ROUTINES LIBRARY
FOR
MULTIPLE NEURAL NETWORKS MANAGEMENT

(for C programs)

Version 3.2

Daniele Denaro
Via Sergio Forti , 47
00144 - ROMA (ITALY)
Tel. 39-6-5292189

Internet :
Mail: daniele@caio.irmkant.rm.cnr.it
ftp anonymous : kant.irmkant.rm.cnr.it
pub/source/nrlib

INDEX

1 - Introduction

This package contains a shell for neural networks real applications.
NRLIB32 and NREBP32 are shareware programs. You can use them to test and verify their functionality. If you will be satisfied register you using form in "readme.txt" file.
NRLIB32 is a software library and NREBP32 a companion program.
This software was developped in C language and is available for any computer; but in this package is also provided compiled version for 286->486 PC systems (with math. coprocessor).

This library provide 50 functions for manage multiple neural networks in C programs.
We can :
- Create in memory one or more nets (different too).
- Save and load all nets on/from a file
- Net compute (input propagate)
- Net train with EBP (Error Back Propagation)

or many other manipulations.

Each net can have any number of layers (to 100 layers).
Each layer can have any number of nodes.
Each node can have any number of links with others nodes.

The nodes have progressive number in a net.
You can choose a different activation function for each node.
There are ten types of activation function : 6 defined and 3 for user definition.
For each node it's also possible to define another function named action, which is called after the activation function.

Range of principals values:
- Net number 1 -- 1000 (in this release)
- Layer number 0 -- 99
- Node number 1 -- 7FFFFFFF (in hex)

Files included in the package:
- README.TXT Abstract and orders details
- NRLIB32.H Include file for library use . Functions are described like in a manual.
- NRLIB32.OBJ Object file that you must link with your program (286 min., math. copr.)
- NRLIB32.C Library source
- NREBP32.EXE Complete program (interactive) for EBP net use
- NREBP32.C Source of NREBP32
- EXAMPLE1.C Example of NRLIB32 use (imitation metaphor)
- EXAMPLE2.C Example of NRLIB32 use (genetic algoritm)
- EXAMPLE1/2.EXE (compiled versions)
- NRLIB32.PS This document in postscript
- NRLIB32.TXT " in ASCII format (without figures)
- NETTRAIN.DAT Trainer file for XOR task
- NNETSINP.DAT Example of nets save file
- NNETXOR.DAT Example of nets save file (trained on XOR task)

2 - Neural network architecture

2.1 - Structure

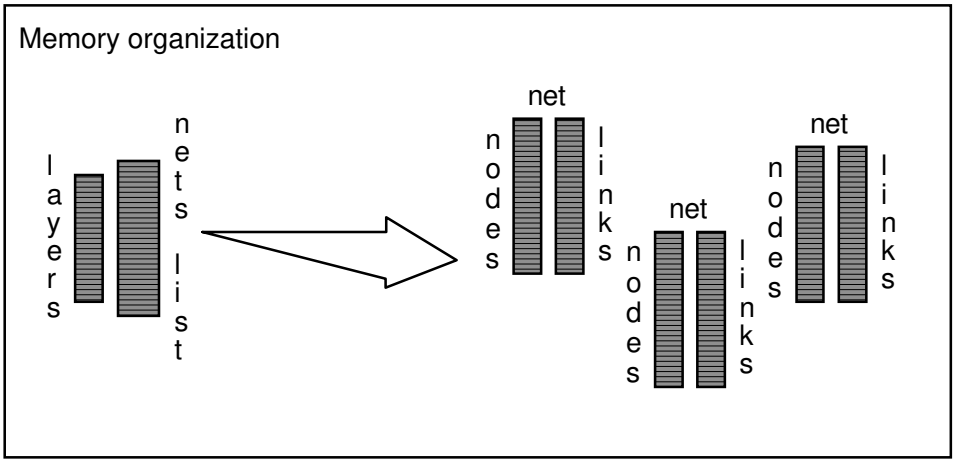
A neural network is a logical structure composed by lots of nodes (neurons) and by links (synapses) between them.
Nodes accept inputs from other nodes or external environment , and propagate this sum to linked nodes . Input propagation is filtered by a saturation function called activation function and by a scaling factor that corresponds to the weight of a particular link .

