Grammatical Evolution Neural Networks (GENN)
User's Guide

The code and scripts described herein have been developed, tested, and extended by many people, the most recent of which is Nicholas Hardison (nhardis@unity.ncsu.edu).

Initial Configuration

GENN was developed and tested on Linux (32-bit RHEL and Ubuntu). GENN is a parallel program which uses the Message Passing Interface (MPI) standard for interprocess communication. Both LAM/MPI and MPICH implementations have been successfully used. If your environment already has a functioning MPI implementation, you should use that one (See the Compiling GENN section, below). If your environment does not already contain a functioning MPI implementation, you will need to configure one. We will describe this process for LAM/MPI. Our example environment uses GCC 3.4.6 (We have also used GCC 3.2.3). You can check your installed version by executing 'g++ --version'. We have noticed spurious results using GCC 4.1.3, and recommend using a GCC 3.x compiler. Various versions of GCC itself can be obtained from www.gnu.org, and your distribution likely has older versions available for download.

Configuring LAM/MPI

First, download the source code from www.lam-mpi.org. We have used version 7.0.6, but newer versions should work as well.

```
$ wget http://www.lam-mpi.org/download/files/lam-7.0.6.tar.gz
```

Unpack the source code and change to the newly created subdirectory.

```
$ tar zxf lam-7.0.6.tar.gz
$ cd lam-7.0.6
```

Now, configure the software. If you have administrative privileges on the machine you're working on, you can install the software in the default, universal location. For our purposes, we will add an additional option to change the install location from '/usr/local' to '/tmp/testing/local'. If you wish to use a different compiler than the default that is in your path, you should set the CC/CXX environment variables appropriately.

```
$ ./configure -prefix=/tmp/testing/local
```

Now, start the compile by using the make command. This takes some time to complete.

```
$ make
```

After this has completed, install the software:

```
$ make install
```

There is now an installed LAM/MPI environment on your machine. This environment must be started before GENN can be executed. Use the 'lamboot' command to do this. If you used a custom location, it

is found under $PREFIX/bin. If you wish to shutdown LAM, you can use the 'lamhalt' command in the same directory. Also in this directory is the 'mpiCC' command this acts as a compiler wrapper which includes options to properly use the LAM/MPI headers and libraries. If you encountered difficulties while configuring LAM/MPI, please consult the package documentation.

Compiling GENN

Unpack the provided source file and change into that directory:

```
$ tar zxf genn.tgz
$ cd genn
```

If you used a non-standard install location for LAM/MPI, you need to use the editor of your choice to configure the GALib [ref] to point to the desired compiler by changing the 'makevars' file in the galib/ directory.

```
$ cd galib245
$ vi makevars
```

If needed, change the line setting the CXX variable to point to the mpiCC you just compiled

```
CXX = /tmp/testing/local/bin/mpiCC
```

Then, compile the library:

```
$ make
```

If needed, change the Makefile for GENN to also use the mpiCC you're using. This makefile is in the genn/src-mpi directory. Change the 'CC' variable to the mpiCC location:

```
CC = /tmp/testing/local/bin/mpiCC
```

Compile GENN by running 'make' in the 'genn/src-mpi' directory.

```
$ make
```

Congratulations! The GENN executable (gennMPI) is now in the gen/exe-mpi directory.

GENN Tunables

Depending on the size of the datasets you plan to analyze, you may need to change the `MAXLINES` constant in the `genn/src-mpi/defines.h` file to a larger number. This constant defines how long a given line of input data can be. If it is too small, GENN will hang trying to read in the data files. If it is too large for your environment, the program will have segmentation faults. You can check the size of the longest line in your data using the provided `linesize_check.pl` perl script.

Another constant you may wish to change is `MAX_MISSING`. When GENN is running, it prints out a message whenever a neural network attempts to evaluate a missing value. In order to limit the size of output files, a cap has been placed on the number of times this can be reported. If you would like to

increase the reporting size, you can change it here.

