HyperNEAT C++ Version 3.0 Manual

June 16, 2010

1 License

2 Usage and Support

We hope that this software will be a useful starting point for your own explorations in interactive evolution with NEAT. The software is provided as is; however, we will do our best to maintain it and accommodate suggestions. If you want to be notified of future releases of the software or have questions, comments, bug reports or suggestions, send an email to jgauci@cs.ucf.edu

Alternatively, you may post your questions on the NEAT Users Group at :
http://tech.groups.yahoo.com/group/neat/.

3 Introduction to HyperNEAT

HyperNEAT is an extension of NEAT (NeuroEvolution of Augmenting Topologies) that evolves CPPNs (Compositional Pattern Producing Networks) that encode large-scale neural network connectivity patterns. A complete explanation of HyperNEAT is available here:

```
@inproceedings{Gauci:NeuralComp10,
author = {Gauci, Jason and Stanley, Kenneth O.},
title = {Autonomous Evolution of Topographic Regularities in Artificial Neural Net-
works },
booktitle = {Neural Computation},
year = {2009},
publisher = {MIT Press},
address = {Cambridge, MA, USA},
note = {To appear},
url = "http://eplex.cs.ucf.edu/publications/2010/gauci.nc10.html",
}
@inproceedings{Stanley:ALife09,
author = {Stanley, Kenneth O. and D'Ambrosio, David B. and Gauci, Jason},
title = {A hypercube-based encoding for evolving large-scale neural networks},
booktitle = {Artificial Life},
volume = {15},
number = {2},
year = {2009},
issn = {1064-5462},
pages = {185–212},
publisher = {MIT Press},
address = {Cambridge, MA, USA},
url = "http://eplex.cs.ucf.edu/publications/2009/stanley.alife09.html",
}
@InProceedings{gauci:aaai08,
author = "Jason Gauci and Kenneth O. Stanley",
title = "A Case Study on the Critical Role of Geometric Regularity in Machine Learn-
ing",
booktitle = "Proceedings of the Twenty-Third AAAI Conference on Artificial Intelli-
gence (AAAI-2008).",
year = 2008,
publisher = "AAAI Press",
address = "Menlo Park, CA",
site = "Chicago",
url = "http://eplex.cs.ucf.edu/publications/2008/gauci.aaai08.html",
}
@InProceedings{gauci:gecco07,
author = "Jason Gauci and Kenneth O. Stanley",
title = "Generating Large-Scale Neural Networks Through Discovering Geo
metric Regularities",
booktitle = "Proceedings of the Genetic and Evolutionary
Computation Conference (GECCO 2007)",
year = 2007,
publisher = "ACM Press",
address = "New York, NY",
site = "London",
url =
"http://eplex.cs.ucf.edu/publications/2007/gauci.gecco07.html",
}
```

The version of HyperNEAT distributed in this package executed the experiments in the above papers. For more information, please visit the EPlex website at: http://eplex.cs.ucf.edu/ or see more of our publications on HyperNEAT and CPPNs at: http://eplex.cs.ucf.edu/publications.html

4 Compiling HyperNEAT

To compile HyperNEAT, the dependencies must be installed or built, then cmake must be run to generate project files, and finally the HyperNEAT project files must be compiled into the executables.

4.1 Required Programs & Libraries

CMake: This program generates project files from the HyperNEAT source code. It is possible to generate project files for most common development environments including visual studio, makefiles, eclipse, codeblocks, and xcode.

Zlib: Zlib allows TinyXMLPlus (see below) to create gzip files and save space on the disk.

TinyXMLPlus: This library is based on the original TinyXML Library from Grinning Lizard, but includes support for creating .xml.gz files. Because HyperNEAT populations tend to be quite large, this library saves disk space by compressing the xml files into an xml.gz. This library is included in the HyperNEAT distribution.

JG Template Library: This library includes a set of utilities and base classes useful for any C++ program. This library is also included in the HyperNEAT distribution.

