

A SOFTWARE PACKAGE FOR NEURAL NETWORK APPLICATIONS DEVELOPMENT

Robert H. Baran
 Naval Surface Warfare Center
 White Oak (code N51)
 Silver Spring, MD 20903-5000

549-63

150519

P-10

ABSTRACT

Original Backprop (Version 1.2) is an MS-DOS package of four stand-alone C-language programs that enable users to develop neural network solutions to a variety of practical problems. *Original Backprop* generates three-layer, feed-forward (series-coupled) networks which map fixed-length input vectors into fixed-length output vectors through an intermediate ("hidden") layer of binary threshold units. Version 1.2 can handle up to 200 input vectors at a time, each having up to 128 real-valued components. The first subprogram, TSET, appends a number (up to 16) of classification bits to each input, thus creating a *training set* of input-output pairs. The second subprogram, BACKPROP, creates a trilayer network to do the prescribed mapping and modifies the weights of its connections incrementally until the training set is learned. The learning algorithm is the "back-propagating error correction procedure" first described by F. Rosenblatt in 1961. The third subprogram, VIEWNET, lets the trained network be examined, tested, and "pruned" (by the deletion of unnecessary hidden units). The fourth subprogram, DONET, makes a TSR routine by which the finished product of the neural net design-and-training exercise can be consulted under other MS-DOS applications.

INTRODUCTION

Recent advances in the manufacture of integrated circuits have led to parallel computers with thousands of microprocessors in a single system and given rise to growing interest in computational methods that support massive parallelism in a natural way. Neural computing aims at (ultimately) achieving human-like performance in computer systems by developing an analogy to the structure and operation of the central nervous system. Computations are executed by simple neuron-like processing units which are interconnected by "synaptic" links of variable strength (or weight). The resurgence of interest in these "connectionist" models, since about 1982, has been said to reflect the total inadequacy of algorithm-driven computing and symbolic artificial intelligence (AI) approaches in dealing with real world problems, speech processing and machine vision being the most cited examples.

On the other hand, neural networks have found abundant use in recent years as practical decision aids which can learn by example to give correct responses to given inputs in situations where, in principle, a set of logical rules could be used to infer the correct response but where, in practice, such rules are difficult to elucidate. Successful case studies have been reported in sonar echo classification, concealed explosives detection, mortgage risk evaluation, medical diagnosis, and many other applications. Much of the popularity surrounding neural network classifiers is a consequence of the relative ease with which they can be trained to substitute for optimally designed expert systems. In most of the interesting applications so far, the powerful new result which enables the neural network to capture the significant factual associations presented in the training data is a learning algorithm called *back-propagation*.

Original Backprop (Version 1.2) is an MS-DOS software package for setting up and training three-layer, feed-forward neural networks to classify input patterns consisting of binary- and real-valued components. It uses the oldest (and least widely known) form of the back-propagation learning algorithm to achieve a degree of flexibility and performance which rivals some of the more popular software-only neural net development products on the market today. Its predecessor (Version 1.1), which was distributed as shareware to members of the international neural networks research community and a tri-services working group, has been applied with some success to the automation of medical diagnosis [1], to active sonar target classification [2], and to

personnel screening. It was also used (to no apparent advantage) in financial forecasting [3]. At the present time, the author is exploring the application of *Original Backprop* to the interpretation of questionnaire data produced by a pre-prototype software package for the prevention and remediation of sexual harassment.

The next section, which explains the historical origins of the learning algorithm, will clarify some points of functionality and terminology which have to be understood before the package can be used effectively. The third section walks the reader through an example problem (using Version 1.1) in which a network with random initial weights is set up and then trained to classify the elements of a training set.

BACKGROUND

The neural network technology of today is based largely on the neuroscience of the 1940s. The classical neuron integrates the pre-synaptic activity of all the neurons influencing it, sending information in the form of electrical impulses down the one-way path formed by its axon. McCulloch and Pitts, in 1943, simplified the neuron to an on/off device, either firing impulses at its peak rate or resting quietly. In 1948, D.O. Hebb theorized that the synaptic weights are modified by a reinforcement control procedure; and he argued that synaptic modification constituted the microscopic, physiological basis of adaptation, learning, and behavioral organization. How could this theory be tested?