GENN Support Scripting

You need to change the GENN cross-validation script slightly in order to point to wherever you have the modules unpacked. In the script `bin/cross_validate-mpi.pl`, change the line

```
use lib "/home/nicholas/motsinger/genn/lib/scripts";
```

to point to the lib/ directory containing `GENNCrossValidation.pm` and `CrossValidation.pm`. In addition, in the `lib/CrossValidation.pm` file, you must also change the lines which resemble

```
require "/home/nicholas/motsinger/genn/lib/scripts/GENNCrossValidation.pm";
```

to point to this directory as well.

File Formats

Data File

Before running a GENN analysis, there are several files that need to be created. The first is the data file, containing genotype information and case/control status. In this file, each sample is described on a different row. The last column in each row is the case/control status (represented as a 0 or 1). The other columns contain genotype data. Columns are separated by whitespace. Missing data are acceptable the 'missing' value is '3'

In order to remove possible linear dependance between different genotypes, this data is transformed using the dummy encoding method proposed by Ott [REF], such that each locus is represented by two columns. A companion script, `ott-dummy.pl`, is included to convert data in more traditional genotype format (one column per locus) into this format. This script takes an input file named `dataname.1.txt` and creates a file named `dataname.1.ott-dummy.txt`. The original file must have the case/control status in the first column (as 0/1), and the remaining columns should encode genotype data as '0', '1', or '2'. These values will be transformed into '-1 -1', '0 2', and '1 -1'. Any values besides '0', '1', or '2' will be transformed into '3 3'.

GENN Configuration File

Each execution of GENN requires a configuration file. We actually create a template (Usually called `modelconfig.genn`) in the same style as the format file this will be used by the cross-validation process (described next) to generate separate configuration files for each run. The file consists of name-value pairs. An example is below:

```
pop_size = 200
max_generations = 400
pvm_exchange_generation = 25
random_seed = 7
```

```
crossover_rate = 0.9
mutation_rate = 0.01
codon_size = 8
wrapper_count = 2
min_chrom_size = 50
max_chrom_size = 1000
update_interval = 5
init_depth = 10
selection = tournament
crossover = standard
train_file = /home/nicholas/motsinger/genndist/data/A300_U300_ZZ.
99/work/data24_0.txt.trn
test_file = /home/nicholas/motsinger/genndist/data/A300_U300_ZZ.
99/work/data24_0.txt.tst
grammar_file = /home/nicholas/motsinger/genndist/data/A300_U300_ZZ.
99/comma_small_mod.gram
output.basename = sample
fitness = balanced
```

When this configuration file is used as a template by the cross-validation script, the `train_file`, `test_file`, and `grammar_file` variables get replaced by appropriate values, so they do not have to be correct here. In general, the only parameter which needs to change from analysis to analysis is `max_generations`. Explanations of the parameters follow:

`pop_size` This is the number of NNs in each population.

`max_generations` This controls the number of generations to run the machine learning process. As a rule of thumb, this should be equal to twice the number of columns in the dataset (four times the number of genotyped loci).

`pvm_exchange_generation` This is how many generations of independent evolution the populations undergo between migrations. During migration, the neural network from each population that best fits the data is copied to the other populations.

`random_seed` This seeds the random number generator used in the evolutionary process.

`crossover_rate` -

`mutation_rate` -

`codon_size` -

`wrapper_count` -

`min_chrom_size` -

`max_chrom_size` -

`update_interval` -

`init_depth` -

`selection` -

`crossover` -

`train_file` -

`test_file` -

`grammar_file` -

Cross-validation Configuration File

A second configuration file (Usually called `model1.config`) is used to control the cross-validation runs. An example is below:

```
# for GENN method in the Crossvalidation system

exeDirectory = /home/nicholas/motsinger/genn/genn/exe-mpi
workingDirectory = /home/nicholas/motsinger/genndist/data/A300_U300_ZZ.99/work
baseDatasetDirectory = /home/nicholas/motsinger/genndist/data/A300_U300_ZZ.99
baseDatasetName = A300_U300_ZZ.99.ott-dummy.txt
methodType = GENN
methodConfigTemplate= /home/nicholas/motsinger/genndist/data/A300_U300_ZZ.
99/modelconfig.genn
methodGrammarFile = /home/nicholas/motsinger/genndist/data/A300_U300_ZZ.
99/comma_small_mod.gram
randomSeed = 1234
maxWeightInput = 5
zeroDigits = 2
constantRightDigits = 2
constantLeftDigits = 0
crossValidation = 10

gennNumProcesses = 4
gennHostsFile = /home/nicholas/motsinger/genn/data/pvmhost
gennTimeout = 30
gennQueue = brc
```

This file is used as a parameter to the cross-validation script. An explanation of the parameters follows:

exeDirectory the directory containing the gennMPI executable.