Boost C++ libraries: The boost libraries are used for the python bindings, threading, filesystem managemnt, and several other features. These libraries are available from a package management system (apt-get on debian, opkg on OE, MacPorts on OS/X) or can be downloaded from the internet (www.boost.org).

WxWidgets: This library is only needed if compiling with the USE_GUI option set to TRUE. It provides a cross-platform API for writing GUI code that is used by the HyperNEAT GUI.

OpenMPI: This library is only needed if compiling with the BUILD_MPI option. This enables HyperNEAT runs to be performed on a cluster environment using the Message Passing Interface 2 (MPI2) interface.

Python: The HyperNEAT visualizer and several other tools are written in Python. If HyperNEAT is built with the BUILD_PYTHON flag set, these tools can use the core HyperNEAT API to function.

OpenCL: This library is only used when the BUILD_GPU flag is enabled. It allows Artifical Neural Networks (ANNs) to use the Graphics Processing Unit (GPU) instead of the CPU.

libBoard: This library creates the vector graphics representations of HyperNEAT substrates. It is included in the zip or can be found here: (http://libboard.sourceforge.net/)

4.2 Building TinyXMLPlus and JGTL

To build TinyXMLPlus, enter the /tinyxmldll/build directory and create two subdirectories, one for debug and one for release:

```
mkdir build/debug
mkdir build/release
```

Now, run cmake on each of those directories with the -i option, and put the appropriate configuration (debug for */debug, release for */release):

```
cmake -i ../../
```

If you are running Windows or OS/X, you can use the cmake GUI. Set the build directory to /tinyxmldll/build/debug and the source directory to /tinyxmldll and click “configure”, then put in the appropriate configuration (debug for */debug, release for */release), hit “configure” again, and then “generate” to generate the solution.

Finally, run "make" on the build/*/ directories and the libraries should be created in the out/ folder

To build JGTL, perform these same steps in the /JGTL/build directory. **NOTE:** JGTL is a header-only library. Building the library only gives access to the example and test programs. As a result, this step is entirely **optional**.

4.3 Building HyperNEAT

To build HyperNEAT, follow the same instructions for building tinyxmldll, but in the /NE/HyperNEAT directory. There are some additional settings that can be controlled from within cmake to influence the build process:

USE_GUI: Setting this to TRUE enables the HyperNEAT GUI interface. This is recommended for most builds, and requires the WxWidgets library. Turning this option off is useful for running on a “headless” environment such as a cluster, where no graphical framework is running.

BUILD_PYTHON: This setting determines whether the python bindings will be built or not. The HyperNEAT visualizer requires python bindings.

BUILD_MPI: This flag controls the building of the HyperNEAT MPI interface. The MPI interface allows HyperNEAT to be run on a cluster of machines.

BUILD_GPU: This flag enables the OpenCL Artifical Neural Network (ANN) implementation. This implementation uses the Graphics Processing Unit (GPU) and is faster than the CPU implementation, but consumes the GPU.

5 Running existing experiments

Once HyperNEAT has been compiled, the existing experiments can be run using the command line or (if compiled with the USE_GUI cmake variable set) with the GUI.

5.1 Comand Line

To use the command line interface, simply pass the data file and the output file as parameters to the HyperNEAT executable, and it will process the data file and put the resulting population as an XML file in the output file location.

As an example, here is a command line for running the Checkers experiment:

```
./Hypercube_NEAT -I ./data/CheckersExperiment.dat \
-O CheckersOutputPopulation.xml -R 1000
```

The -I switch controls the input parameter file (typically a .dat file). The -O switch controls the output XML file containing the population. Note that HyperNEAT outputs files in the format *.xml.gz, so HyperNEAT will append ".gz" to the end of the output automatically. Use a program like gunzip to convert the .xml.gz back into a .xml file. The -R switch contains the random seed for the population. Omitting a -R switch will make HyperNEAT use the current time in milliseconds as the random seed.