By 1954, digital computers had been brought to bear on the problems of brain modeling. The computer was indispensable, because the mathematics involved large numbers of variables and their nonlinear interactions. If "intelligent" behavior was going to emerge from networks of McCulloch-Pitts neurons with Hebbian synapses, it would have to be discovered by computer simulation. No one carried this idea so far, so fast as Frank Rosenblatt, whose discoveries were summarized in a 1961 Cornell Aeronautical Laboratories technical report, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms* [4]. In 1962, *Neurodynamics* was published and distributed by Spartan Books (now defunct [5]). The first 300 pages of Rosenblatt's report were devoted to three-layer, series-coupled perceptrons composed of binary threshold units (or McCulloch-Pitts neurons). The front end of the perceptron consists of sensory (S) units on which a pattern of binary digits is impressed. The back end consists of response (R) units which register the classification. Between these is a layer of association (A) units each of which forms a weighted sum of the pattern components and then turns ON (or OFF) when the sum exceeds (does not exceed) a threshold. Similarly, the R-units turn ON or OFF according to the values of the weighted sums that they compute after scanning the A-layer. In Figure 1, some newer terminology is superimposed on this 30-year-old design.

Rosenblatt trained simple perceptrons to solve problems in pattern recognition. For example, let the S-units form a grid-like retina on which horizontal and vertical bars are impressed by turning ON the units in a particular row or column. Let there be only one R-unit which we want to turn ON in response to vertical bars only. The weights of the front-end connections (from S to A) and the back-end connections (from A to R) are initially just random numbers; and the perceptron's initial performance might be correct about half the time. The training process follows a sequence of cycles. A pattern is presented at the front end and propagated through the A-layer to the R-unit. If the response is correct, go on to the next pattern. If incorrect, then change the weights of all A-to-R connections which contribute to the error. The weight change will be negative when the A-unit helps to turn ON the R-unit in response to a horizontal bar, positive when it inhibits the R-unit's response to a vertical bar. As this procedure is repeated again and again, the perceptron's incorrect responses become fewer and fewer.

In 1969, MIT computer scientists Marvin Minsky and Seymour Papert published a book, *Perceptrons*, which is widely regarded as having had a chilling effect on the subject. In the third (1988) edition, Minsky recalls that perceptron research had already reached a dead end [6]. After Rosenblatt died in a boating accident on the Chesapeake Bay, in 1971, Minsky and Papert dedicated the second edition of *Perceptrons* in his memory. Yet the memory of what Rosenblatt accomplished faded quickly in the years that followed as students increasingly accepted the Minsky-Papert perceptron as a substitute for the original--and found it lacking in problem-solving ability.

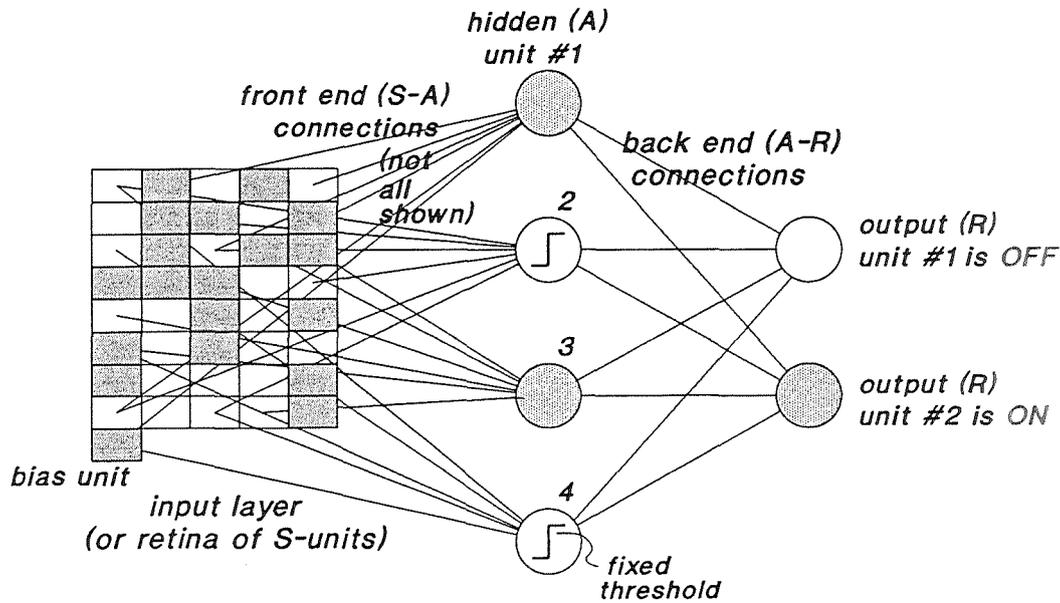


Figure 1. A SIMPLE PERCEPTRON features three layers of neuron-like units with weighted connections feeding excitation (and inhibition) in the forward (l.-to-r.) direction only. Here one of the two output units turns ON to the input pattern.