Links structure and weights values identify the net behaviour , i.e. the status of output nodes as answer of a stimulus on input nodes. Identified the proper architecture , we can modify the weights to train a net to the behaviour selected.

For more details on neural network architecture theory and training methods , we remand to specialized literature.

This library allows to define any neural net architecture, but the macro-functions for net management are made for a forward layered net, to be trained with EBP (Error Back Propagation) rule.

The principal characteristic of this library is that it uses a sequential list of nodes with their parameters (i.e. input , status , pointer to activation function , pointer to links etc.), and a sequential list of all links; this structure allow flexibility , computing speed and memory optimisation at the same time.
For this reason each node can have different activation function or different bias as well as different links number.

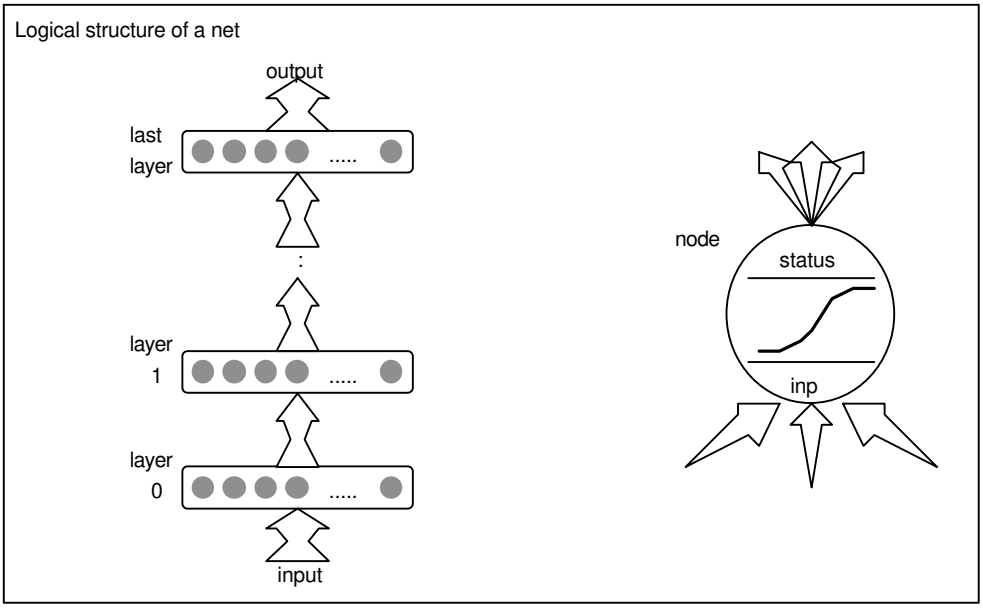


Over this organization there is another lists structure. This structure allows to describe the whole of nets ; moreover it logically organize the net nodes in layers.
Pointers are often used for more quickness.
Memory is dinamically allocated to according nets dimension and availability.

In theory, layers information can be ignored , because the original net structure in memory dasn't need layers information to run along the net.
But, even if there are routines for single node access and manipulation , more helpful routines are provide for use in forward nets with layers organization . In fact , forward propagation of input and EBP training functions are made for multilayered forward nets.

Therefore a net is typically organized in layers. First layer is layer 0 and corresponds to net input. It is possible to have up to 100 layers (last layer = 99). Last layer correspond to net output.

If we want the net answer to an input stimulus, we must copy input values (in normalized floating format) to input buffer of input nodes , and , after net computing , we can receive output answer on reading status buffer of output nodes.

