NAME

tlearn – simulator program for neural networks

SYNOPSIS

tlearn [options]

DESCRIPTION

tlearn is a simulator program which allows you to dynamically (at run-time) configure networks with a wide range of architectures. It implements the back propagation of error learning algorithm, the Williams & Zipser algorithm for fully-recurrent networks, and simple recurrent networks with fixed copy-back links.

tlearn is run non-interactively as a command with command-line options. These options specify such things as the network description file, data pattern files, learning rate, number of learning cycles, initial random seed, etc. Typically, lengthy training sessions are run asynchronously (i.e., backgrounded with &). After training, **tlearn** saves the network weights and biases in a file. This file can then be loaded back in for testing purposes.

OPTIONS

There are a number of arguments may be given in the command line when the program is invoked. The first two arguments are obligatory.

-f <fileroot>

The fileroot is the part of the filename in the five input files which remains constant. For example, the argument *-f xor* imples the files *xor.cf, xor.data, xor.teach* [Obligatory]

-s <number>

Run for <number> of sweeps. A sweep consists of single forward activation cycle and (if learning occurs) a backward weight adjustment pass. NOTE: the default presentation schedule is to select from the .data and .teach files in sequential order unless the -R flag is specified.

When the program terminates, the weights and biases will be saved in a weights file which has the form *<fileroot>.<number>.wts,* where the fileroot is as given with the -f option, <number> is the number of sweeps that went into producing this weights file, and .wts is a literal extension. [Obligatory]

-m <number>

Set the momentum to <number>. [Default: 0.0]

-l <wts.file>

Load in a weights file. The name of the weights file must be completen completely; it need not folow the <fileroot>.<#>.<wts> pattern.

-n <number>

The number of clock ticks per intput vector [Default: 1]

-r <number>

Set learning rate to <number>. [Default: 0.1]

-C <number>

Checkpoint every <number> of iterations. If a large number of sweeps are being run, it is often a good idea to "checkpoint" the simulation by periodically saving the weights file. This allows you to monitor the progress of the learning, and also can be used to restart the program if it should terminate prematurely.

-E <number>

Report average rms error in .err file, averaged over and recorded every <number> sweeps.

-S <number>

Set seed for random number generator to <number>. [Default: time of day]

-X Read auxiliary reset file for reset times. Network activations will be reset to 0.0 immediately prior to the cycles indicated in this file. The filename must be of the form <fileroot>.reset. [Default: no

reset]

-M <number>
Stop sooner than -s sweeps if rms is less than number. [Default: 0.0]

-R Present inputs in random order. This requires a line in the teach file for every line in the .data file. [Default: sequential]

-U <number>
Update weights every <number> of sweeps. [Default: 1]

-V Verify output activations on each sweep. No learning occurs when this flag is given. Tlearn will report the values of output nodes for -s sweeps, processing input patterns sequentially.

-P Probe activations of selected nodes on each sweep. No learning occurs if this flag is given. Tlearn will report the values of nodes marked as "SELECTED" in the .cf file for -s sweeps, processing input patterns sequentially.

-H <1 or 2>
Use cross-entropy error instead of sum-squared error. If 1 is specified, cross-entropy is used, but the error report (-E) will be in terms of rms error; if 2 is specified the error report (-E) will be in terms of the (costlier to compute) cross-entropy statistic.

-B <#> Initialize biases to output units to be approximately #; an initial value of -2.5 is often useful.

-L Use temporally recurrent learning (Williams/Zipser algorithm). [Default: off]

-I Ignore input values during extra ticks. Relevant if -L specified.

-T Ignore target values during extra ticks. Relevant if -L specified.

FILES

tlearn requires three input files: *network configuration, input patterns, output (teacher) patterns*. An additional input file may be used to specify a reset schedule. **tlearn** may also create output files for weights, error, and a command history. All files should begin with the same name; this is referred to as the *fileroot*. The different files are distinguished by having different extensions (where an extension consists of a period followed by a fixed designator). The files have the following extensions:

.cf	**network configuration [required]**
.data	**input patterns [required]**
.teach	**output (teacher) patterns [required]**
.reset	**reset schedule**
.wts	**saved weights after training**
.err	**error report**
.cmd	**history of command-line options for previous**

Thus, test.cf, test.data, test.teach are possible names of input files for a given simulation. The format and function of each of these files is discussed below.

.cf file This file describes the configuration of the network. It must conform to a fairly rigid format, but in return offers considerable flexibility in architecture.

There are three sections to this file. Each section begins with the keyword in upper case, flush-left. The three section keywords are **NODES:**, **CONNECTIONS:**, and **SPECIAL:**. Note the colon. Sections must be described in the above order.

