# Reference Manual for SDCC

© 1997-2006 Thomas R. Shultz, All rights reserved

## About sdcc

The cascade-correlation algorithm was invented by Scott Fahlman at Carnegie Mellon University. The enclosed software includes both Fahlman's Lisp code (here referred to as cc) and the LNSC user shell (referred to as sdcc) designed to make it easier to use Fahlman's code in designing, running, and analyzing simulations.

SDCC is a user shell for the 11 May 1991 version of cc that includes an option to run in SDCC (Sibling-Descendant Cascade-Correlation) mode. The sdcc program enables three types of input-output patterns (training patterns, test patterns, and multiple test patterns); two times of pattern creation (once at the start vs. fresh patterns every epoch); two sources of patterns (stored in file vs. created from code); and three types of recorded data (errors, unit activations, and contributions). Recording of unit activations is not available for multiple test patterns.

This manual is organized into five sections concerning loading the program, setting up a run, executing a run, analyzing the results of a run, and performing contribution analysis.

If you are new to cascade-correlation, then you may be interested to know first about the necessary minimum aspects of sdcc. At a minimum, you call set-patterns to create the initial network and then call train to train the network. Even before you call set-patterns, you may want to call seed-random to ensure fresh random numbers and set-path to enable easy access to pattern files or locations for storing results. Then you can add in various other parameter settings or procedure calls as needed.

## Loading Program

Open Common Lisp. Load a compiled version of cc and then load a compiled version sdcc. The compiled version of cc must be loaded before sdcc can be compiled. You can safely ignore any warnings that occur during these loadings.

## Setting a Directory Path

It is often desirable to set a directory path for saving the results of simulations. This enables fairly brief calls to procedures that require arguments of file paths. In general, a directory path must be a double quoted string with directory names separated by specified characters.

Typically, one would define a global variable called *path* that could be initialized to nil.

```
(defvar *path* nil)
```

Then *path* could later be set to the particular file path desired. For example, in Windows a path could be set by:

```
(setq *path* "c:\\hard\\models\\cascor\\problems\\*")
```

In running under Unix, an analogous path setting might be:

```
(setq *path* "/home/lima/shultz/models/cascor/problems/")
```

In running under Macintosh, a path could be set with:

```
(setq *path* "hard:models:cascor:problems:*")
```

Then *path* could be used whenever a file path needs to be specified.

## Setting up a Run

(**seed-random**) procedure

Seeds the random number generator from the internal time.

***encoder*** global variable

If t keep input-output weights at 0. Use with encoder networks that reproduce their inputs on their outputs. Otherwise hidden units will not be required to abstract useful functions. Default value is nil.

***sdcc**** global variable

If t allows candidate hidden units to be siblings (i.e., installed on existing top-most layer) instead of being descendants (i.e., installed on new top-most layer). Otherwise candidates are always descendants. Default value is nil.

***sibling-factor*** global variable

Factor multiplying the correlation of each sibling candidate with network error. Default value of 1.0.

***descendant-factor*** global variable

Factor multiplying the correlation of each sibling candidate with network error. Default value of 0.8. This is to penalize recruiting descendants a bit. Research has found that 0.8 is often an optimal value.

(**random-inputs** n &optional high) procedure

Sets the stage for random inputs which the network must then learn to ignore. n is the number of random inputs required; high is highest value of random inputs. Sets the global variables of *random-inputs*, *n-random-inputs*, and *high-random-input* to true, n and high, respectively. The last of these is set to 1 + high if high is an integer. The default value for high is 1. If you want random input units, call random-inputs before calling set-training-patterns.

***random-inputs*** global variable

If t, add random inputs at each epoch. Default value is nil. Reset by calling random-inputs.

***n-random-inputs*** global variable

The number of random inputs. Reset by calling random-inputs. Default value is nil.

***high-random-input*** global variable

The highest random input. Reset by calling random-inputs. Default value is nil.

***uniform-input-noise*** global variable

The range of random distortion added to each training and test input at each epoch during training. The value added is uniformly distributed between -*uniform-input-noise* and +*uniform-input-noise*. Default value is nil. If you want to use *uniform-input-noise* setq it to some value before calling set-training-patterns.

```
(setq *uniform-input-noise* .01)
```

***some-uniform-input-noise*** global variable

Works like *uniform-input-noise* except that only some specified inputs are distorted. List of maximum amount of noise followed by list of position numbers in sub-vector to be noised. Default value is nil. If you want to use *some-uniform-input-noise* setq it to some value before calling set-training-patterns.

***uniform-output-noise*** global variable

The range of random distortion added to each training and test output at each epoch during training. The value added is uniformly distributed between -*uniform-output-noise* and +*uniform-output-noise*. Default value is nil. If you want to use *uniform-output-noise* setq it to some value before calling set-training-patterns.

