How-To-Use-It-Manual
Stimulus Generation and Association Programs
Brown University Neural Modeling Group
Release 2.05
BSB and ASSOCIAT

James A. Anderson
Department of Cognitive and Linguistic Sciences
Brown University
Box 1978
Providence, RI 02912
`James_Anderson@brown.edu`

October 30, 1994
October 24, 1997 (C version)

1 Introduction

This manual will tell you how to use a set of programs developed at the Department of Cognitive and Linguistic Sciences, Brown University. The programs are research programs designed to generate, maintain, learn, and use sets of stimuli and matrices used in neural modelling research.

The programs were originally written in VMS Pascal to run on a VAX, but are now also available in C to run under Unix. This document has been modified slightly to reflect the latter version. There are two programs of major interest. One is called ASSOCIAT and generates the matrices that associate pairs of vectors. Actually, the internal representation of a matrix is in the form of an array of C structures named Neuron, but acts mathematically like a matrix. The other is a program with several parts called BSB. One part generates stimuli which are large state vectors. These state vectors represent strings of 25 characters in a 200 dimensional system. Once generated, the state vectors can be associated together using ASSOCIAT. Another part realizes the dynamics of the simple non-linear model called the 'Brain-state-in-a-Box'. (See Anderson, Silverstein, Ritz and Jones, 1977, Anderson and Mozer, 1981; Anderson, 1986; and Anderson and Murphy, 1986; Anderson, 1993,1994)

2 Files

The programs BSB and ASSOCIAT need to be supplied with several file names, in order either to read or write the files. The file names are supplied as command line arguments when the programs are run. Below are the commands to create the stimulus files, to create the matrix, and to run the demo.

```
./bsb fdis.stm gdis.stm tdis.stm
./associat fdis.stm gdis.stm ndis.neu
./bsb fdis.stm gdis.stm tdis.stm ndis.neu
```

The first generates the three `.stm` files, the second uses these as input to construct the `.neu` file, and the third uses all four files as input. (The initial `./` is only necessary if your path does not include the current directory.)

By convention, the input and output vectors are given the extension '`.stm`' and the file containing the matrix are given the extension '`.neu`'. As a general point, the files constructed with these programs can be quite large. In the 200 dimensional system the files associated with the matrix are 324000 bytes long.

3 Making the Stimuli using BSB

The first step in making stimuli is to run BSB. Some initial information appears. The general pattern of displays in BSB is to have status information on the top 5 lines of the screen and the command prompts on the bottom line. Since the author of these programs was influenced by UCSD Pascal at an impressionable age, extensive use is made of self-documenting prompts. For example, '`T)hreshold`' means that if 'T' (or 't') is typed at the command prompt ('>') the threshold interpretation parameter can be changed from its default. In general, typing a single letter, followed by <RETURN> will cause an action. All commands can be upper or lower case and most are protected to some extent (not totally!) against erroneous inputs. Not all commands appear on the prompt lines, for reasons of space, particularly for setting some of the less frequently used parameters in the simulation.

Figure 1. Initial appearance of the screen.

————————

```
BSB Neural Net Programs. 3-89  B)SB.  C)hange. E)xit. H)elp.
L)ist. M)ode. N)eurons. R)ead. S)ave. T)hreshold.  W)rite.
 Threshold:  0.5000   F File:    0  G File:    0  T File:    0
NO NEURON FILE.  Display TF.

Initializing Files.
Reading neuron and stimulus files from disk.
Reading F file.
Reading G file.
Reading T file.
Reading neuron file (autoassociative).
```

```
C>
```

At first, there are no files present for BSB to work with. Let us assume you want to create a series of stimuli for future use. The prompt line tells you how many stimuli are present in the F File, the G File, and the T File. When the program first appears, these values are all zero. F File and G File have the customary meanings they have in our neural modelling literature: i.e. the F File is the input set of state vectors and the G File is the output set of state vectors. The T file contains a set of 'Test' vectors, usually partial input stimuli, which are used to test the reconstructive and processing powers of the system.

Although this simple form of our programs only performs autoassociation, (i.e., F=G), more complex versions can have different input and output sets.

Input vectors are constructed from strings of characters. We will assume the system is 200 dimensional. A character in the string is represented by its ASCII number, i.e. 'a' is 97, '5' is 53, '=' is 61, and so on. These numbers are converted into binary representation, so 'a' becomes '01100001'. Then the zeros are replaced with minus ones, so we can maintain rough equality of numbers of positive and negative elements which is convenient for the models, so the actual internal representation of 'a' is

$$\text{`a'} \longmapsto -1,1,1,-1,-1,-1,-1,1.$$

Each character requires 8 vector elements, so a 200 dimensional system contains enough elements for 25 characters. The parity bit is computed but not used in the simulations at present. One reason for its presence, is that eight bits allow the construction of 'orthogonal' character representations, which can sometimes be a convenience. If anyone wanted actually to use these programs for anything substantial it is strongly recommended that a less arbitrary way of coding letters be used. The state vectors contain single precision floating point numbers. Currently no more than 100 stimuli are allowed in any one file but this can be easily changed in the programs if necessary.