The revival of perceptron-like models in the 1980s was made possible by a combination of developments, including the widespread perception that AI had reached a plateau, and by the availability of cheaper, faster computers with large amounts of inexpensive RAM (which is needed to store the synaptic weights of large networks). The connectionist models of the 1980s overcame some weaknesses of the Minsky-Papert perceptron. Rumelhart, McClelland and the PDP Research Group (1986), in their first widely-read volume on *Parallel Distributed Processing*, emphasized the importance of having a "hidden layer" of neuron-like units sandwiched between the input and output layers of the network [7]. It was as if Minsky and Papert had done away with the A-units in the perceptron. So these had to be re-invented as "hidden units"! The PDP Group pointed out that these hidden units give three-layer networks the ability--in principle--to solve virtually any pattern classification problem.

But the "powerful new result" that drove the progress of artificial neural networks in the late 1980s was an algorithm called "back-propagation" which permits three-layer networks to learn internal representations of data sets for which no mathematical model can be written down to specify the correct responses to given inputs. Instead, the neural network learns by example in the course of many passes through a training set. In 1986, T. Sejnowski demonstrated NETtalk, the neural network that learned to read aloud in English. The input units in the three-layer network represented sequences of letters from a text. The output units corresponded to the "phonemes" of which spoken English is made. The phonemes were transmitted to a speech synthesizer. NETtalk learned by example to convert letter strings into phonemes. The PDP Group's back-propagation technique was used to modify the weights in a way that resisted and eventually corrected the errors. The speech produced by the network was initially just a meaningless babble. As training progressed around the clock on a mainframe computer, the sounds became more and more intelligible. After the network had learned the training set, it showed the ability to generalize by "reading aloud" the remaining text. This and a legion of other persuasive demos have testified to the power of back-propagation, which has driven the great majority of neural network applications to date. The technology has evolved so far, so fast, that its roots have become almost invisible. According to the prevailing historical view, back-propagation is radically different from the training procedures used with perceptrons [8, 9]. Although the introduction of hidden units gives a feed-forward

network the potential to learn an arbitrary input-to-output mapping, in this view, no technique had existed for training the weights of a network with one or more hidden layers.

It is true that Rosenblatt usually left the weights of the front end (S-to-A) connections at their initial values and applied corrective modifications only to the back end (A-R) weights. In chapter 13 of *Neurodynamics*, however, Rosenblatt addressed the limitations imposed by neglecting to modify the front-weights: "Only one constraint needs to be dropped in order to obtain the most general system of this class: the requirement that the S-to-A connections must have fixed values, only the A-to-R connections being time dependent. In [Chapter 13], variable S-to-A weights will be introduced and the applications of an error-correction procedure will be analyzed. It would seem that a considerable improvement in performance might be obtained if the S-to-A connections could somehow be optimized by a learning process rather than accepting the arbitrary or pre-designed network with which the perceptron starts out. It will be seen that this is indeed the case, provided that certain pitfalls in the design of a reinforcement control procedure are avoided."

With this rationale, Rosenblatt introduced a "back-propagating error correction procedure" consisting of a brief list of rules for assigning errors to hidden (A) units based on their interactions with output (R) units that assume the wrong state in response to the training input. Back-propagation is a "supervised" learning algorithm which obtains its feedback from the output units, computing errors by comparing their observed states to preassigned correct values, propagating errors (and corrections) back towards the front (input) end of the net if a satisfactory solution cannot be found quickly by making corrections at the output end. The actual modification to the weights is formally the same whether an output unit or a hidden unit (or A-unit) is considered. Thus if the error assigned to a unit is positive, the weights of all connections from active units are increased, eventually turning it on. If the error is negative, the weights of connections from active units are decreased. The essential feature of the method is a probabilistic procedure for assigning errors to hidden units.