***normal-input-noise*** global variable

The standard deviation of random distortion added to each training and test input at each epoch during training. The noise value added is normally distributed with a mean of 0 and the specified standard deviation. Default value is nil. If you want to use *normal-input-noise* setq it to some value before calling set-training-patterns.

```
(setq *normal-input-noise* .01)
```

***some-normal-input-noise*** global variable

Works like *normal-input-noise* except that only some specified inputs are distorted. List of noise standard deviations followed by list of position numbers in sub-vector to be noised. Default value is nil. If you want to use *some-normal-input-noise* setq it to some value before calling set-training-patterns.

***normal-output-noise*** global variable

The standard deviation of random distortion added to each training and test output at each epoch during training. The value added is normally distributed with a mean of 0 and the specified standard deviation. Default value is nil. If you want to use *normal-output-noise* setq it to some value before calling set-training-patterns.

(**set-patterns** patterns type) procedure

Sets up input and output vectors of type from patterns. If patterns is a string filename, the patterns are read from that file. Type is 'train, 'test, or 'change-train. Use 'train to encode training patterns at start of run. Use 'test to establish a single set of testing patterns that are different from training patterns. Use 'change-train to encode a change to a new training pattern using the same network. Patterns must be encoded as a nested association list in which the input patterns are keys and the output patterns are associates. A pattern is a list of the input or output values.

```
(((1st-input-pattern) (1st-output-pattern))
 ((2nd-input-pattern) (2nd-output-pattern))
 .
 .
```

((nth-input-pattern) (nth-output-pattern)))

***select-patterns*** global variable

If t, fresh training and/or test patterns are selected at each epoch during output phases. Default value is nil. If used, a procedure called select-patterns must be defined as a call to set-patterns, with 'test or 'change-train as type, as required. The select-patterns procedure has no parameters. Not required for random input or for input or output noise.

***select-multi-patterns*** global variable

If t, fresh multiple test patterns are selected at each epoch during output phases. Default value is nil. If used, a procedure called select-multi-patterns must be defined, with no parameters, to update *multi-test-patterns*.

(**select-random** n lst) procedure

Randomly select n items from lst without replacement.

***output-type*** global variable

Holds the type of activation function used on the output units. The default value is :sigmoid. Sigmoid is symmetric in the range -0.5 to +0.5. You can use setq to change *output-type* to :linear.

***linear-outputs*** global variable

List of exception output unit indices with linear activation functions. Use only when *output-type* is :sigmoid. Default value is nil.

***unit-type*** global variable

Holds the type of activation function used by the hidden units. The default value is :sigmoid. You can use setq to change it to :asigmoid or :gaussian. Sigmoid is symmetric in the range -0.5 to +0.5, whereas asigmoid is asymmetric in the range 0.0 to 1.0. Gaussian is symmetric in the range 0.0 to 1.0.

***score-threshold*** global variable

Holds a positive rational number used to compute output error. An output is considered correct if the difference between that output and the desired value is smaller than this value. When the number of error-bits that are wrong by more than *score-threshold* decreases to 0, the program declares victory. The default value for *score-threshold* is 0.40, which is appropriate for sigmoid output units representing 2 values. You can use setq to establish a different setting. If you are using sigmoid output units to represent 3 values, 0.25 would be a more appropriate value for *score-threshold*.

***proportion-threshold*** global variable

A proportion that provides a score threshold for learning continuous functions. Multiplies targets to provide a valid score-threshold for each target. Declare victory when all outputs are within this proportion of their targets. Default value is nil.

***tolerance*** global variable

If proportion of *error-bits* < *tolerance*, declare victory by setting *error-bits* to 0. Default value is 0, which requires *error-bits* to legitimately reach 0 for victory.

***test*** global variable

If t, then a test epoch is run every so often during training. Default value is nil. Allows continual recording of output activations, errors, contributions, Hinton values, and net information.

***test-interval*** global variable

Run a test epoch every *test-interval* output training cycles. Default value is 0.

***test-last-output-phase-epoch*** global variable

If t, run a test epoch after the last output phase epoch just prior to a shift to the input phase. Default value is nil. Useful for collecting output activations, contributions, or Hinton values only after the net has adjusted to each new hidden unit.

***record-hintons*** global variable

If t, record Hinton values every test-epoch. Default value is nil. Results stored in *hintons*.

***save-nets-for-test-setups*** global variable

If t, record net information every test-epoch for later use by test-setup. Default value is nil. Results stored in *nets-info*. Useful for recording test contributions from selected epochs.

***multi-test-patterns*** global variable

Contains a list of test patterns or file names, each holding a set of test patterns. Default value is nil. If t, tests-epoch is called to test them. If *test* is t, then these test patterns will be tested every *test-interval*. This is needed only for multiple test patterns.