The technically inclined might like to know that stimuli are structures composed of a string part and a numerical equivalent form:

```
/* Number_of_characters is related to Dimensionality of vectors */
#define Number_of_characters 25
#define Dimensionality       8*Number_of_characters
/* String_length refers to strings in the struct Stimulus.Name */
#define String_length  60       /* Must be > Number_of_characters */
typedef float Vector[Dimensionality];
typedef char String[String_length+1];
typedef struct {
  String Name;
  Vector Val;
} Stimulus;
```

The program will automatically look for the assigned stimulus and neuron files, and write its progress to the screen. If it doesn't find a file, it will write '`Stimulus file not found`' to the command line (i.e. the 'C>' line). You may have to watch carefully to see this line appear, and then disappear when the next file is read. The information at the top of the screen will also inform you of progress. In the example, there are no stimuli in the F file, G file, and T file and this is noted in the third line on the screen. There is also a comment that there is '`NO NEURON FILE`' on the fourth line.

To contruct a stimulus, type 'C' from the first command level. This allows you to '`C)hange`' the stimuli in the Files, which currently contain no stimulus vectors. There are a number of options in '`C)hange`' to allow you to do various useful things to the stimuli.

Suppose you want to make a new stimulus, say the first stimulus in the F File. Type 'R', for '`Replace.`' The program will then prompt you as to which stimulus to replace. If you say '`F1`', the F[1] position in the F File (which starts off as all blanks) will be replaced with what you type and the appropriate state vector will be constructed. Figure 2 shows the screen after the contents of several vectors (f1, f2, and f3) have been replaced with ASCII character strings and f4 is about to be replaced. (The names of f1, f2 and f3 were `L)isted` and appear in the center of the screen.

Figure 2. Appearance after several f stimuli are made.

——————

```
Template: 1234567890123456789012345
New     : This will be f4.




   F[ 1].  This is stimulus f1.____  G[ 1].
   F[ 2].  This is stimulus f2.____  G[ 2].
   F[ 3].  And this is f3._________  G[ 3].





Replace which stimulus (set, number) :  f4
```

——————

After you choose the stimulus to replace, '`Template:`' will appear at the top of the screen. This is a set of 25 digits to serve as a reference marks for constructing your stimulus. Underneath it is '`New :`' which is where the cursor is and where you can write the new string. Any ASCII character except '_' can be placed here and will be represented as its appropriate byte. The ASCII character '_' is represented in the vector as all zeros. If you hit return before the end of the string, the vector will be filled out with zeros, represented as underlines. If you type in more than 25 characters, the excess will be ignored. Remember to use <DELETE> to remove characters from a line. Using the <BACKSPACE> key will insert 'CTRL-H' control characters into the line, causing erratic behavior of the system that is difficult to detect.

After you have typed in a few stimuli, you can see the current set by typing 'L' for '`L)ist`'. This will list either the F file and the G File or the T file and the F file, depending on which M)ode you have chosen. If there are more stimuli than will fit on the screen you can move the file B)ackwards or F)orwards by typing 'b' or 'f'. Typing any other letter will return you to where you were. The listing will remain on the screen for reference until new information has to be written over it.

Briefly the commands in C)hange are:

`A)dd`. Form a state vector as the sum of other state vectors. The resulting vector values are divided by the number in the sum (i.e. the average of the vectors is taken). The '`.Name`' of this type of state vector will appear in the `L)isting` as, for example, '`Sum f01 f02 f03`' if it was the sum of stimulus vectors F[1], F[2], and F[3]. Up to 16 vectors can be summed.

`C)opy`. Copy a state vector in one location to another location. The program will prompt you for the stimulus to be copied and where to put it.

`E)dit`. This useful command will let you 'edit' an already existing stimulus. It will ask you for the stimulus number to edit. It will then give the old stimulus as the template and let you type in the new stimulus right below it. When you type <RETURN> the new stimulus vector will replace the old one.

`I)ndividual` values. Will show the individual element values. The elements arranged on a byte per line with the number of the first and last elements on the line numbered on left and right sides of the screen, and the ASCII interpretation of the byte on the left. This command has its own command line.

`C)hoose` chooses which vector to show.