USING ORIGINAL BACKPROP

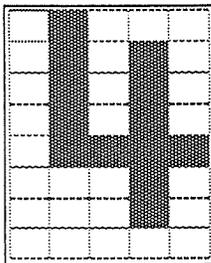
Original Backprop includes four subprograms: (1) *TSET*, a graphical interface for creating the training sets; (2) *BACKPROP*, which generates and trains neural networks; (3) *VIEWNET*, which lets the user analyze and simplify the networks; and (4) *DONET* which recalls and runs the finished product of the neural net design-and-training exercise under other software applications.

TSET presents the user with an 5-by-8 grid of picture elements (or "input units") which can be toggled ON or OFF with a keystroke. In Figure 2, the grid is used to draw 16 patterns representing the hexadecimal symbols zero through F. Pattern number 5, for example, is the symbol "4" which has the binary representation 100. This training set, consisting of four pages of four patterns each, will show an appropriately configured neural net how to map the symbol patterns into binary numbers. Onscreen help is provided for moving around in the pattern set and for naming the individual picture elements when appropriate. In a medical diagnosis problem [1], for example, the picture elements could be placed in one-to-one correspondence with the (40 or fewer) symptoms and named accordingly so that the meaning of the "input unit" is clearly defined as the cursor is moved around the grid in the process of data entry. Training sets are saved as ASCII *.set files. In Figure 2, the file name is *hex.set*; and it is divided into four pages of four patterns each. Version 1.1 limits the size of the training set to 10 pages of binary-valued patterns. Version 1.2 increases the capability to 200 patterns with up to 128 components each and lets the picture elements be represented with 8-bit precision (and 256 colors).

BACKPROP is operated from two menus. The Main Menu presents these Options: (1) get a training set; (2) get a neural net; (3) create a new network; (4) test/train a network; and (5) quit. Option (1) is the obvious starting point. Once a *.set file has been retrieved, it is displayed in a binary string format as shown in Figure 3 for the hex-to-binary conversion problem. The desired mapping is from "input" into "class". The "out" column in the table is all zeros at this point; but one can return to this screen later on (by Option e, below) when training is underway to see how the output units of the network compare to the desired classifications. Selecting Option (3) produces the screen shown in Figure 4. Observe that the numbers of input and output units have defaulted to the numbers indicated by the dimensions of the training set. The number of hidden units has

EDIT/CREATE a Training Set
file name: hex.set

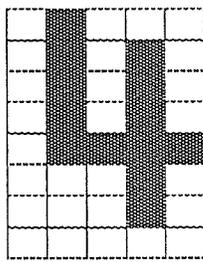
pattern 5 0 1 0 0 0



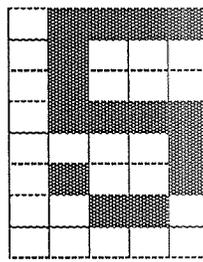
input unit 0:
pattern name: 4

Arrows move cursor; T toggles unit.
Press Insert to record pattern
and advance to next pattern.
Press - (minus) to back up.
F1 = Help