NODES:
This is the first line in the file. The second line specifies the total number of nodes in the network as "nodes = #". Inputs do NOT count as nodes. The total number of inputs is specified in the third line as "inputs = #". The fourth line specifies the number of nodes which are designated as outputs according to "outputs = #'. (Note that these two lines essentially give the lengths of the .data and

.teach vectors.) Lastly, the output nodes are listed specifically by number (counting the first node in the network as 1) in the order that the .teach information is to be matched up with them. The form of the specification is "output nodes are <node-list>". (If only a single output is present one can say "output node is #"). If no output nodes are present, this line is omitted. Spaces are critical.

Node number can be important for networks in which there are fixed copy-back links. Copy-back links allow for saving of node activations so that they can be used on the next sweep. Because node activations are calculated in ascending order, with the order determined by the number of the node, it is important that node activations be saved *after* they are calculated. It is also important that a unit which receives input from a node which is serving as a state/context unit (and has thus storing some other nodes activation from the previous time cycle) calculate its activation before the state/context unit gets updated on the current sweep. Both considerations lead to the following rule of thumb: Any node receiving input from a state/context unit must have a *lower* node number than the state/context unit. This is illustrated in the example .cf file at the end of this document.

CONNECTIONS:
This is the first line of the next section. The line following this must specify the number of groups, as in "groups = #" (All connections in a group are constrained to be of identical strength; e.g., as in the T/C simulation in Chapter 8/PDP-I; in most cases groups = 0.) Following this, information about connections is given in the form:

<node-list> from <node-list> [= <fixed> | <group #> | <min & max>]

[It is also possible to say:

<node-list> from <node-list> = min & max fixed

or

<node-list> from <node-list> = min & max fixed one-to-one

This last form is used, e.g., when node 1 is fed from node 4, node 2 is fed from node 5, and node 3 is fed from node 6, as opposed to the ususal case of node 1 being fed from nodes 4-6, node 2 being fed by nodes 4-6, and node 3 being fed by nodes 4-6.]

A <node-list> is a comma-separated list of node numbers, with dashes indicating that intermediate node numbers are included. A <node-list> contains NO SPACES. Nodes are numbered counting from 1. Inputs are likewise numbered counting from 1, but are designated as "i1", "i2", etc. Node 0 always outputs a 1 and serves as the bias node. If biases are desired, connections MUST be specified from node 0 to specific other nodes (not all nodes need biased). Groups must be labeled by integers ascending in sequence from 1. It is also permissible to say

<group #> = <min & max>

provided that the group has already been completely defined.

SPECIAL:
This is the first line of the third and final section. Optional lines can be used to specify whether some nodes are to be linear ("linear = <node-list>"), which nodes are to be bipolar ("biploar = <node-list>"), which nodes are to have no delay ("not_delayed = <node-list>"; not_delayed is the default for all nodes unless the -L flag is used), which nodes are selected for special printout when the -P flag is given ("selected = <node-list>"), and the initial weight limit on the random

initialization of weights ("weight_limit = <node-list>"). Again, SPACES ARE CRITICAL. Example .cf files are given at the end of this document for several network architectures.

.data file

This file defines the input patterns which are presented to tlearn. The first line must either be "distributed" (the normal case) or "localist" (when only a few of many input lines are nonzero). The next line is an integer specifying the number of input vectors to follow. Since exactly one input vector is used for each time step, this is equivalent to specifying the number of time steps. The remainder of the .data file consists of the input. These may be input as integers or floating-point numbers. In the (normal) "distributed" case, the input is a set of vectors. Each vector contains NI floating point numbers, where NI is the number of inputs to the network. Note that these input vectors are always used in the exact order that they appear in the .data file (unless the -R flag is specified). In the "localist" case, the input is a set of <node-list>'s (defined below) listing only the numbers of those nodes whose values are to be set to one. Node lists follow the convetions described in the .cf file.

EXAMPLES:

```
distributed
4
0   0
1   1
0   1
1   0

localist
4
1
2,3
2
1-3,5
```

.teach file

This file is required whenever learning is to be performed. As with the .data file, the first line must be an either "distributed" (the normal case) or "localist" (when only a few of many target values are nonzero). The next line is an integer specifying the number of output vectors to follow. Unlike the .data file, there does not need to be an output vector for every time step. Instead, each output vector ∗must be∗ preceded by an integer time stamp indicating the time at which this output vector is to become effective. An output vector is kept as the target until the next time stamp in the .teach file.

In the (normal) "distributed" case, each output vector contains NO floating point numbers, where NO is the number of outputs in the network. An asterisk ("∗") may be used in place of a floating point number to indicate a "don't care" output. Note that the time stamps must appear in ascending order.