`D)isplay` displays the elements.

`M)ode` (see below).

`Q)uit` returns to `C)hange`.

`T)hreshold` (see below).

`X)change` allows you to change individual values.

`L)ist`. Lists the state vectors in the files. If the display is in FG mode, the F File and the G File will be listed. If the display is in T mode, the T File and the F File will be listed. If more than 16 vectors are present, only sixteen at a time will be shown. The 16 stimuli displayed can be shifted forward or backward by typing 'F' or 'B'. L)ist can be left by typing any character other than 'F' or 'B'.

`M)ode`. Change listing mode from displaying the F File and the G File to displaying the T File and the F File or vice versa.

`Q)uit`. Return to wherever you entered `C)hange` from. `C)hange` can also be entered from BSB.

`R)eplace`. Replace a member of one of the files with a new stimulus.

`S)ave`. Save the file of stimuli to a disk file. The program will prompt you as to which file(s) you want to save. The saved file will be given the names you specified on the command line. Don't forget to `S)ave` your work!

>>>The program does NOT automatically save new stimuli.<<<

`T)emplate`. Sometimes it is convenient to use a template other than digits to make remembering what goes where easier. `T)emplate` allows you to make your own string of ASCII characters to serve as a template, or to use a member of one of the Stimulus files as a model for a template.

4 Using ASSOCIAT to Make a New Learning Matrix

Suppose you have constructed a set of F and G stimuli. You have saved them to appropriate disk files, and now wish to form the association matrix between the input and output sets. This is done with the program named ASSOCIAT. ASSOCIAT can sometimes take sizeable amounts of computer time on older VAXes. Also the data files ASSOCIAT automatically creates to store the results are very large (324000 bytes in a 200 dimensional system) so be sure you have enough free space in your directory to write such a large file, otherwise the results of all the computing may be lost.

If you run ASSOCIAT, first the F and G files will be listed so you can see if they are satisfactory. Below is a listing of a session using ASSOCIAT to generate an association matrix. The data set is a set of ambiguous descriptors used in the disambiguation demonstration. (This is the output seen when the script file `lrndis` is executed.)

```
ASSOCIAT program.  October 28, 1994.

The program is reading the FFILE.
The dimensionality of the system is 200

The program is reading the GFILE.

F and G stimuli used.

 F[ 1] : BaseballGameBat BallDiamd
 G[ 1] : BaseballGameBat BallDiamd

 F[ 2] : Vampire MythBat NiteDracu
 G[ 2] : Vampire MythBat NiteDracu

 F[ 3] : Animal  LiveBat WingFlyng
 G[ 3] : Animal  LiveBat WingFlyng

 F[ 4] : Poker   GameBeerTablCards
 G[ 4] : Poker   GameBeerTablCards
 F[ 5] : Tennis  GameCortBallRackt
 G[ 5] : Tennis  GameCortBallRackt

 F[ 6] : Dancing RichPrtyBallSocty
 G[ 6] : Dancing RichPrtyBallSocty

 F[ 7] : GeoShapeTwoDCrclSqreDiamd
 G[ 7] : GeoShapeTwoDCrclSqreDiamd

 F[ 8] : GeoModelTreDSphrBallTetra
 G[ 8] : GeoModelTreDSphrBallTetra

 F[ 9] : ExpJewelRichRubyOpalDiamd
 G[ 9] : ExpJewelRichRubyOpalDiamd

Seed for RN generator           : 123
Number of associations to learn : 250
Use CORRECTION procedure? Y or N: y
Use old Nfile as start? Y or N  : n
Number of synapses              : 100
Setup completed.

   10  Nr:    8   Cosine:  5.16E-01
   20  Nr:    5   Cosine:  9.01E-01
   30  Nr:    4   Cosine:  9.48E-01
   40  Nr:    1   Cosine:  9.60E-01
   50  Nr:    2   Cosine:  9.25E-01
   60  Nr:    6   Cosine:  9.47E-01
   70  Nr:    5   Cosine:  9.87E-01
   80  Nr:    2   Cosine:  9.74E-01
   90  Nr:    4   Cosine:  9.97E-01
  100  Nr:    3   Cosine:  9.70E-01
  110  Nr:    9   Cosine:  9.91E-01
  120  Nr:    5   Cosine:  9.95E-01
  130  Nr:    4   Cosine:  9.98E-01
  140  Nr:    7   Cosine:  9.89E-01
  150  Nr:    4   Cosine:  9.97E-01
  160  Nr:    8   Cosine:  9.96E-01
  170  Nr:    5   Cosine:  9.96E-01
  180  Nr:    8   Cosine:  9.99E-01
  190  Nr:    4   Cosine:  9.99E-01
  200  Nr:    4   Cosine:  9.99E-01
  210  Nr:    5   Cosine:  9.99E-01
  220  Nr:    8   Cosine:  9.99E-01
  230  Nr:    7   Cosine:  1.00E+00
  240  Nr:    1   Cosine:  9.99E-01
  250  Nr:    2   Cosine:  9.99E-01

Accuracy of recall of input set.
```

```
1  Name: BaseballGameBat BallDiamd
Cosine:  9.99E-01  Length:  1.40E+01

2  Name: Vampire MythBat NiteDracu
Cosine:  1.00E+00  Length:  1.39E+01

3  Name: Animal  LiveBat WingFlyng
Cosine:  9.99E-01  Length:  1.40E+01

4  Name: Poker   GameBeerTablCards
Cosine:  1.00E+00  Length:  1.40E+01

5  Name: Tennis  GameCortBallRackt
Cosine:  9.99E-01  Length:  1.39E+01

6  Name: Dancing RichPrtyBallSocty
Cosine:  1.00E+00  Length:  1.41E+01

7  Name: GeoShapeTwoDCrclSqreDiamd
Cosine:  1.00E+00  Length:  1.40E+01

8  Name: GeoModelTreDSphrBallTetra
Cosine:  9.99E-01  Length:  1.38E+01

9  Name: ExpJewelRichRubyOpalDiamd
Cosine:  9.98E-01  Length:  1.38E+01


Writing to neuron output file.
```