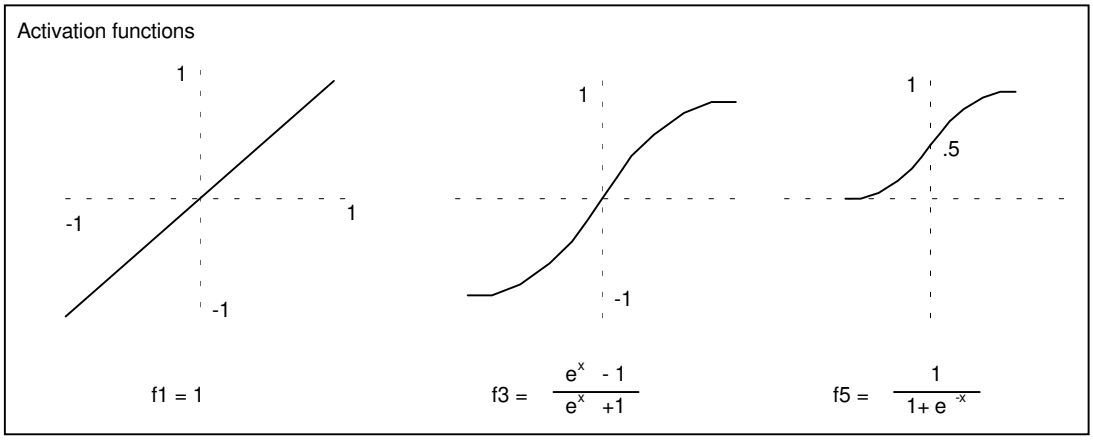


Net creation routines require an array of values for each node characteristic . Each value is valid for all nodes of a layer , i.e. all nodes of a layer are created equals. And each node is linked with each node of next layer. Node bias are 0 for default and links weights are randomicaly assigned , but you can modify them by single node functions if need it.

In the net elaboration phase , input node buffer is modified by the activation function and the result is put in status node buffer (also called activation).

2.2 - Activation functions

There are 6 types of activation function provided by the library. The principals are:
- Linear function (simply move input to status) : name f1 (useful for input nodes)
- Positive/negative sigmoid : name f3 (by table) or f13 (calculated)
- Only positive sigmoid : name f5 (by table) or f15 (calculated)



Functions f3 and f5 are realized by look table , with 0.01 precision , for more speedness.
Also their derivatives are realized by look table.
Functions f13 and f15 (and their derivatives) are calculated in simple precision ,for more accuracy.
Names are transformed in pointers on creating or loading net in memory.
Other four function names (f7,f8,f9,f10) are recognized but are provided by user if needed (see next paragraph for more details).
Function f2,f4,f6 have been holded for compatibility with previous version and can be seen as reserved names .

In addiction , we can define another function for a node. This function is called "action" in this library and can be used for more sophisticated applications as a Hebs training rule for example.
This function is user provided and the names recognised are a1 -- a6.
If function name is "0" or an unknown name , it is ignored in node computing.
If it is defined, it is valuated after activation function.

Net training is usually performed by EBP rule. Library routines for this purpose are very simple to use. They start with a difference < out calculated - out expected> loaded in error node buffer of last layer and back propagate error to first layer on correcting links weights at the same time.
For back error propagation they use correspondent derivative of activation function present on each node .

2.3 - User activation function

As described in previous paragraph, names allowed are: f7,f8,f9,f10 .
They are defined in NRLIB32.H file as null functions.
If user want, he may to redefine this functions in this file or in his program.

User activation functions have four parameeters: ex. f7(int net,LONG nn, int f,float *d)
- net number
- node number
- flag function/derivative (0 function , 1 derivative)
- pointer to derivative return value buffer

Activation functions are called by library function for forward phase and back propragation phase.
For this reason flag is used to distinguish between forward (normal) phase and back propagation phase. In forward phase , activation function is used to modify input and put result in status for each node with this function. In back propagation phase, derivative is instead used as returned value in buffer d.

In other words, if function is called with flag f = 0 , we must modify node input , write result in node status and blank node input (for next cicle) ; this is forward computation.
If function is called with flag f = 1, we must calculate activation function derivative for actual status value (we must determine derivative in function of status value because in back propagation phase node input is inavaillable ; it has been blanked in forward phase).
If you don't use Error Back Propagation algoritm you can put 0 on derivative buffer.

Definition example :
Saturation : $y=1- e^{-x}$ $dy/dx = e^{-x} = 1- y$

```
f7(int net , LONG nn , int f, float *d)
{
  float buff;
  if (f == 0) { rinp(net, nn, &buff);       /* read node input  (rinp : library function)  */
                buff=1-exp(-buff) ;         /* modify input */
                wstat(net, nn, buff);       /* write node status (wstat : library function) */
                winp(net, nn, 0);           /* blank node input  (winp : library function)  */
                return ; }
   if (f == 1) {rstat(net,nn,buff);         /* read node status (rstat : library function) */
                *b = 1- buff ;              /* derivative value */
                 return ; }
}
```

Action functions (a1 -- a6) have instead two parameters : net number and node number ; and have free definition.

3 - Functions list

For details on functions and their parameters you can see the NRLIB32.H file, where they are described .