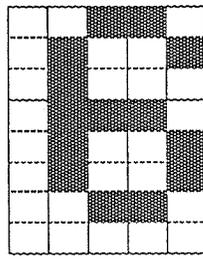
patt. 5 0100



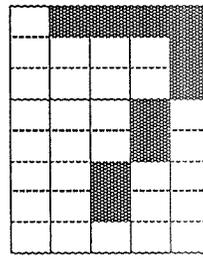
patt. 6 0101



patt. 7 0110

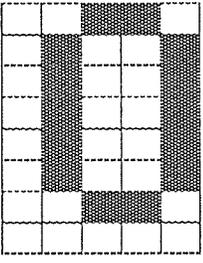


patt. 8 0111

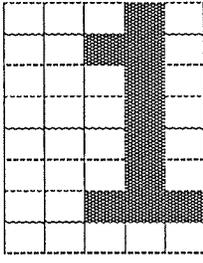


page 2

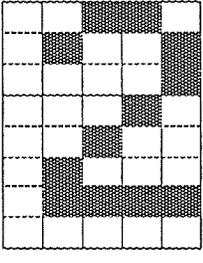
patt. 1 0000



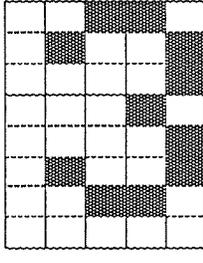
patt. 2 0001



patt. 3 0010

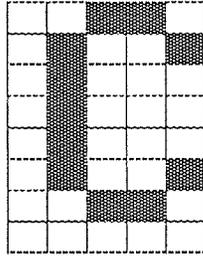


patt. 4 0011

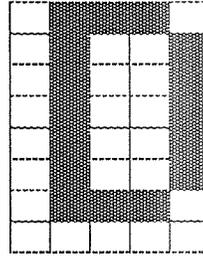


page 1

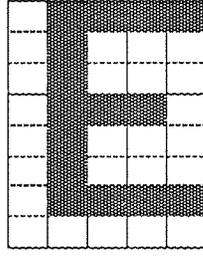
patt. 13 1100



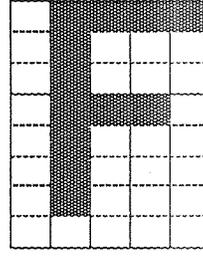
patt. 14 1101



patt. 15 1110



patt. 16 1111



page 4

FIGURE 2. TSET lets Version 1.1 users create input patterns on a 5-by-8 grid and attach as many as five classification bits to each pattern. The product is a training set which is saved as a *.set file.

TRAINING SET

File name: hex.set { 16 patterns }

Page Down to view more patterns. Strike a key to continue.

#	name	input	class	out
1	0	00110010010100101001010010100100110	0000	0000
2	1	00010001100001000010000100001000111	0001	0000
3	2	00110010010000100010001000100001111	0010	0000
4	3	00110010010000100010000010100100110	0011	0000
5	4	01000010100101001010011110001000010	0100	0000
6	5	01111010000100001111000010100100110	0101	0000
7	6	00110010010100001110010010100100110	0110	0000
8	7	01111000010000100010000100010000100	0111	0000
9	8	00110010010100100110010010100100110	1000	0000
10	9	00110010010100100111000010100100110	1001	0000
11	A	00110010010100101111010010100101001	1010	0000
12	B	01110010010100101110010010100101110	1011	0000
13	C	00110010010100001000010000100100110	1100	0000
14	D	01110010010100101001010010100101110	1101	0000
15	E	01111010000100001110010000100001111	1110	0000

FIGURE 3. BACKPROP displays the first 15 elements of the training set (hex.set) as binary strings.

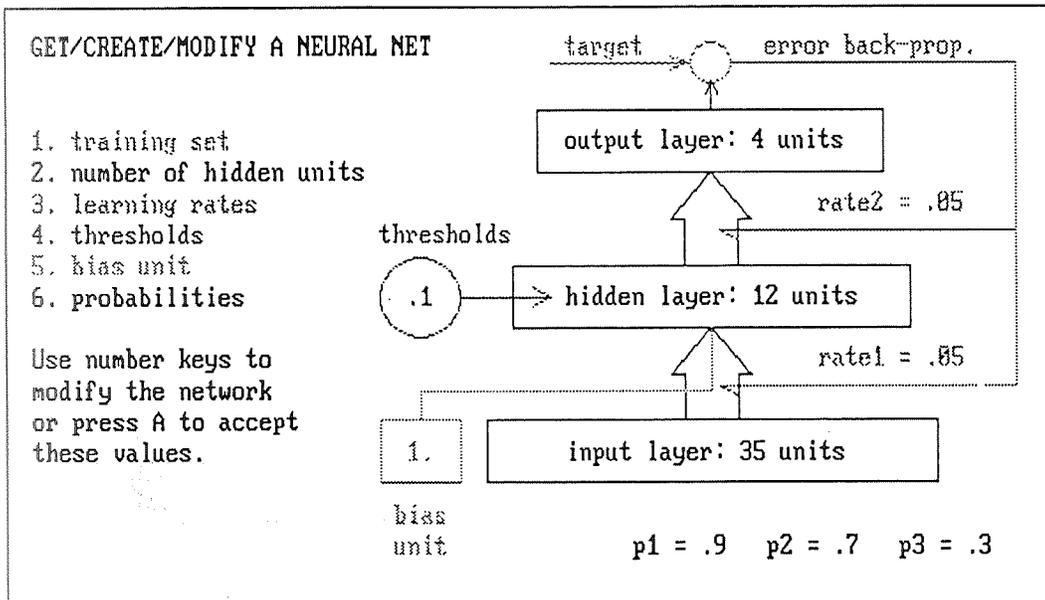


FIGURE 4. BACKPROP creates three-layer networks with the appropriate numbers of input and output units (as required by the training set). This screen serves as a control panel to adjust the number of hidden units and other network parameters.

been set to 12. Thresholds have been established in these hidden units and the input layer bias unit has been turned on.