The Prompts are mostly self explanatory.

1. Seed starts off the random number generator. Pairs of associations from the set of stimulus pairs are presented randomly. A particular sequence can be repeated if the same seed is used.

2. The number to be learned depends on the number of items in the stimulus sets. If the Widrow-Hoff error correction procedure is used, about 30 presentations per pair is a good place to start. If the linear associator is used, only one presentation of each pair is necessary.

3. Correction Procedure. If 'Y' or 'y', the Widrow-Hoff error correction procedure will be used, otherwise, the linear associator without error correction will be used.

4. Used Assigned Nfile. If 'Yes', a previously constructed `.neu` neuron file will be used as a starting point, otherwise all matrix elements start off at zero. You can teach an old matrix new tricks if you wish. If 'No', then a new matrix will be created.

5. Number of synapses is the number of non-zero synapses in the matrix. The rest of the synapses are set identically to zero. Execution speed is a linear function of number of synapses because of the way the programs are written

and the files are constructed. This is because the matrices are not stored as matrices but as lists of non-zero synaptic connections. Fewer synapses means less computation time. Usually 50% connectivity is adequate for most systems we have investigated up to now and seems to make the systems work better than full connectivity, for reasons we now have some insight into. It takes a while to set up the matrix, so expect a wait. If you set a connectivity of over 80%, you will get a warning message, telling you to expect a VERY long wait. However, 100% connectivity is handled separately, and sets up very fast.

After the matrix is set up and the system starts learning, the program will print out a progress report every 10 presentations. It will tell you the cumulative number of trials, the stimulus pair presented that trial, and the cosine between obtained and desired output vectors, so you can check on progress and make sure the stimuli are being presented correctly. After learning is over, the program will test how well the system learned. It will take each F in the F File in turn as input, compute the output, and compare that output with the correct association by computing the cosine between the actual and correct answers. A cosine of 1.0 is perfect. The length of the vectors is also computed. The new neuron file will then be written to whatever file was specified on the command line. Remember, in a 200 dimensional system, these files are 324000 bytes long so make sure you have enough free space in your account.

5 Using the Matrix

To test the behavior of a previously constructed matrix, run BSB.

You must read in the matrix and the stimulus sets you wish to test. This is done automatically in BSB. The program will inform you of its progress and whether or not it found the specified files. Success or failure in retrieving files will appear on the status line (i.e. the bottom line). This line will appear and disappear so you have to watch carefully if you are interested. The status box (the top five lines) will tell you how many stimuli are in the stimulus files (0 if the files were not present) and whether neuron files are present. (See Figure 1.)

If the program cannot find the required files, it will tell you, but will not stop the program. Some other input errors may abnormally terminate the program and you will get an error message. If you exit the program abnormally, by hitting <CTRL>`C`, or if you manage to do something fatal to the input, expect problems. Since your terminal has been reset to allow for the convenient display format, strange things may have happpened to your display, like having all input and output appear on one line. You can either write a small program to reset your terminal to normal or run BSB again and simply type ‘e’ to `E)xit` from the program normally which will reset your VT-100 or terminal emulator. A useful short program is listed below which will reset your terminal after an abnormal exit:

```
#include <stdio.h>
void main(void) {
    printf("%c%s",0x1b,"[1;24r");     /* Restore scrolling */
    printf("%c%s",0x1b,"[f");
    printf("%c%s",0x1b,"[2J");        /* Clear screen */
}
```

6 The Brain State in a Box Model

To use the Brain-State-in-a-Box (BSB) Model, type ‘B’, for B)SB. There now appear new prompt and command lines. There are a number of parameters of the simulation which can be changed if necessary, but the system comes with appropriate defaults for general use.