Grammar File

This file contains the grammar which will be used by GENN. An example is below.

```
N = {p,pn,pinput,wt,winput,cop,op,v,num, dig}
T = {PA,PS,PD,PM,W,*,+,/,-,V1-20,Concat,.}
S = p

<p>      ::= <pn>(<pinput>)

<pn>     ::= PA
           | PS
           | PM
           | PD

<pinput> ::=  W(<winput>), W(<winput>), 2
           |  W(<winput>), W(<winput>), W(<winput>), 3
           |  W(<winput>), W(<winput>), W(<winput>), W(<winput>), 4
           |  W(<winput>), W(<winput>), W(<winput>), W(<winput>), W(<winput>), 5

# arguments to functions need commmas between each argument

<winput> ::= <cop>, <v>
           | <cop>, <p>

<cop>    ::= (<cop> <op> <cop>)
           | Concat(<num>)

<op>     ::= +
           | -
           | *
           | /


<num>    ::=  . 0 0 <dig> 4
           |  . 0 <dig> 3

<dig>    ::= 0
           | 1
           | 2
           | 3
           | 4
           | 5
           | 6
           | 7
           | 8
           | 9

<v>      ::= V1-20
```

This file is also used as a template by the cross-validation script. Since the number of variables (<v>) depends on the individual analysis, the cross-validation script will analyze the dataset and create another grammar file to be used, with the appropriate number of variables within it.

GENN Analysis [local MPI install, not LSF]

The first phase of an analysis is to execute GENN on the full dataset. Format the data and create configuration files as described above, then execute the cross-validation script with its configuration file as the parameter. You can capture the output to a file using the `tee` command.

```
$ /home/nicholas/motsinger/genn/bin/cross_validate-
mpi.pl /home/nicholas/motsinger/genn/data/hiv1-mpi/model1.config | tee analysis.log
```

After analysis is completed, the analysis logfile can be interpreted by the `simplecount.pl` script, which summarizes the cross-validation consistency of variables found by the different runs.

```
$ cat analysis.log | /home/nicholas/motsinger/genn/bin/simplecount.pl
Runs: 10 (Make sure this is what you expect)
Locus   Count   %
Avg. training misclassification: 0.20926
Avg. testing misclassification: 0.22334
10      10      100
5       10      100
11      2       20
84      1       10
74      1       10
21      1       10
75      1       10
71      1       10
3       1       10
64      1       10
61      1       10
2       1       10
20      1       10
65      1       10
```

If the main analysis discovers what appear to be causal variables (5 and 10, above), the next step is to ascertain significance by way of a permutation test which forces models to only contain those variables. Create a new genotype data file containing only data on the loci found, create a model1.config script to analyze it, and run the analysis to get the testing misclassification rate. Then, determine the null distribution of this statistic by running the permtest.pl script. In general 1000 replications is adequate:

```
$ /home/nicholas/motsinger/genn/bin/permtest.pl /home/nicholas/motsinger/genn/bin/c
ross_validate-
mpi.pl /home/nicholas/motsinger/genn/data/analysis1/permtest /home/nicholas/motsing
er/genn/data/analysis1/model1.config 1000
```

After execution, a file in the working directory called `distribution.out` will be created, containing a sorted list of statistics. Compare the misclassification error you determined for the forced model above, and see where it lies in this distribution to determine the p-value. For example, if your misclassification

error was .142, and the beginning of `distribution.out` looks like:

```
0.42494
0.42545
0.4274
0.42756
0.42961
0.43007
0.43011
0.43195
0.43277
```

The p-value would be p<.001. If the misclassification error had been .42742, then the p-value would be p=.002.