BACKPROP's Main Menu Option (4) brings up a Training Menu which contains these seven new options: (a) begin training; (b) freeze/unfreeze weights; (c) modify network; (d) save network; (e) review patterns; (f) continue training; and (g) return to Main Menu. Choosing Option (a) now starts the process of learning to associate binary numbers with the symbols in *hex.set*. Figure 5 (top) shows the learning curve which resulted from 278 cycles through the training set of 16 patterns. Two learning curves are actually displayed: A red curve shows the number of incorrectly classified patterns in the current epoch of 50 cycles and a white curve (presently in the upper left corner) shows the average number of errors epoch-by-epoch. It turns out that *hex.set* is a rather difficult assignment. Pressing the ESCape key after cycle number 278, where the error rate has dropped below two-thirds, Option (c) is used to re-access the network parameter control screen of Figure 4. Modifying the "learning rates" (so that *rate1* = .01 and *rate2* = .001), then continuing the training with Option (f), the learning process is rapidly completed as shown in the bottom half of Figure 5. Note that Rosenblatt's stochastic learning algorithm, although it guarantees convergence when a solution exists, does not give the sort of monotonic learning curve that users of PDP back-propagation are accustomed to seeing. The trained network is saved as a *.*net* file after exercising Option (d). Since there are 12 hidden units in the hex-to-binary conversion network, the weights are saved in a file called *hex12.net*.

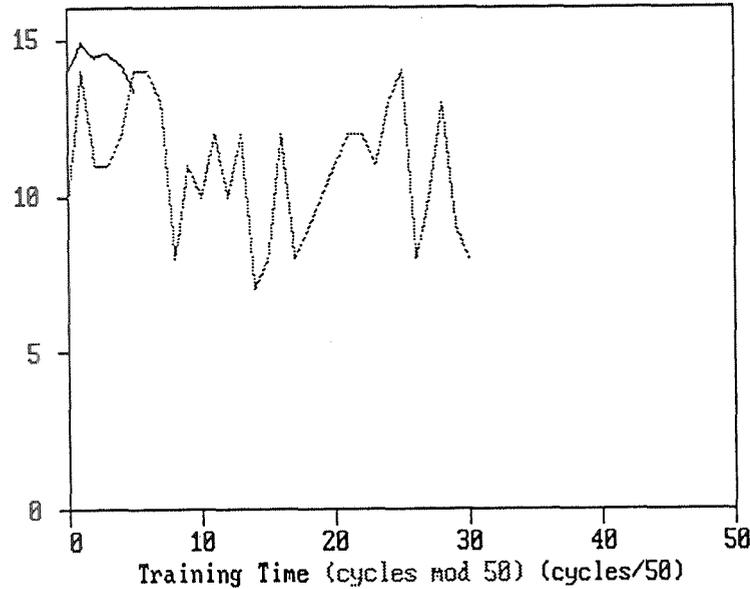
VIEWNET lets the trained network be examined, tested, and "pruned" by the deletion of marginally useful hidden units. *VIEWNET*'s menus control the acquisition of *.*set* and *.*net* files and give the user two "views" of how the network deals with the training set. The detailed view is presented on the "Neural Net Display Screen" (Figure 6) which allocates a small box for each unit and a wide box for each weight. The cursor moves up and down the hidden layer. In Figure 6, the cursor illuminates hidden unit #12; and the weights displayed are all those of the (S-A) connections fanning into this unit from the input layer together with those fanning out to the four output units. On the right side of the screen, the four output units still register "0000" (instead of the desired "0101") because the SPACEbar has not been pressed to propagate the input (pattern #6) forward. A less complicated depiction of the network's performance is obtained by listing the hidden layer activation vectors as columns under the corresponding pattern numbers as in Figure 7.