Figure 3 shows the appearance of the initial BSB screen.

Figure 3. Initial appearance of BSB Screen

——————

```
BSB>P)asses :  16   U)Limit: 1.3000   T)hreshold: 0.5000   Stim. #) : T 1
Mx: Synapses:  95 F)eedback: 0.7000       D)ecay: 0.6500
RESTART









TF BSB  X)ecute C)hange L)ist R)estart V)als Q)uit >
```

——————

The variable x is the system state vector. The actual equation being used is

$$x(t + 1) = \text{Decay} * x(t) + \text{Feedback} * A\ x(t) + \{f(0)\}.$$

Decay is a decay parameter which measures the length of duration of the ‘short term memory’ of the system. (If there was no feedback, activity would decay geometrically with this fraction. Feedback is a parameter multiplying the feedback through the matrix.

If it is enabled, the term f(0) will add the initial state vector to the state vector at every step. It could correspond to a continous input from an earlier processing stage. This is related to what is called ’clamping’ in the Boltzmann machine literature and effectively holds non-zero input vector elements constant so they cannot change in later iterations. This option is used in the DRUGS simulation. It is enabled if ‘O’ or ‘o’ is typed from the command line.

The possible commands in the BSB system are:

C)hange. Change the state vectors in the F File, G File or T File. (see earlier description of this part of the program).

D)ecay allows change of the decay constant in the above equation.

F)eedback allows change of the feedback constants in the above equation.

L)ist. List the vectors in the files. If the display is in FG mode, the F File and the G File will be listed. If the display is in T mode, the T File and the F File will be listed. If more than 16 vectors are present, only sixteen at a time will be shown. The part displayed can be moved forward or backward by typing ‘F’ or ‘B’. L)ist can be left by typing any character other than ‘F’ or ‘B’.

M)ode. Changes the files L)isted from F and G to T and F or vice versa.

O)riginal stimulus. This option adds the original stimulus vector to each iteration. It is closely related to what is called ‘clamping’ in the Boltzmann machine literature since it effectively ensures that initial values never decay away but are always present at high amplitude. A comment will appear on the fifth line of the display if this option is in effect. If O) is typed again, the option is cancelled and the original stimulus will not be added.

P)asses. The number of iterations to be performed before the command line returns and you can cease iterations or change parameters. The default value for P)asses is 16, the number of lines on the scrolling portion of the CRT screen.

Q)uit. Return to the highest command level.

R)estart the simulation with the initial stimulus. Otherwise typing X)ecute will simply have the program continue what it was doing. The left part of the fourth status line will say RESTART is the system will start from zero iterations. Otherwise, the number of completed iterations will appear here.

T)hreshold allows change of the interpretation threshold. When the output of the equation above is generated it is simply a 200 dimensional vector of floating point numbers. In order to see what it is actually doing, this vector is interpreted, i.e. turned back into an ASCII string. This is done by letting every value greater than Threshold be give value +1 and every value less than -Threshold be given value -1. Values nearer to zero than plus or minus threshold are considered uninterpretable, and this particular byte is interpreted as the zero character, ‘_’. Non-printing characters are represented as ‘#’. This thresholding algorithm is only for the operator’s convenience (i.e. it eliminates low amplitude ASCII garbage from the interpretations and makes them easier to read) and it has no effect on the values in the vector that are used for the next iteration. The parity bit is not used in the interpretation and it is not checked for correctness.

U)limit. Allows change of the upper and lower limits of the box. Note upper and lower limit need not be the same.

V)alues. Will display the values of the current state vector in terms of tenths of upper or lower limit. If the value is equal to the upper or lower limit, +L or -L will appear. A digit refers to tenths of the appropriate limit, i.e. +5 means that the value of that element is between 0.5 and 0.6 of the upper limit.

X)ecute. Start iterating the current state vector for P)asses iterations. Will continue from where it stopped if not R)eset.

#) allows choice of the stimulus number to be used for starting the iterations if the system is R)eset.