In the "localist" case, each output vector is a single number designating the single node whose target is a one as opposed to a zero.

EXAMPLE:

```
distributed
4
0   .1
1   .9
2   *
5   0.
```

.reset file
This file is required whenever the -X flag is used. As with the .teach file, the first line must be an integer specifying the number of time stamps to follow. Each time stamp is an integer specifying the time step at which the network is to be completely reset. As with the .teach file, the time stamps must appear in ascending order.

EXAMPLE:
2
0
3

<fileroot>.<runs>.wts
At the conclusion of a tlearn session, the results of training are saved in a "weights file." This file name incorporates the fileroot, the number of learning cycles (runs) which resulted in this network, and ends with "wts" as the literal extension. This file contains weights and biases resulting from training. Weights are stored for every node (except the bias node, 0), from every node (including the bias node). A sample weights file for the XOR (2x2x1) network would be: file.

NETWORK CONFIGURED BY TLEARN
weights after 10000 sweeps
WEIGHTS
TO NODE 1
-6.995693
4.495790
4.495399
0.000000
0.000000
0.000000
TO NODE 2
2.291545
-5.970089
-5.969466
0.000000
0.000000
0.000000
TO NODE 3
4.426321
0.000000
0.000000
-9.070239
-8.902939
0.000000
This file can also be produced by requesting periodic checkpointing (either in order to recreate intermediate stages of learning, or to avoid having to re-run a lengthy simulation in the event of premature termination). This weights file can be loaded into tlearn in order to test with a trained network.

.err file
If error logging is requested, a file will be produced containing the rms error, saved at user-specifiable intervals.

.cmd file
This file records the command line options when the program is invoked (the file is written in append mode). This is useful in documenting the value of program variables.

EXAMPLE .cf FILES

EXAMPLE 1

This illustrates a feed-forward network which implements a 2x2x1 XOR network (cf. Chapter 8/PDP-I). Notice that in tlearn, the 2 inputs are NOT nodes; the network itself has only the 2 hidden units and 1 output unit. (There are still learnable connections from the 2 inputs to the 2 hidden units.)

```
NODES:
nodes = 3
inputs = 2
outputs = 1
output node is 3
CONNECTIONS:
groups = 0
1-3     from    0
1-2     from    i1-i2
3       from    1-2
SPECIAL:
selected = 1-2
weight_limit = 1.0
```

EXAMPLE 2 This illustrates a network that receives 3 inputs, has 4 hidden units, 2 output unit, and 4 copy-back units; each copy-back unit receives the activation of the corresponding hidden unit at the prior cycle. Notice that the copy-back units are linear, receive no bias, and have fixed downward connections from the hidden units. In the number scheme, i1-i3 designate the 3 inputs; nodes 1-4 are the hidden units; nodes 5-6 are the output units; and nodes 7-8 are the copy-back (state/context) units.

```
NODES:
nodes = 10
inputs = 3
outputs = 2
output nodes are 5-6
CONNECTIONS:
groups = 0
1-6                     from0
1-4                     fromi1-i3
1-4                     from7-8
5-6                     from1-4
7-8                     from1-4=1. & 1.  fixed  one-to-one
SPECIAL:
linear  = 7-8
weight_limit = 1.
selected = 1-4
```

EXAMPLE 3 This illustrates a network which receives 9 inputs, has 3 hidden units (1-3) and 1 output unit (4). The 3 hidden units have limited receptive field; each one receives from only 3 of the inputs. In addition, the connections are grouped (i.e., trained to assume the same values), thus benefiting by the learning that other nodes in the group does (e.g., even when deprived of input). The result is that each hidden unit has 3 input different weights; each of the 3 weights is similar weight leading into the other 2 hidden units. This scheme is similar to the T/C network in Chapter 8/PDP-I. confined to limiting values. Finally, weights are confined to the range -5/+5.

```
NODES:
nodes = 4
```

```
inputs = 9
outputs = 1
output node is 4
CONNECTIONS:
groups = 3
1-4                             from  0
1                               from  i1  =  group 1
1                               from  i2  =  group 2
1                               from  i3  =  group 3
2                               from  i4  =  group 1
2                               from  i5  =  group 2
2                               from  i6  =  group 3
3                               from  i7  =  group 1
3                               from  i8  =  group 2
3                               from  i9  =  group 3
4                               from  1-3
group 1  =  -5 & 5
group 2  =  -5 & 5
group 3  =  -5 & 5
SPECIAL:
selected = 1-3
weight_limit = 0.1
```

FILES

?.cf, ?.data, ?.teach, ?.reset, ?.wts, ?.err, ?.cmd, /usr/src/exp_table