RNNLIB is a recurrent neural network library for sequence learning problems. Applicable to most types of spatiotemporal data, it has proven particularly effective for speech and handwriting recognition.

**Contents**

* Contents
* Introduction
* [Installation](https://sourceforge.net/p/rnnl/wiki/Home/#installation)
* [Usage](https://sourceforge.net/p/rnnl/wiki/Home/#usage)
* [Autosave](https://sourceforge.net/p/rnnl/wiki/Home/#autosave)
* [Data File Format](https://sourceforge.net/p/rnnl/wiki/Home/#data-file-format)
* [Examples](https://sourceforge.net/p/rnnl/wiki/Home/#examples)
* [Utilities](https://sourceforge.net/p/rnnl/wiki/Home/#utilities)
* [Experimental Results](https://sourceforge.net/p/rnnl/wiki/Home/#experimental-results)
* [Citations](https://sourceforge.net/p/rnnl/wiki/Home/#citations)
* [References](https://sourceforge.net/p/rnnl/wiki/Home/#references)

**Introduction**

RNNLIB is a recurrent neural network library for sequence labelling  
problems, such as speech and handwriting recognition. It implements the  
Long Short-Term Memory (LSTM) architecture1, as well as more  
traditional neural network structures, such as Multilayer Perceptrons  
and standard recurrent networks with nonlinear hidden units. Its most  
important features are:

* Bidirectional Long Short-Term Memory2, which provides access to  
  long range contextual information in all input directions
* Connectionist Temporal Classification3, which allows the system to  
  transcribe unsegmented sequence data
* Multidimensional Recurrent Neural Networks4, which extends the  
  system to data with more than one spatiotemporal dimension (images,  
  videos, fMRI scans etc.)

All of which are explained in more detail in my Ph.D. thesis5. The library also implements the multilayer, subsampling structure developed for offline arabic handwriting recognition6. This structure allows the network to efficiently label high resolution data such as raw images and  
speech waveforms. Taken together, the above components make RNNLIB a generic system for  
labelling and classifying data with one or more spatiotemporal dimensions. Perhaps its greatest strength is its flexibility: as well as speech and handwriting7 recognition, it has so far been applied (with varying degrees of success) to image classification, object recognition, facial expression recognition, EEG and fMRI classification, motion capture labelling, robot localisation, wind turbine energy prediction,signature verification, image compression and touch sensor classification. RNNLIB is also able to accept a wide variety of different input representations for the same task, e.g. raw sensor data or hand-crafted features (as shown for online handwriting8). See my [homepage](http://www6.in.tum.de/Main/Graves) for more publications.

RNNLIB also implements adaptive weight noise regularisation14, which makes it possible to train an arbitrary neural network with stochastic variational inference (or equivalently, to minimise the two part description length of the training data given the network weights plus the weights themselves). This form of regularisation makes overfitting virtually impossible; however it can lead to very long training times.

**Installation**

RNNLIB is written in C++ and should compile on any platform. However it is currently only tested for Linux and OSX.

Building it requires the following:

* A modern C++ compiler (e.g. gcc 3.0 or higher)
* [GNU Libtool](http://www.gnu.org/software/libtool/)
* [GNU automake version 1.9](http://www.gnu.org/software/automake/)  
  (NOTE: will not work with version 1.10)
* [NetCDF scientific data  
  library](http://www.unidata.ucar.edu/software/netcdf/)
* [Boost C++ Libraries](http://www.boost.org/) version 1.36 or higher  
  (headers only, no compilation needed.)

In addition, the following python packages are needed for the auxiliary scripts in the ‘utils’ directory:

* [SciPy](http://www.scipy.org/)
* [matplotlib](http://matplotlib.sourceforge.net/)
* [PIL](http://www.pythonware.com/products/pil/)

And these packages are needed to create and manipulate netcdf data files with python, and to run the experiments in the ‘examples’ directory:

* [ScientificPython](http://sourcesup.cru.fr/projects/scientific-py/)  
  (NOT Scipy)
* [netCDF Operator](http://nco.sourceforge.net/)

To build RNNLIB, first download the source, then enter the root directory and type

./configure

make

This should create the binary file ‘rnnlib’ in the ‘src’ directory. Note  
that on most linux systems the default installation directory for the  
Boost headers is ‘/usr/local/include/boost-VERSION\_NUMBER’ which is not  
on the standard include path. In this case type

CXXFLAGS=-I/usr/local/include/boost-VERSION\_NUMBER/ ./configure

make

If you wish to install the binary type:

make install

By default this will use ‘/usr’ as the installation root (for which you will usually need administrator privileges). You can change the install path with the --prefix option of the configure script (use ./configure  
--help for other options)

It is recommended that you add the directory containing the ‘rnnlib’ binary to your path, as otherwise the tools in the ‘utilities’ directory will not work.