A recurring question in neural net research concerns the number of hidden units needed to solve the problem presented by the training set. If too few hidden units are employed, training progress may be extremely slow or the solution may actually be unattainable irrespective of any time limit. Use of too many hidden units results in "brittle" solutions and networks that do not generalize well. Some pioneering work of Australian Navy investigators J. Sietsma and R. Dow suggests that the most practical and expedient approach is to first set up and train a network with an abundance of hidden units and then "prune" the trained network by selectively deleting those units which contribute little or nothing to overall performance [10]. *VIEWNET* is the tool that makes it practical to implement such a strategy. From the screen shown in Figure 6, the user can pick a hidden unit (corresponding to a row of the binary array), delete it, and see what effect this has on the correctness of the net's response to each training pattern. Although it requires some work, moving "manually" back-and-forth between *VIEWNET* (to prune one or two units at a time) and *BACKPROP* (to correct the few new errors thus incurred) leads to efficient solutions in much less time than it would take using *BACKPROP* alone. (For example, a network with seven hidden units can be obtained by pruning *hex12.net* in stages; but for *BACKPROP* to solve the problem posed by *hex.set* directly--starting with just seven hidden units--seems to take far more than 250,000 cycles.) A desirable feature which has *not* been included in Version 1.2 (but deferred to later upgrades) is an "autoprune" option which would obviate the need for such "manual" labor.

Version 1.2 improves upon its predecessor by supporting larger training sets and networks. It also includes a new subprogram, *DONET*, to exercise trained networks (retrieved as *.*net* files) and display their responses to given inputs in a dialog box that pops up under other applications (like spreadsheets and word processors). In Version 1.2, *BACKPROP* can be made to find more robust solutions by injecting low-level "noise" into the input patterns in the concluding phases of the training process.

LEARNING CURVE
 training set (file name) = hex.set

Number of Errors (Mean Errors)



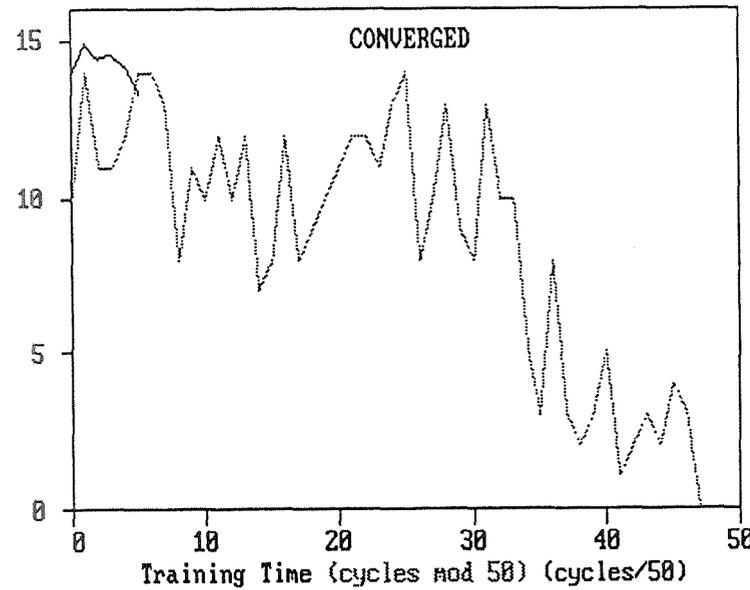
Pattern Number and Status			
correct/incorrect			
1	11	21	31
2	12	22	32
3	13	23	33
4	14	24	34
5	15	25	35
6	16	26	36
7	17	27	37
8	18	28	38
9	19	29	39
10	20	30	40

cycle number 278
 # of errors 8

ESCape returns to
 Training Menu

LEARNING CURVE
 training set (file name) = hex.set

Number of Errors (Mean Errors)



Pattern Number and Status			
correct/incorrect			
1	11	21	31
2	12	22	32
3	13	23	33
4	14	24	34
5	15	25	35
6	16	26	36
7	17	27	37
8	18	28	38
9	19	29	39
10	20	30	40

cycle number 294
 # of errors 0

ESCape returns to
 Training Menu

FIGURE 5. *BACKPROP* produces learning curves to show the number of misclassified patterns as a function of the number of cycles through the training set. The slow progress in the first 278 cycles (top) is accelerated by *lowering* the learning rates from their default values (as described in the text). All 16 patterns in *hex.set* are learned in 294 cycles (bottom).

