    OUTPUT teacher i
    TO UNIT last-error
    INPUT teacher i)
  do (CONNECT UNIT output-unit i
    OUTPUT activation-output
    TO UNIT last-error
    INPUT student i)
  )

(loop for i from 0 below nbr-intermediate-units
  do (CONNECT UNIT control OUTPUT do-testing-run
    TO UNIT intermediate-unit i INPUT do-testing-run)
  do (CONNECT UNIT control OUTPUT enable
    (1+ (layer-number i layer-boundary-list))
    TO UNIT intermediate-unit i INPUT enable))

(loop for i from 0 below nbr-output-units
  do (CONNECT UNIT control OUTPUT do-testing-run
    TO UNIT output-unit i INPUT do-testing-run)
  do (CONNECT UNIT pattern-generator OUTPUT teacher i
    TO UNIT output-unit i INPUT teacher)
  )

(loop for i from 0 below nbr-input-units
  do (loop for j from 0 below (nth 1 layer-boundary-list)
    do (CONNECT UNIT input-unit i OUTPUT activation-output
      TO UNIT intermediate-unit j INPUT activation-input
      INITIALIZE
        (weight = (random-weights))
        (terminal-index = i)
      )))

(loop for layer-of-ith from 1 below (- nbr-layers 2)
  do (loop for i from (nth (1- layer-of-ith) layer-boundary-list)
    below (nth layer-of-ith layer-boundary-list)
    for k from 0
    do (loop for j from (nth layer-of-ith layer-boundary-list)
      below (nth (1+ layer-of-ith) layer-boundary-list)
      for l from 0
      do (CONNECT UNIT intermediate-unit i
        OUTPUT activation-output
        TO UNIT intermediate-unit j
        INPUT activation-input
        INITIALIZE
          (weight = (random-weights))
          (terminal-index = k))
      do (CONNECT UNIT intermediate-unit j
        OUTPUT delta-times-wt-output k
        TO UNIT intermediate-unit i
        INPUT delta-times-wt-input l)
      )))

(loop for i from (nth (- nbr-layers 3) layer-boundary-list)
  below (nth (- nbr-layers 2) layer-boundary-list)
  for k from 0

```

```
do (loop for j from 0 below nbr-output-units
  do (CONNECT UNIT intermediate-unit i OUTPUT activation-output
      TO UNIT output-unit j INPUT activation-input
      INITIALIZE
        (weight = (random-weights))
        (terminal-index = k))
  do (CONNECT UNIT output-unit j OUTPUT delta-times-wt-output k
      TO UNIT intermediate-unit i
      INPUT delta-times-wt-input j)
))
```



```

((and (= pred-julian julian) ;same julian
      (< pred-reg1 region)) ;not yet to current
 (setq pred-list (parse-pred-line ;get new line
                    (read-line predfile nil "eof"))))
((and (= pred-julian julian) ;same julian
      (= pred-reg1 region) ;same region
      (/= pred-reg2 0)) ;but not simple region
 (setq pred-list (parse-pred-line ;get new line
                    (read-line predfile nil "eof"))))
((and (= pred-julian julian) ;same julian
      (= pred-reg1 region) ;same region
      (= pred-reg2 0)) ;simple region pred.
 (setq pred-list (parse-pred-line ;get new line
                    (read-line predfile nil "eof")))
 (return this-pred-list)))) ;got a matching one!
(data-pair (parse-line line matching-pred-list yesterday today))
(when (not (null data-pair))
  (setq data-pairs
        (append data-pairs
                (list data-pair))))
)
))))))

```

```

;;;-----
(defun parse-pred-line (pred-line) ;parse a line from the THEO predictions file
;;;-----
(with-input-from-string (pred-str pred-line)
  (let ((julian (read pred-str nil "eol"))
        (time (read pred-str nil "eol"))
        (reg1 (read pred-str nil "eol"))
        (reg2 (read pred-str nil "eol"))
        (intr (read pred-str nil "eol"))
        (Tc (read pred-str nil "eol"))
        (Tm (read pred-str nil "eol"))
        (Tx (read pred-str nil "eol")))
    (list julian time reg1 reg2 intr Tc Tm Tx)))