Project files are provided for the following integrated development environments in the ‘ide’ directory:

* kdevelop (KDE, linux)
* xcode (OSX)

**Usage**

RNNLIB can be run from the command line as follows:

Usage: rnnlib [config\_options] config\_file

config\_options syntax: --<variable\_name>=<variable\_value>

whitespace not allowed in variable names or values

all config\_file variables overwritten by config\_options

setting <variable\_value> = "" removes the variable from the config

repeated variables overwritten by last specified

All the parameters determining the network structure, experimental setup etc. can be specified either in the config file or on the command line.

The main parameters are as follows:

| **Parameter** | **Type** | **Allowed Values** | **Default** | **Comment** |
| --- | --- | --- | --- | --- |
| autosave | boolean | true,false | false | see below |
| batchLearn | boolean | true,false | true if RPROP is used, false otherwise | false => gradient descent updates at the end of each sequence, true => at the end of epochs only |
| dataFraction | real | 0-1 | 1 | determines fraction of the data to load |
| hiddenBlock | list of integer lists | all >=1 |  | Hidden layer block dimensions |
| hiddenSize | integer list | all >=1 |  | Sizes of the hidden layers |
| hiddenType | string | tanh, linear, logistic, lstm, linear\_lstm, softsign | lstm | Type of units in the hidden layers |
| inputBlock | integer list | all >= 1 |  | Input layer block dimensions |
| maxTestsNoBest | integer | >=0 | 20 | Number of error tests without improvement on the validation set before early stopping |
| optimiser | steepest, rprop | steepest |  |  |
| learnRate | real | 0-1 | 1e-4 | Learning rate (steepest descent optimiser only) |
| momentum | real | 0-1 | 0.9 | Momentum (steepest descent optimiser only) |
| subsampleSize | integer list | all >= 1 |  | Sizes of hidden subsample layers |
| task | string | classification, sequence\_classification, transcription |  | Network task. sequence\_\* => one target for whole sequence (not for each point in the sequence). transcription => unsegmented sequence labelling with CTC. |
| trainFile | string list |  |  | Netcdf files used for training. Note that all datasets can consist of multiple files. During each training epoch, the files will be cycled through in random order, with the sequences cycled randomly within each file |
| valFile | string list |  |  | Netcdf files used for validation / early stopping |
| testFile | string list |  |  | Netcdf files used for testing |
| verbose | boolean | true,false | false | Verbose console output |
| mdl | boolean | true,false | false | Use adaptive weight noise (M)inimum (D)escription (L)ength regularisation |
| mdlWeight | real | 0-1 | 1 | weight for MDL regularisation (0 => no regularisation; 1 => *true* MDL) |
| mdlInitStdDev | real | >0 | 0.075 | initial std. dev. for MDL adaptive weight noise |
| mdlSamples | int | >=1 | 1 | number of Monte Carlo samples to pick for each sequence to get stochastic derivs for MDL adaptive weight noise (more samples => less noisy derivatives, more computationl cost) |
| mdlSymmetricSampling | boolean | true,false | false | if true, use symmetric (AKA antithetical) sampling to reduce variance in the derivatives |

Parameter names and values are separated by whitespace, and must  
themselves contain no whitespace. Lists are comma separated, e.g.:

trainFile a.nc,b.nc,c.nc

and lists of lists are semicolon separated, e.g.:

hiddenBlock 3,3;4,4

See the ‘examples’ directory for examples of config files. To override parameters at the command line, the syntax is:

rnnlib --OPTION\_NAME=VALUE CONFIG\_FILE

so e.g.

rnnlib --learnRate=1e-5 CONFIG\_FILE

will override the learnRate set in the config file.

**Autosave**

If the 'autosave' option is true the system will store all dynamic information (e.g. network weights) as it runs. Without this there will be no way to to resume an interrupted experiment (e.g. if a computer crashes) and the final trained system will not be saved. If saving is activated, timestamped config files with dynamic information appended will be saved after each training epoch, and whenever one of the error measures for the given task is improved on. In addition a timestamped log file will be saved, containing all the console output. For example, for a classification task, the command

rnnlib --autosave=true classification.config

might create the following files

* classification@2009.07.17-13.08.40.712422.best\_classificationError.save
* classification@2009.07.17-13.08.40.712422.best\_crossEntropyError.save
* classification@2009.07.17-13.08.40.712422.last.save
* classification@2009.07.17-13.08.40.712422.log

**Data File Format**

All RNNLIB data files (for training, testing and validation) are in [netCDF](http://www.unidata.ucar.edu/software/netcdf/) format, a binary  
file format designed for large scientific datasets.