6.1 Examples

Several examples of iterations using the matrices generated by a disambiguation simulation are shown next. Figure 5 shows the pairs of associations. Figure 6 shows the results of the first 16 iterations and the next 16 iterations on ‘bat ball’. Figure 7 shows the results of the first 16 and second 16 iterations on ‘bat nite’. (The common word, ‘bat’, is ambiguous and the context is able to choose the correct meaning. More details are given in Anderson and Murphy, 1986.)

‘Check’ refers to the number of element values equal to the Upper or Lower limit and is a rough measure of the length of the vector and how rapidly the vector is changing. Positive feedback rapidly increases the number of saturated elements.

Figure 5. Stimulus Set for Disambiguation Example

```
 BSB Neural Net Program. 10-94 B)SB.  C)hange. E)xit. H)elp.
 L)ist. M)ode. N)eurons. R)ead. S)ave. T)hreshold.  W)rite.
 Threshold:  0.5000   F File:    9  G File:    9  T File:   40
 Synapses :  100  Display FG.

    F[ 1]. BaseballGameBat BallDiamd   G[ 1]. BaseballGameBat BallDiamd
    F[ 2]. Vampire MythBat NiteDracu   G[ 2]. Vampire MythBat NiteDracu
    F[ 3]. Animal  LiveBat WingFlyng   G[ 3]. Animal  LiveBat WingFlyng
    F[ 4]. Poker   GameBeerTablCards   G[ 4]. Poker   GameBeerTablCards
    F[ 5]. Tennis  GameCortBallRackt   G[ 5]. Tennis  GameCortBallRackt
    F[ 6]. Dancing RichPrtyBallSocty   G[ 6]. Dancing RichPrtyBallSocty
    F[ 7]. GeoShapeTwoDCrclSqreDiamd   G[ 7]. GeoShapeTwoDCrclSqreDiamd
    F[ 8]. GeoModelTreDSphrBallTetra   G[ 8]. GeoModelTreDSphrBallTetra
    F[ 9]. ExpJewelRichRubyOpalDiamd   G[ 9]. ExpJewelRichRubyOpalDiamd




 C>
```

Figure 6. Bat and Ball After 16 and 32 Iterations

```
BSB>P)asses :  16   U)Limit: 1.3000   T)hreshold: 0.5000   Stim. #) : t 5
Mx: Synapses: 100 F)eedback: 0.2000       D)ecay: 0.9000
RESTART

      1.  ____________Bat Ball_____  Check:   0
      2.  ____________Bat Ball_____  Check:   0
      3.  ____________Bat Ball_____  Check:   0
      4.  ____________Bat Ball_____  Check:   1
      5.  ____________Bat Ball_____  Check:   5
      6.  ____________Bat Ball_____  Check:  10
      7.  _a__________Bat Ball_____  Check:  14
      8.  _a__________Bat Ball_____  Check:  16
      9.  _a__________Bat Ball_____  Check:  21
     10.  _a__________Bat Ball_____  Check:  25
     11.  _a_e________Bat Ball__a__  Check:  45
     12.  _a_e________Bat Ball__a__  Check:  56
     13.  _a_e_______eBat Ball__a__  Check:  67
     14.  _a_e_______eBat Ball__a__  Check: 105
     15.  Ba_e____G__eBat Ball__a__  Check: 108
     16.  Ba_e____G__eBat Ball__a__  Check: 116

TF BSB X)ecute C)hange L)ist R)estart V)als Q)uit >
```

```
BSB>P)asses :  16   U)Limit: 1.3000   T)hreshold: 0.5000   Stim. #) : t 5
Mx: Synapses: 100 F)eedback: 0.2000       D)ecay: 0.9000
It:  16

     17.  Ba_e____G__eBat Ball__a__  Check: 124
     18.  Ba_e____G__eBat Ball__a__  Check: 129
     19.  Ba_e____Ga_eBat Ball__a__  Check: 135
     20.  Ba_e____Ga_eBat Ball__a__  Check: 138
     21.  Ba_e____Ga_eBat BallD_a__  Check: 144
     22.  Ba_e__l_Ga_eBat BallDia__  Check: 149
     23.  Ba_e__l_Ga_eBat BallDia__  Check: 154
     24.  Ba_e__l_GameBat BallDia__  Check: 159
     25.  Base__llGameBat BallDiam_  Check: 161
     26.  Base__llGameBat BallDiam_  Check: 170
     27.  Baseb_llGameBat BallDiam_  Check: 174
     28.  BaseballGameBat BallDiamd  Check: 179
     29.  BaseballGameBat BallDiamd  Check: 181
     30.  BaseballGameBat BallDiamd  Check: 187
     31.  BaseballGameBat BallDiamd  Check: 192
     32.  BaseballGameBat BallDiamd  Check: 197

TF BSB X)ecute C)hange L)ist R)estart V)als Q)uit >
```