5.2 GUI

To load an experiment, go to file->load_experiment and then select the .dat file that corresponds to the experiment you would like to load.

When the experiment is loaded, go to file->run_experiment to begin the experiment. In the console, you can see the results for each generation.

If you wish to pause the experiment, go to file->pause_experiment. This command will finish the current generation and then halt the experiment. You can continue by going to file->continue_experiment.

At the moment, the restart_experiment function has not been implemented and you can't load another experiment without first closing the program (although you may "continue experiment" after you have paused).

It is possible to view an individual interactively by choosing the individual you want from the spinners and then clicking "view". This brings up a separate window where you can interact with the individual (if the experiment supports it) and you can also see the CPPN that created the individual.

Some experiments contain a post-hoc analysis that does a more complete test of an individual. This analysis can be achieved by clicking the "Analyze" button and the results are typically displayed in the console.

6 How to create a new experiment

The following sections describe how to create a new experiment with the HyperNEAT C++ Source code.

6.1 Copy existing experiment files

Make a copy of an existing experiment or start with the HCUBE_EmptyExperiment.h/cpp[1] files in the doc directory:

```
Command Line:

---Start with an existing experiment---

cp NE/HyperNEAT/include/Experiments/HCUBE_CheckersExperiment.h
   NE/HyperNEAT/include/experiments/HCUBE_MyNewExperiment.h
cp NE/HyperNEAT/src/Experiments/HCUBE_CheckersExperiment.cpp
   NE/HyperNEAT/src/experiments/HCUBE_MyNewExperiment.cpp

---OR use the EmptyExperiment---

cp NE/HyperNEAT/doc/HCUBE_EmptyExperiment.h
   NE/HyperNEAT/include/experiments/HCUBE_MyNewExperiment.h
cp NE/HyperNEAT/doc/HCUBE_EmptyExperiment.cpp
   NE/HyperNEAT/src/experiments/HCUBE_MyNewExperiment.cpp
```

Now, add the new files to the cmake project. Open NE/HyperNEAT/CMakeLists.txt and add two lines:

```
CMakeLists.txt:

...
        src/Experiments/HCUBE_CheckersCommon.cpp
        src/Experiments/HCUBE_CheckersExperiment.cpp
    src/Experiments/HCUBE_MyNewExperiment.cpp        <--- NEW LINE
...

...
        include/Experiments/HCUBE_CheckersCommon.h
        include/Experiments/HCUBE_CheckersExperiment.h
    include/Experiments/HCUBE_MyNewExperiment.h        <--- NEW LINE
...
```

After this step, rerun cmake to regenerate new project files with your files added. If cmake fails, it probably means that the new files do not exist in the correct location.

6.2 Define the New Experiment Type

All HyperNEAT experiments are defined with an ID in HCUBE_Defines.h. At the time of writing, the following defines are present:

[1]Note that the Empty Experiment purposely has compiler errors in areas of the code that require user customization and the experiment will not compile without addressing those areas.

```
enum ExperimentType
{
    EXPERIMENT_XOR=0,                                   //0
    EXPERIMENT_TIC_TAC_TOE=6,                           //6
    EXPERIMENT_TIC_TAC_TOE_GAME=9,                      //9
    EXPERIMENT_TIC_TAC_TOE_NO_GEOM_GAME=10,             //10
    EXPERIMENT_FIND_POINT_EXPERIMENT=11,                //11
    EXPERIMENT_FIND_CLUSTER_EXPERIMENT=12,              //12
    EXPERIMENT_FIND_CLUSTER_NO_GEOM_EXPERIMENT=13,      //13
    EXPERIMENT_CHECKERS=15,                             //15
    EXPERIMENT_CHECKERS_NO_GEOM=16,                     //16
    EXPERIMENT_CHECKERS_ORIGINAL_FOGEL=24,              //24
    EXPERIMENT_CHECKERS_SUBSTRATE_GEOM=28,              //28
    EXPERIMENT_END
};
```

Add a new entry at the end of the list for your new experiment.