;;;-----
(defun parse-line (line pred-list yesterday today) ;return an encoded input pattern
;list of the form:
;((julian region) (inputs) (c m x) (Tc Tm Tx))
;region=0 if whole sun, -1 if whole sun NA
;Tc...=-1 if Theo predictions Not Avail.
;also appends current line to list
;of today's lines
;;;-----

(with-input-from-string (line-str line)
  (let ((*package* (find-package "cl-user"))) ;to allow this to run in p3:
    (when (equal 'R (read line-str nil "eol")) ;skip "F"
      (terpri) (write-line line)
      (write-char #\.) ;indicate a region example
      (let* ((julian (car today))
             (region (read line-str nil "eol"))
             (same (read line-str nil "eol")) ;not used
             (lat (read line-str nil "eol")) ;not used
             (long (read line-str nil "eol")) ;not used
             (extnt (read line-str nil "eol")) ;not used
             (max (read line-str nil "eol")) ;not used
             (maxarea (read line-str nil "eol")) ;not used
             (C (if (equal 'y (read line-str nil "eol"))
                    1 0)) ;1 if "y"
             (H (if (equal 'y (read line-str nil "eol"))
                    0 1)) ;0 if "y"
             (mag (read line-str nil "eol")) ;not used
             (spots (read line-str nil "eol")) ;not used
             (area (if (> (read line-str nil "eol") 700)
                      1 0)) ;1 if >700
             (large (read line-str nil "eol")) ;not used
             (c-class (read line-str nil "eol"))
             (m1-class (read line-str nil "eol"))
             (m2-class (read line-str nil "eol"))
             (x-class (read line-str nil "eol"))
             (class (string (read line-str nil "eol")))
             (zurich (case (char class 0) ;Zurich letter

```



```

      (#\A '(0 0 0))
      (#\B '(0 0 1))
      (#\C '(0 1 0))
      (#\D '(0 1 1))
      (#\E '(1 0 0))
      (#\F '(1 0 1))
      (#\H '(1 1 0))
      (t (print "Bad zurich class ") (prin1 (char class 0))
         (terpri) '(0 0 0)))
(size (case (char class 1) ;spot size
      (#\X '(0 0 0))
      (#\R '(0 0 1))
      (#\S '(0 1 0))
      (#\A '(0 1 1))
      (#\H '(1 0 0))
      (#\K '(1 0 1))
      (t (print "Bad size class ") (prin1 (char class 1))
         (terpri) '(0 0 0))))
(dist (case (char class 2) ;distribution
      (#\X '(0 0))
      (#\O '(0 1))
      (#\I '(1 0))
      (#\C '(1 1))
      (t (print "Bad dist class ") (prin1 (char class 2))
         (terpri) '(0 0))))
(Tc (sixth pred-list))
(Tm (seventh pred-list))
(Tx (eighth pred-list))

(if (/= Tc -1) (write-char #\,) ;indicate THEO prediction available
(nconc today (list (list region c-class m1-class m2-class x-class)))
(list
 (list julian region)
 (append zurich size dist
         (cond ((null yesterday) '(0 0)) ;if no prev day data
               (t ;there is prev day data
                  (loop with i = 0 do ;for all prev day regions
                        (incf i) ;get next region data
                        (let* ((prev-day (nth i yesterday))
                               (prev-region (nth 0 prev-day))
                               (prev-c (nth 1 prev-day))
                               (prev-m1 (nth 2 prev-day))
                               (prev-m2 (nth 3 prev-day))
                               (prev-x (nth 4 prev-day)))
                          (cond ((null prev-day) (return '(0 0)))
                                ((equal region prev-region)
                                 (cond ((> prev-x 0) (return '(1 0)))
                                       ((> prev-m2 0) (return '(1 0)))
                                       ((> prev-m1 1) (return '(1 0)))
                                       ((= prev-m1 1) (return '(0 1)))
                                       ((> prev-x 0) (return '(0 0))))
                                )))
                )))
      (list C H area))
(list (if (> c-class 0) 1 0) ;any c flare
      (if (> (+ m1-class m2-class) 0) 1 0) ;any m flare
      (if (> x-class 0) 1 0) ;any x flare
(list Tc Tm Tx) ;THEO predictions
))))))

```