Figure 7 Bat and Nite after 16 and 32 Iterations

```
BSB>P)asses :  16   U)Limit: 1.3000   T)hreshold: 0.5000   Stim. #) : t16
Mx: Synapses: 100 F)eedback: 0.2000       D)ecay: 0.9000
RESTART


      1.  ____________Bat Nite_____  Check:   0
      2.  ____________Bat Nite_____  Check:   0
      3.  ____________Bat Nite_____  Check:   0
      4.  ____________Bat Nite_____  Check:   0
      5.  ____________Bat Nite_____  Check:   0
      6.  ____________Bat Nite_____  Check:   1
      7.  ____________Bat Nite_____  Check:   4
      8.  ____________Bat Nite_____  Check:  10
      9.  ________M___Bat Ni_eD____  Check:  21
     10.  _a______M___Bat Ni_eD____  Check:  24
     11.  _a______M_t_Bat Ni_eD_a__  Check:  31
     12.  _a_____ M_t_Bat Ni_eD_a__  Check:  51
     13.  _a_____ M_t_Bat Ni_eD_a__  Check:  71
     14.  _a_____ M_t_Bat Ni_eD_a__  Check: 100
     15.  _a_____ M_t_Bat Ni_eD_a__  Check: 119
     16.  _a_____ M_t_Bat Ni_eD_a__  Check: 123

TF BSB X)ecute C)hange L)ist R)estart V)als Q)uit >

______________
   ______________

BSB>P)asses :  16   U)Limit: 1.3000   T)hreshold: 0.5000   Stim. #) : t16
Mx: Synapses: 100 F)eedback: 0.2000       D)ecay: 0.9000
It:  16

     17.  _a__i__ M_t_Bat NiteD_a__  Check: 127
     18.  _am_i_e Myt_Bat NiteD_a__  Check: 141
     19.  _am_i_e MythBat NiteD_ac_  Check: 147
     20.  _ampire MythBat NiteDracu  Check: 158
     21.  Vampire MythBat NiteDracu  Check: 166
     22.  Vampire MythBat NiteDracu  Check: 169
     23.  Vampire MythBat NiteDracu  Check: 171
     24.  Vampire MythBat NiteDracu  Check: 173
     25.  Vampire MythBat NiteDracu  Check: 180
     26.  Vampire MythBat NiteDracu  Check: 193
     27.  Vampire MythBat NiteDracu  Check: 199
     28.  Vampire MythBat NiteDracu  Check: 200
 >> Fully limited. Finished.



TF BSB X)ecute C)hange L)ist R)estart V)als Q)uit >

______________
```

7 Programming the System

The system as described is primarily interactive. However, it is often convenient, when doing a systematic study of a system, to look at a great many test vectors. This release of the programs does not allow non-interactive programming.

However, Unix script files will allow you to do this conveniently from outside the program. Script files are files of system and program commands that are automatically executed when when the file is ‘run’. The first line should be

```
#!/bin/sh
```

and ‘keyboard’ input should be contained in a separate file. Script files should be executable: run

```
chmod 755 <filename>
```

if necessary.

Suppose we had a set of say, five test stimuli, an autoassociative system, and we wished to iterate every test stimulus 50 times and save the output of the program for future study. The command line would be

```
./bsb testf.stm testg.stm testt.stm test.neu < test.input
```

Figure 8 shows the contents of the response file `test.input`.

When BSB is run and the lines from `test.input` are taken as input to the program. The input files are `L)isted`. We want to use the `B)sb` model. In this case, we want each `P)ass` to consist of 50 iterations and the `O)riginal` input vector to be added after each iteration. Then test stimuli t1 through t5 are successively `X)ecuted`, the system being `R)eset` after each new test stimulus. Then `B)sb` is `Q)uit` and the program `E)xited`.

The output will be be printed on the screen, or can be redirected to a file.

Figure 8

```
l
m
l
b
p
50
o
#
t1
x
r
#
t2
x
r
#
t3
x
r
#
t4
x
r
#
t5
```

```
x
r
q
e
```

8 Demonstration Command Files

Because it is sometimes convenient to be able to regenerate matrices or make stimulus files, we have provided command files for three demonstrations.

We have already given examples of one demonstration, on lexical disambiguation. There are two other neural net demonstrations available. One is a simple drug data base, which contains information about drugs, diseases and microorganisms. There is also a demonstration of an associative system learning Ohm's Law, which can 'reason' about functional dependencies.