6.3 Add the Experiment to ExperimentRun

At the top of the ExperimentRun.cpp file, add a #include for your experiment header file.

The setupExperiment(...) function inside the ExperimentRun class is resposible for creating experiments. There is a large switch statement which creates a new experiment based on the ExperimentType enum. Add an entry here for your experiment.

6.4 Create a new parameters file

In the NE/HyperNEAT/out/data/ folder there are many .dat files. Each of them provides parameters for a single experiment. Copy the CheckersExperiment.dat file to a new file (in this case, MyNewExperiment.dat), then open it and change the experiment ID to match the new entry you created in the ExperimentType enum:

```
MyNewExperiment.dat:

...
SignedActivation 1.0
ExtraActivationUpdates 9.0
OnlyGaussianHiddenNodes 0.0
ExperimentType XX.0          <--- REPLACE "XX" with new experiment ID
```

6.5 Test the Cloned Experiment

As a sanity check, run the cloned experiment for a few generations to make sure there are no errors:

```
./Hypercube_NEAT -I ./data/MyNewExperiment.dat \
-O TestOutputPopulation.xml -R 1000
```

6.6 Modify the Cloned Experiment to Do What You Want

This is entirely up to you :-)

7 Running the Visualizer

There are a few steps to running the visualizer and they can vary depending on your OS & setup. If you have trouble, please contact Jason Gauci or Kenneth Stanley using one of the aformentioned media.

7.1 Setting paths

Depending on where your PyHyperNEAT shared library is, you might have to adjust the relative path to this library:

```
sys.path.append('../../out/release')
^-- Assumes your PyHyperNEAT.dll or .so file is in /NE/HyperNEAT/out/release
from PyHyperNEAT import *
```

If python cannot find the PyHyperNEAT module after you've built the source with BUILD_PYTHON, it's because this directory is incorrect.

Next, you must set the paths for your input and output files. The visualizer takes .xml.gz files containing a HyperNEAT population, and can output images ready for publication. At the top of HyperNEATVisualizer.py file, change the populationFileName and outputDirName to match your population file and output directory:

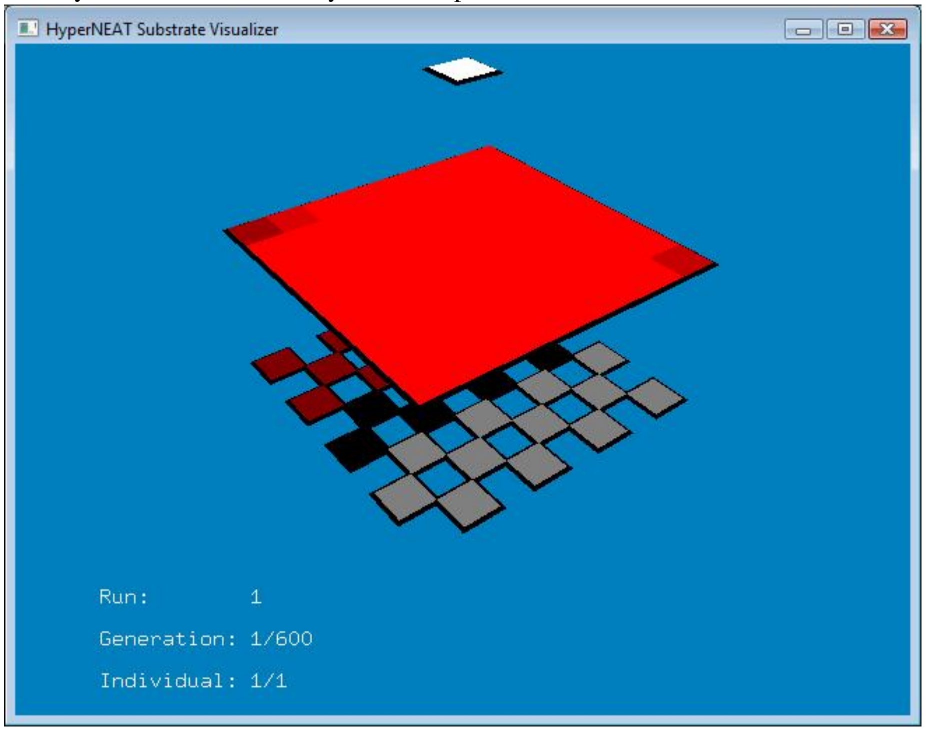
```
populationFileName = "C:/.../Run$RUN_NUMBER$_best.xml.gz"
#populationFileName = "/Users/pawn/.../Run$RUN_NUMBER$_best.xml.gz"

outputDirName = "C:/Programming/NE/HyperNEAT/out/images"
#outputDirName = "/Users/pawn/Programming/NE/HyperNEAT/out/images"
```