Load and save nets :
loadnet (char *fileinp);
savenet (char *fileout);

Create nets :
createstruct (tn, vlay[]);
createnet (net,LONG vn[],char vf[][10],char va[][10]);

Create all equals nets :
createallnet (tn, nlay,LONG vn[],char vf[][10],char va[][10]);

Reinitialise net :
ininet (net);

Copy net to net:
copynet (nta, ntb);

Release memory allocated for nets :
cancnets ();

Modify global parameters :
param (char *name, char *val);

Compute net :
produce (net,char *finpdat, char *foutdat);
compute (net,float vinp[],float vout[],float vdat[]);
propagate (net, laya, layb);

Train net :
train (net,char *filetrain);
learn (net,float verr[]);
backprop (net, laya, layb);
backproperr (net, laya, layb);
cancerr (net, laya, layb);

Read or modify nets parameters:

rmnet (*totnet);
wmnet (totnet);
rmlay (net, *mlay);
totnn (net, LONG *tnode);
wmlay (net, mlay);
rlay (net, nlay,LONG *sn,LONG *en);

Read or modify nodes parameters of a layer :

rstatl (net, nlay,float stat[], dim);
wstatl (net, nlay,float stat[], dim);
rinpl (net, nlay,float inp[], dim);
winpl (net, nlay,float inp[], dim);
rerrl (net, nlay,float err[], dim);
werrl (net, nlay,float err[], dim);
rbiasl (net, nlay,float bias[], dim);
wbiasl (net, nlay,float bias[], dim);
rwgtl (net,LONG nn, nlayp,float wgt[], dim);
wwgtl (net,LONG nn, nlayp,float wgt[], dim);
rwgtil (net,LONG npp, nlaya,float wgt[], dim);
wwgtil (net,LONG npp, nlaya,float wgt[], dim);

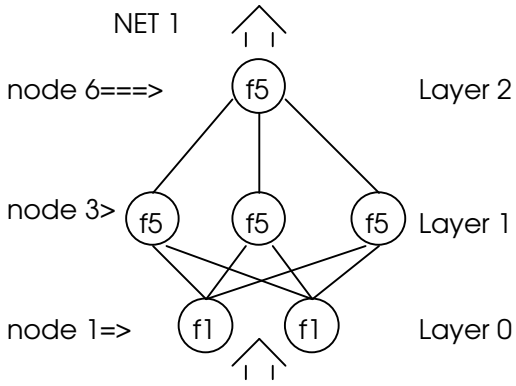
Read or modify single node parameters :

rstat (net,LONG nn,float *stat);
rbias (net,LONG nn,float *bias);
rinp (net,LONG nn,float *inp);
rerr (net,LONG nn,float *err);
rfun (net,LONG nn,char fun[10],char ac[10]);
wstat (net,LONG nn,float stat);
winp (net,LONG nn,float inp);
werr (net,LONG nn,float err);
wbias (net,LONG nn,float bias);
wfun (net,LONG nn,char fun[10],char ac[10]);
rnconn (net,LONG nn,LONG *ncl);
rconn (net,LONG nn,LONG cl,float *wgt ,LONG *np);
wwgt (net,LONG nn,LONG cl,float wgt);
wnodp (net,LONG nn,LONG cl,LONG np);
wnconn (net,LONG nn,LONG ncl);

Version 3.2 differs from version 3.1 by bugs corrections in functions rwgtl and wwgtl ; and activation functions f13 and f15 added.

4 - Example of library use

Imagine that we wont to define a net of 3 layers as described in picture.



We must include NRLIB32.H file and link our program with NRLIB32.OBJ. For DOS system memory model is large.

Define array for number nodes , activation function and action function for each layer:

```
  LONG vn[3];char vf[3][10], va[3][10];
```

and initialize it

```
  vn[0]=2; strcpy(vf[0],"f1"); strcpy(va[0],"0");   layer 0 : 2 nodes with linear activation
  vn[1]=3; strcpy(vf[1],"f5"); strcpy(va[1],"0");   layer 1 : 3 nodes with sigmoid activation
  vn[2]=1; strcpy(vf[2],"f5"); strcpy(va[2],"0");   layer 2 : 1 node with sigmoid activation
```

now create net

```
   createallnet(1,2,vn,vf,va)     single net with maximum layer 2 ;
                                  each layer characteristics done by arrays vn,vf,va
```

or (alternative way)

```
    int vlay[2];     define array of maximum layer of each net
    vlay[1]=2;       maximum layer of net 1 (vlay[0] is not used)
    createstruct(1,vlay);        define structure that describe only one net
    createnet (1,vn,vf,va);      create net 1 of structure
```

Suppose that we wont to trainer this net to learn XOR function, and save net trained.

```
     param("ne","1000");         define 1000 epocs for training
     train(1,"nnetxor.dat");     train net 1 with trainer file "nnetxor.dat"
```

File nnetxor.dat contain records with various input-ouput coupled according to XOR function.