In all three cases, shell scripts are available that mean that BSB and ASSOCIAT do not need to be run directly from the command line. Each script calls the appropriate program, supplying command line parameters and 'keyboard' input with the help of additional file. There are three scripts required for each demonstration:

Script	Input	Program	Function
`mkdis`	`mkdis.input`	`bsb`	Generate stimulus files (`*dis.stm`)
`lrndis`	`lrndis.input`	`associat`	Generate matrix (`ndis.neu`)
`dis`	`dis.input`	`bsb`	Use above files in demo
`mkdrugs`	`mkdrugs.input`	`bsb`	Generate stimulus files (`*drugs.stm`)
`lrndrugs`	`lrndrugs.input`	`associat`	Generate matrix (`ndrugs.neu`)
`drugs`	`drugs.input`	`bsb`	Use above files in demo
`mkohms`	`mkohms.input`	`bsb`	Generate stimulus files (`*ohms.stm`)
`lrnohms`	`lrnohms.input`	`associat`	Generate matrix (`nohms.neu`)
`ohms`	`ohms.input`	`bsb`	Use above files in demo

It is just necessary to type the script file names (in the order shown above) and watch the program run. However if you want to see in more detail (and you will) what the programs are doing, then print or display the script and `.input` files for reference, and run the program manually.

It may take a while (10 or 20 minutes on a MicroVAX CPU, a few seconds on an Alpha or Pentium machine) to make the connection matrices using ASSOCIAT. It may be of some value to have complete working 'scripts' for some common operations of BSB and ASSOCIAT. These operations are listed in the command files.

The details of the Ohms and Drugs simulations are described at length in Chapter 16 in James A. Anderson, "An Introduction to Neural Networks", MIT Press, Cambridge, MA. These programs are written so as to be easy to modify for a particular application. Most of the simulations described in Chapters 15, 16, and 17 in the book were done with modified versions of these programs.

9 File Naming and Maintenance

The program associat is self-contained in `associat.c`, however bsb is built from ten modules:

```
bsb.c
bsb_int.c
bsb_list.c
bsb_make.c
bsb_num.c
bsb_proc.c
bsb_readf.c
bsb_readnf.c
bsb_vt100.c
bsb_writef.c
```

After making a change to bsb or associat (or to any of the programs from the earlier chapters), the program will need to be rebuilt. The way to do this is to use the 'make' command. It may be invoked in two ways:

```
make
```

causes all programs to be updated, wherever necessary. Alternatively

```
make <progname>
```

will update just the specified program. For example,

```
make bsb
```

will rebuild just bsb.

Rebuilding is a multistep process, potentially involving several versions of several source code files. The 'make' command only performs such operations as are necessary, and so rebuilding is rarely carried out from scratch.

However should you wish to rebuild all files from scratch, the best way is to type

```
touch *.c
make
```

The first updates the times and dates of all C files, and the second invokes make.

10 Acknowledgement

This research was primarily supported by National Science Foundation Grants BNS-82-14728 BNS-85-18675, and BNS-90-23283 to James Anderson, Department of Cognitive and Linguistic Sciences, Brown University, Providence, RI, 02912. Please acknowledge these grants if you make use of these programs in published material.

11 References

J.A. Anderson (1993), The BSB Model, Ed. M. Hassoun, Associative Neural Networks, New York, NY: Oxford University Press

J.A. Anderson (1994), An Introduction to Neural Networks, Cambridge, MA: MIT Press.

J.A. Anderson, Neural models for cognitive computation. IEEE Transactions: Systems, Man, and Cybernetics. 1983, **SMC-13**, 799-815. (Reprinted in V. Vemuri (Ed.), Artificial Neural Networks: Theoretical Concepts, Washington, DC: Computer Society Press of the IEEE, 1988.)

J.A.Anderson, Cognitive Capabilities of a Parallel System. In E. Bienenstock, F. Foglemann, and G. Weisbuch. (Eds.) Disordered Systems and Biological Organization, Berlin: Springer, 1986.

J.A. Anderson and M. Mozer, Categorization and selective neurons. In: G. Hinton and J. Anderson (Eds.), Parallel Models for Associative Memory. Hillsdale, New Jersey: Erlbaum Associates, 1981. Revised Edition, 1988.

J.A. Anderson and G.L. Murphy, Psychological Concepts in a Parallel System. In J.D. Farmer, A. Lapedes, N. Packard, and B. Wendroff. (Eds.) Evolution, Games, and Learning. New York: North Holland, 1986. (With Gregory L. Murphy).

J.A. Anderson, J. Silverstein, S. Ritz, and R. Jones, Distinctive features, categorical perception and probability learning: some applications of a neural model. Psychological Review, 1978, **85**, 597-603. (Reprinted in J.A. Anderson and E. Rosenfeld (Eds.), Neurocomputing, Cambridge, MA: MIT Press, 1988).