NEURAL NET DISPLAY SCREEN (netfile = hex12.net)

BIAS		thresholds: 0.10			
1 0	-1.719421E-02	21 0	4.825204E-01	1 1	1.002412E-01
2 1	-2.925786E-01	22 0	-1.213647E-01	2 0	-4.081562E-02
3 1	4.735093E-03	23 0	-1.045046E-01	3 1	7.021353E-02
4 1	-7.121240E-02	24 0	-1.781987E-01	4 0	1.048494E-03
5 1	-3.578509E-01	25 1	2.277248E-01	5 1	
6 0	2.250073E-01	26 0	-2.370896E-01	6 1	
7 1	4.721043E-01	27 1	1.008853E-01	7 1	
8 0	3.072990E-01	28 0	4.238055E-01	8 0	
9 0	4.367573E-01	29 0	-8.180900E-02	9 0	
10 0	-6.324503E-02	30 1	7.096180E-03	10 0	
11 0	3.131445E-01	31 0	4.376945E-01	11 1	
12 1	3.523290E-01	32 0	4.577186E-04	12 0	
13 0	1.542164E-01	33 1	1.899377E-02	13	
14 0	3.097313E-01	34 1	-1.425915E-01	14	
15 0	3.101549E-01	35 0	3.185956E-01	15	
16 0	-8.552058E-02	36		16	
17 1	4.872202E-01	37			
18 1	-6.364523E-02	38			
19 1	2.338015E-02	39			
20 1	-1.814835E-01	40			
20 1	-1.790298E-01				

Input layer: 35 units

Hidden layer: 12 units

ESCape-Go to Menu

NETWORK OPERATION

Arrows-----Hidden units

PgUp/PgDn--Patterns

Del/Ins----Pruning

SPACE-----Fwd. Prop.

Weight, Value, Bit, Class

Output layer: 4 units

T-SET: hex.set
(16 patterns)
Net File: hex12.net
Pattern #6: 5
of errors: 0
CONVERGED

FIGURE 6. VIEWNET's Neural Net Display Screen illuminates the contents of weight file hex12.net and shows how the network responds to the elements of the training set.

Training Set: hex.set
Netfile: hex12.net
Command (F1 to get help)?

pattern#	
000000001111111	
1234567890123456	

unit#	
1	1011010110110001
2	01010101000
3	10001011001
4	01110001011
5	11110000111
6	10111111000
7	11010100110
8	00000101010
9	0000110110001000
10	0001110000010111
11	1011001000100010
12	0001001011111010

COMMAND OPTIONS

0: Escape to menu

x: Delete hid. unit x

-1: re-insert hid. unit

rrx: ON rectangle row x

rlx: OFF rectangle row x

rdx: ON rectangle col x

rux: OFF rectangle col x

Total Errors: 0

error indicator

FIGURE 7. VIEWNET gives users the ability to "prune" the network by deleting marginally useful hidden units.

AVAILABILITY

Original Backprop, Version 1.2, will be ready for release in February, 1993. Requests should be sent to the author by regular mail.

ACKNOWLEDGEMENTS

Studies leading to the development of *ORIGINAL BACKPROP* were sponsored by the Office of Naval Research through the Naval Surface Warfare Center's Independent Research Program. Most of the critical components of Version 1.1 were designed and programmed in Borland Turbo-C, during the summer of 1991, by Dovid Lipman, who is presently a student at the Ner Israel Rabbinical College (in Baltimore).

REFERENCES

- [1] Agyei-Mensah, S.O., and Lin, F.C. (1992). Application of neural networks in medical diagnosis: the case of sexually transmitted diseases. *Submitted for publication*.
- [2] Harrison, R.W. (1991). *A neural network for classifying active sonar returns* [Naval Surface Warfare Center, Dahlgren, VA], Tech. Report No. 91-327.
- [3] Coughlin, J.P. (1992). Measures of serial data compressibility by neural network predictors. *Proc. Int'l. Joint Conf. on Neural Nets. [IJCNN Baltimore '92]*, 755-761.
- [4] Rosenblatt, F. (1961). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms* [Cornell Aero. Lab., Buffalo, NY], Tech. Report No. VG-1196-G-8].
- [5] Rosenblatt, F. (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms* [Spartan Books, Washington, DC].
- [6] Minsky, M.L. and Papert S.A. (1988). *Perceptrons* [Expanded Edn., MIT Press].
- [7] Rumelhart, D.E., McClelland J.L., and the PDP Research Group (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. I [MIT Press, Cambridge, MA].
- [8] Widrow, B. and Lehr, M.A. (1990). Thirty years of adaptive neural networks: Perceptron, Madaline, and Backpropagation. *Proceedings of the IEEE* 78(9), 1415-1442
- [9] Denning, P.J. (1992). Neural networks. *American Scientist* 80, 426-429.
- [10] Sietsma, J., and Dow, R.J.F. (1988). Neural net pruning - why and how. *Proc. IEEE Int'l. Conf. on Neural Networks*, 325-333.