```

;;; -*- Package:P3 ; Mode: Lisp; Base:10 ; Lowercase:Yes ; -*-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Description: THEO.AUX
;;;
;;; Auxiliary file containing lisp functions used by TheoNet
;;;
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; These are used to determine the size of certain unit arrays and input & output
;;; arrays. They are also used in connecting units according to their levels.

(defun make-layer-boundary-list (layer-size-list)
  (loop for i from 1 below (length layer-size-list)
        with bdry-list = (make-list (length layer-size-list) :initial-value 0)
        do (setf (nth i bdry-list) (+ (nth i layer-size-list) (nth (1- i) bdry-list)))
        finally (return bdry-list)
        ))

(defun layer-number (int-unit-index layer-boundary-list)
  (loop for nbr-thru-ith in layer-boundary-list
        for layer from 0
        until (> nbr-thru-ith int-unit-index)
        finally (return layer)))

(defun within-layer-index (int-unit-index layer-boundary-list)
  (- int-unit-index
     (nth (1- (layer-number int-unit-index layer-boundary-list)) layer-boundary-list)))

(defun nbr-in-next-higher-layer (int-unit-index layer-size-list layer-boundary-list)
  (nth (1+ (layer-number int-unit-index layer-boundary-list)) layer-size-list))

(defun nbr-in-next-lower-layer (int-unit-index layer-size-list layer-boundary-list)
  (nth (1- (layer-number int-unit-index layer-boundary-list)) layer-size-list))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; This function is used to initialize each weight and bias in the network

(defun random-weights ()
  (si:random-in-range -1.0 1.0))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; These functions compute the squashing function and its derivative

(defun squash (net-input)
  (// 1.0 (+ 1.0 (exp (- net-input)))))

(defun squash-prime (output)
  (* output (- 1.0 output)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; auxiliary lisp functions used by the pattern generator unit

;;; NOTE: used only by older versions of theonet (theo.p3 and theonet-old.p3)

(defvar class-alist '((A 0 0 0)
                    (B 0 0 1)
                    (C 0 1 0)
                    (D 0 1 1)
                    (E 1 0 0)
                    (F 1 0 1)
                    (H 1 1 0)))

(defvar size-alist '((X 0 0 0)
                   (R 0 0 1)
                   (S 0 1 0)
                   (A 0 1 1)
                   (H 1 0 0)
                   (K 1 0 1)))

(defvar distribution-alist '((X 0 0) (O 0 1) (I 1 0) (C 1 1)))

(defvar activity-alist '((1 0) (2 1)))

(defvar evolution-alist '((1 0 0) (2 0 1) (3 1 0)))

```

```
(defvar previous-day-alist '((1 0 0) (2 0 1) (3 1 0)))

(defvar complex-alist '((1 0) (2 1)))

(defvar complex-this-pass-alist '((1 0) (2 1)))

(defvar spot-area-alist '((1 0) (2 1)))

(defvar region-area-alist '((1 0) (2 1)))

(defun encode (value alist)
  (append (cdr (assoc value alist))
          nil))

(defun translate (pat)
  (nconc (encode (first pat) class-alist)
         (encode (second pat) size-alist)
         (encode (third pat) distribution-alist)
         (encode (fourth pat) activity-alist)
         (encode (fifth pat) evolution-alist)
         (encode (sixth pat) previous-day-alist)
         (encode (seventh pat) complex-alist)
         (encode (nth 7 pat) complex-this-pass-alist)
         (encode (nth 8 pat) region-area-alist)
         (encode (nth 9 pat) spot-area-alist)))

(defun do-divide (argle bargle) ;kludge for bug in p3 (wont support / function in method)
  (if (= bargle 0) 0.0 (// argle bargle)))
```