A netCDF file has the following basic structure:

* Dimensions:
* Variables:
* Data:

Following the statement ‘Variables’ the variables that will listed in the ‘Data’ section are declared. For example

float foo[ 3 ]

would declare an array of floats with size 3. For saving variable sized array the size can be declared after ‘Dimensions’. So the example would  
look like:

Dimensions:

fooSize= 3

Variables:

float foo[ fooSize ];

Following ‘Data’ the actual values are stored:

Data:

foo = 1,2,3;

The data format for RNNLIB is specified below. The codes at the start determine which tasks the dimension/variable is required for:

* R = regression (sum-of-squares error with linear outputs)
* T = transcription (sequence labelling with connectionist temporal  
  classification outputs)
* C = classification (cross-entropy error with softmax outputs)
* SC = sequence\_classification (as above, but only one target per  
  sequence)
* O = optional, not required for any task

Dimensions:

* numSeqs = total number of data sequences
* numTimesteps = total number of timesteps (sum of lengths of all sequences)
* inputPattSize = size of input vectors (e.g. 3 if input points are RGB pixels)
* ( O ) maxSeqTagLength = length of longest sequence tag string   
  (including null terminator)
* ( R ) targetPattSize = size of target vectors
* ( T, SC ) maxTargStringLength = length of longest target string  
  (including null terminator)
* ( T, C, SC ) numLabels = number of distinct class labels
* ( T, C, SC ) maxLabelLength = length of longest label string  
  (including null terminator)

Variables:

* float inputs[numTimesteps,inputPattSize] = array of input vectors
* int seqDims[numSeqs,numDims] = array of sequence dimensions
* ( R ) float targetPatterns[numTimesteps,targetPattSize] = array of  
  regression target vectors
* ( C ) int targetClasses[numTimesteps] = array of target classes
* ( T, SC ) char targetStrings[numSeqs,maxTargStringLength] = array of  
  target strings for transcription
* ( T, C, SC ) char labels[numLabels, maxLabelLength] = class label  
  names (can just be “1”,“2”…)
* ( O ) char seqTags[numSeqs,maxSeqTagLength] = array of tags for  
  sequences (e.g. filename they were created from)

[netCDF Operator](http://nco.sourceforge.net/) provides several tools for creating, manipulating and displaying netCDF files, and is recommended for anyone wanting to make their own datasets. In particular the toold ncgen and ncdump convert ASCII text files to and from netcdf  
format.

**Examples**

The ‘examples’ directory provides example experiments that can be run with RNNLIB. To run the experiments, the ‘utilities’ directory must be added to your pythonpath, and the following python packages must be installed:

* [SciPy](http://www.scipy.org/)
* [ScientificPython](http://sourcesup.cru.fr/projects/scientific-py/)
* [PIL](http://www.pythonware.com/products/pil/)

In each subdirectory type

./build\_netcdf

to build the netcdf datasets, then

rnnlib SAMPLE\_NAME.config

to run the experiments. Note that some directories may contain more than 1 config file, since different tasks may be defined for the same data. The results of these experiments will not correspond to published results, because only a fraction of the complete dataset is used in each  
case (to keep the size of the distribution down). In addition, early stopping is not used, because no validation files are created. However the same scripts can be used to build realistic experiments, given more data.

If you want to adapt the python scripts to create netcdf files for your own experiments, [here](http://gfesuite.noaa.gov/developer/netCDFPythonInterface.html) is a useful tutorial on using netcdf with python.

**Utilities**

The ‘utilities’ directory provides a range of auxiliary tools for RNNLIB. In order for these to work, the directory containing the ‘rnnlib’ binary must be added to your path. The ‘utilities’ directory must be added to your pythonpath for the experiments in the ‘examples’  
directory to work. The most important utilities are:

* dump\_sequence\_variables.sh: writes to file all the internal  
  variables (activations, delta terms etc.) of the network while  
  processing a single sequence
* plot\_variables.py: plots a single variable file saved with  
  ‘dump\_sequence\_variables’
* plot\_errors.sh: plots the error curves written to a log file during  
  training
* normalise\_inputs.sh: adjusts the inputs of one or more netcdf files  
  to have mean 0, standard deviation 1 (relative to the first file,  
  which should be used for training)
* gradient\_check.sh: numerically checks the network’s gradient  
  calculation

All files should provide a list of arguments if called with no arguments.The python scripts will give a list of optional arguments, defaults etc. if called with the single argument ‘-h’. The following python libraries are required for some of the scripts:

* [SciPy](http://www.scipy.org/) (for all scripts)
* [matplotlib](http://matplotlib.sourceforge.net/) (for all  
  plotting/visualisation scripts)
* [PIL](http://www.pythonware.com/products/pil/) (for  
  plot\_variables.py)
* [ScientificPython](http://sourcesup.cru.fr/projects/scientific-py/)  
  (for normalise\_inputs.sh)

**Experimental Results**

RNNLIB’s best results so far have been in speech and handwriting recognition. It has matched the best recorded performance in phoneme recognition on the TIMIT database9, and recently won three handwriting recognition competitions at the ICDAR 2009 conference, for offline  
French10, offline Arabic11 and offline Farsi character classification12. Unlike the competing systems, RNNLIB worked entirely on raw inputs, and therefore did not require any reprocessing or alphabet-specific feature extraction. It also has among the best published results on the IAM Online and IAM offline English handwriting databases13.

**Citations**

If you use RNNLIB for your research, please cite it with the following reference:

@misc  
{rnnlib,  
Author = {Alex Graves},  
Title = {RNNLIB: A recurrent neural network library for sequence learning problems},  
howpublished = {\url{http://sourceforge.net/projects/rnnl/}}}