***record-train-errors*** global variable

If t and *test* is t, record errors from training patterns every test-epoch. Default value is nil. Results stored in *train-errors*.

***record-test-errors*** global variable

If t and *test* is t, record errors from test patterns every test-epoch. Default value is nil. Results stored in *test-errors*.

***record-multi-errors*** global variable

If t and *test* is t, record errors from multiple test patterns every test-epoch. Default value is nil. Results are stored in *multi-errors*.

***mark-hiddens-errors*** global variable

If t, hidden unit installations are marked with an *H* in error files. Default value is nil.

***record-proportion-train-correct*** global variable

If t and *test* is t, record proportion of correct training patterns every test-epoch. Default value is nil. Results stored in *proportion-train-correct*.

***record-proportion-test-correct*** global variable

If t and *test* is t, record proportion of correct test patterns every test-epoch. Default value is nil. Results stored in *proportion-test-correct*.

***record-proportion-multi-correct*** global variable

If t and *test* is t, record proportions of correct multiple test patterns every test-epoch. Default value is nil. Results stored in *proportion-multi-correct*.

***record-train-contributions*** global variable

If t, contributions (output-weight * activation of sending unit) are recorded for the training patterns every test-epoch. Default value is nil. Results stored in *train-contributions*.

***record-test-contributions*** global variable

If t, contributions (output-weight * activation of sending unit) are recorded for the test patterns every test-epoch. Default value is nil. Results stored in *test-contributions*.

***scale-contributions-by-target*** global variable

If nil, contributions are computed as $c_{ijk} = w_{jk}a_{ij}$. If t, then $c_i = w_{jk}a_{ij}2t_{ki}$. Where $w_{jk}$ is the weight connecting sending unit j with output unit k, $a_{ij}$ is the activation of sending unit j given input pattern i, and $t_{ki}$ is the target for output unit k given input pattern i. Default value is nil.

***record-train-activations*** global variable

If t and *test* is also t, then unit activations are recorded from running training patterns every test-epoch. Default value is nil. Results are stored in *train-activations* in reverse order by epoch. Order is bias, inputs, hiddens, outputs.

***record-test-activations*** global variable

If t and *test* is also t, then unit activations are recorded from running test patterns every test-epoch. Default value is nil. Results are stored in *test-activations* in reverse order by epoch. Order is bias, inputs, hiddens, outputs.

***record-multi-test-activations*** global variable

If t and *test* is also t, then unit activations are recorded from running multi-test-patterns every test-epoch. Default value is nil. Results are stored in *multi-test-activations* in reverse order by epoch. Order is inputs, hiddens, outputs.

***display-error*** global variable

If t, the results of error testing are displayed in the Listener window. Default value is nil.

## Running

(**train** outlimit inlimit rounds &optional last-epoch restart) procedure

Train the network until stagnation or victory is reached. Outlimit and inlimit are upper bounds on the number of cycles in each output and input phase, respectively. Rounds is an upper limit on the number of unit-installation cycles. Recommended values for outlimit, inlimit, and rounds, respectively, are 100, 100, 25. One optional parameter is last-epoch, with a default value of 999. Training stops after last-epoch. Another optional parameter is restart, with a default value of nil. If restart is t, restart from the current point; the net is not re-initialized.

(**test-epoch** &optional *score-threshold*) procedure

Run the training and test patterns and, if *display-error* is t, report results. The optional argument of *score-threshold* has a default value of 0.40. If relevant options are in effect, records Hinton values, errors, proportions correct, unit activations, and contributions.

(**tests-epoch** &optional *score-threshold*) procedure

Run the multi-test patterns and, if *display-error* is t, report results. The optional argument of *score-threshold* has a default value of 0.40. If relevant options are in effect, records errors, proportions correct, and unit activations.

## Processing Results

***weights*** global variable

A vector of vectors. For each hidden unit, a vector of weights coming into each non-output unit indexed by the outside vector from each unit (bias and input units first, then hidden units) indexed by the inside vector.

***output-weights*** global variable

A vector of vectors. For each output, a vector of output weights coming from the unit indexed by the inside vector.

(**hintonize** weights output-weights &optional square-values) procedure

Compute the weight values needed for Hinton diagrams and store them in *hintons*. Convert the vectors of weights and output-weights to a-lists of values from 1 to square-values. The default setting for square-values is 7. If *record-hintons* is t, hintonize is called from test-epoch, with the default value for square-values.

### *train-errors* global variable

List of true error for training patterns, in reverse order by epoch. Default value is nil.

### *test-errors* global variable

List of true error for test patterns, in reverse order by epoch. Default value is nil.

### *multi-errors* global variable

Epoch x multiple test-patterns list of true errors, in reverse order by epoch. Default value is nil.

### *proportion-train-correct* global variable

List of proportions of correct training patterns, in reverse order by epoch. Default value is nil.