Save net trained, on file "xorout.dat"

```
      savenet("xorout.dat");
```

If we wont to reuse this trained net in another program we can load it simply.

```
      loadnet("xorout.dat");
```

And use it as a filter function

```
      produce(1,"fileinp.dat","fileout.dat");
```

Where fileinp.dat contain input values and fileout is writed by net answers.

5 - File format of nets saving

First record		"totnets"/max_net_num/total_layers/
Other record		"nnet"/net_num/total_nodes/total_links/
	or	"lay"/layer_num/
	or	"node"/node_num/input/bias/stat/fun_name/ac_name/
	or	"conn"/pointed_node;weigth/...../pointed_node;weigth/

example :

```
totnets/1/3/

nnet/1/6/9/
lay/0/
  node/1/0.0000/0.0000/0.0000/f1/*/
    conn/3;-0.3916/4;-0.3968/5;-0.1319/
  node/2/0.0000/0.0000/0.0000/f1/*/
    conn/3;-0.3734/4;-0.1154/5;-0.2262/
lay/1/
  node/3/0.0000/0.0000/0.0000/f5/*/
    conn/6;0.0296/
  node/4/0.0000/0.0000/0.0000/f5/*/
    conn/6;-0.2434/
  node/5/0.0000/0.0000/0.0000/f5/*/
    conn/6;0.1603/
lay/2/
  node/6/0.0000/0.0000/0.0000/f5/*/
    conn/
```

- it's possible to continue record on more lines
- each field ends with / character
- input,bias,status,weigt are float , other integer
- fun_name and ac_name are strings

6 - NREBP program

NREBP program is included in the package. This program is an interactive software that allows us to create or load one or more nets and activate or train them.

It has 3 way of using :
- like a filter
- interactive nets management
- batch train

The 3 modalities are decided by parameeter fu (fu = 0,1,2).

In the first use , it receives an input from standard input and evaluates a net producing an output on standard output. This is the default functionality.
At starting phase , it load nets from default nets file or user provided file.

For example , if command is :

NREBP32 /fu=0 <stimul.dat >results.dat (or NREBP32 <stimul.dat >result.dat)

it load nets from default nnetsinp.dat and activate net 1 with input read from file stimul.dat and write output to file results.dat.

In the second use , it appears as menu driven software. It allow us to create or load nets , compute or train them and save nets to a file.
Interactive interface dasn't need grafic device but simply text line device with ANSI escape control characters, for better software portability.

Example of command :

NREBP32 /fu=1

To testing this program, net definition file "nnetsinp.dat" and trainer file "nettrain.dat", for XOR task, are included in package. You can load net and train it. Or you can load trained net by "nnetxor.dat" file, and verify corrects answers.

Third use allows us to train a nets in batch mode for a set number of epochs .

For example, if command is:

NREBP32 /fu=2 /ne=5000

it loads nets from default file nnetsinp.dat and trains net 1 reading trainer file nettrain.dat for 5000 times; then it writes last averege quadratic error and saves nets to default file nnetsout.dat.
For more details ask help typing : NREBP32 /h

7 - Version 4 enhancements

This version is already available ; for any request contact me (see file README.TXT).
Functions available increase to 59.
More complex logic structure is definable for a net.
With simple routines like "createnet", "compute", "learn" etc. , we can manage net with memory contest layer or recursive layer.

Each layer can be linked with more layers (to 10) , no more with following layer only.
Each layer can have a pointer to a layer that realize a memory contest.
Each layer can have a pointer to a layer that realize a duplicated layer.

Before starting compute phase , library routine, copies nodes status of a layer to input buffer of the contest memory layer correspondent.
For layer duplicated , compute routine , copies nodes status of the layer to the nodes status of layer duplicated at the same time when compute it.