### *proportion-test-correct* global variable

List of proportions of correct test patterns, in reverse order by epoch. Default value is nil.

### *proportion-multi-correct* global variable

List of proportions of correct multiple test patterns, in reverse order by epoch. Default value is nil.

### *train-activations* global variable

List of lists of unit activations for training patterns, in reverse order by epoch. For each training pattern, activations are listed in order of inputs, hidden units, and outputs. Default value is nil.

### *test-activations* global variable

List of lists of unit activations for test patterns, in reverse order by epoch. For each test pattern, activations are listed in order of inputs, hidden units, and outputs. Default value is nil.

***multi-test-activations*** global variable

List of lists of unit activations for multi-test patterns, in reverse order by epoch. For each test pattern, activations are listed in order of inputs, hidden units, and outputs. Default value is nil.

***train-contributions*** global variable

Lists of contributions for the training patterns, in reverse order by epoch (epoch * input pattern * output-weight). Default value is nil.

***test-contributions*** global variable

Lists of contributions for the test patterns, in reverse order by epoch (epoch * input pattern * output-weight). Default value is nil.

(**save-contributions** lst path type net) procedure

Save contributions list in separate files for each epoch. List is *train-contributions* or *test-contributions*. Path is a string file path. Type is 'train or 'test. Net is index in case multiple nets are being run. Net index should be updated in the procedure that calls save-contributions. Uses lists->file with space as separator.

(**lists->file** lst file &optional (separator " ")) procedure

Reverse lst and save it in a file. Items are printed 1 per line if flat lst. File must be a double quoted string. Separator is used to separate items on a line for embedded lst. Default value for separator is space. Useful for saving any flat list or embedded lists.

***hintons*** global variable

Holds a list of Hinton values, in reverse order by epoch.

***structure*** global variable

List of hidden units per layer in reverse order of installation, i.e., last layer is first. Useful for SDCC. Be sure to reverse *structure* before consing it onto a list of structures.

***nets-info*** global variable

Lists of net information, in reverse order by epoch, for use of select-contributions. Net information includes *nunits*, reversed *structure* (i.e., first layers listed first), *weights*, *output-weights*, *connections*, and *nconnections*. Vectors are stored in list form to preserve information from earlier epochs.

(**select-contributions** positions net) procedure

Compute test-pattern contributions at selected epochs. Assumes that a file labelled "netxresults", where x is an integer, has a list of test-patterns as its first item, a reversed list of *outputs-epochs* as its second item, and a reversed list of *nets-info* lists as its third item. Positions is a list of positions in the reversed *nets-info* list using addressing from 0. Corresponding ordered contributions are stored in numbered files with addressing from 1.

(**save-for-test-setup** filename) procedure

Save net information in file for later use.

(**use-test-setup** filename) procedure

Retrieve net information from file and set up network.

(**make-Weight-Diagram** hinton-values) procedure

Draw a Hinton diagram of the weights in the network. The argument is a list of Hinton values generated by hintonize. Hinton values are created when *record-hintons* is t. The hinton-values list needs to be quoted when using make-Weight-Diagram. Press the right mouse button to see options when weight diagram has been drawn. Weight diagrams can be processed with a bitmap program such as Paint. To use make-Weight-Diagram, the file Weight-Diagram-Win.lsp must first be loaded. Does not work with Allegro 5.

## Contribution Analysis

There are three ways to record contributions for network analysis in sdcc.

1. At the end of every network run.

This is fine for analysis of final network knowledge representations, but does not provide a developmental picture. After training, set *record-train-contributions* and/or *record-test-contributions* to t, call test-epoch, and call save-contributions to store the contributions in a file. Note that if *test* has been set to t with some *test-interval*, say for some other purpose like recording errors, then *record-train-contributions* and *record-test-contributions* should be set to nil before training so as to avoid recording contributions every test-interval.

2. At every test-interval.

With a small test-interval, this is thorough but expensive. You may not really want that many sets of contributions. Before training, set the following parameters: *record-train-contributions* and/or *record-test-contributions* to t, *test* to t, and *test-interval* to some integer. After training, call save-contributions to store the contributions in separate files for each output-phase.

3. At the end of every output phase.

This may be an efficient way to get a developmental trace of network knowledge representations, but may not be sufficiently precise in terms of providing a snap shot of each stage because stages may be missed. Before training, set the following parameters: *record-train-contributions* and/or *record-test-contributions* to t, and *test-last-output-phase-epoch* to t. After training, call save-contributions to store the contributions in separate files for each output-phase.

However the contributions have been generated, they can be subjected to PCA using a statistics program. Ordinarily, in a PCA, we analyze the covariance matrix, apply a varimax rotation, and use a scree test to decide how many components to retain. Note that rescaled component loadings are standardized loadings.

Thomas R. Shultz
14 June 2006