Note that you can put $RUN_NUMBER$ in the string and the visualizer will automatically substitute the current run number. This allows you to view several runs without modifying the code.

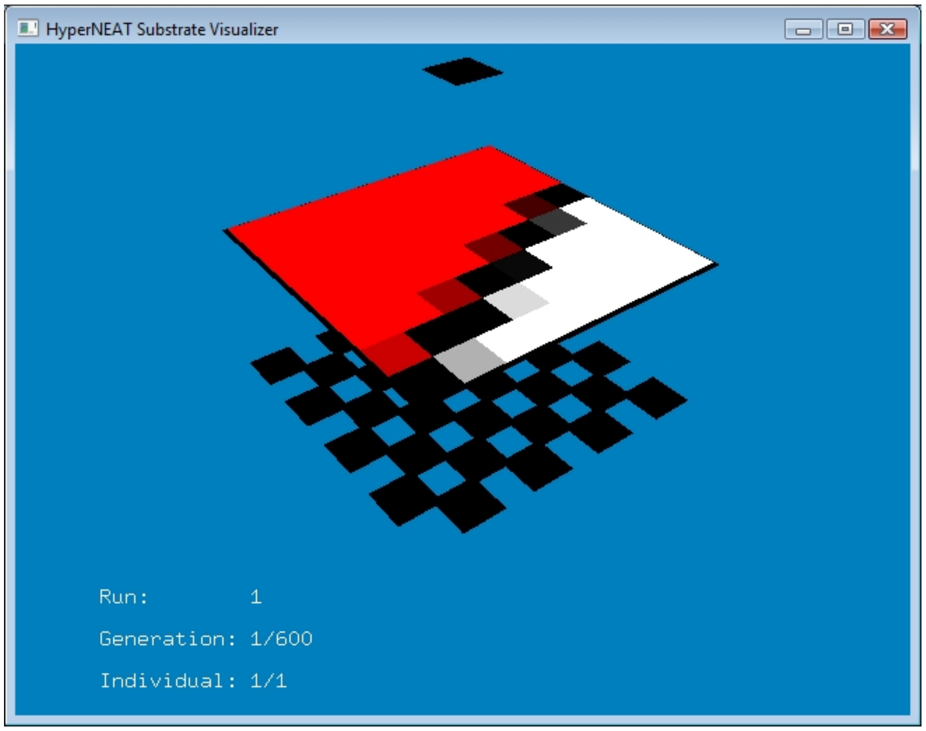
7.2 Visualizing activation levels and link weights

When you start the visualizer, you will be presented with a screen like this:



```
Run:        1
Generation: 1/600
Individual: 1/1
```

As long as the mouse is not on a square, the color of the squares determines the current activation level. Red squares denote negative activation and white squares denote positive activation. When you move the mouse over a square like so:



```
Run:        1
Generation: 1/600
Individual: 1/1
```

the colors will change and now represent the connection weights from the square the mouse is pointing at to that square. If a square is white, it means that there is a positive connection emanating from the square under the cursor to that square. If a square is red, it signifies a negative connection.

Left-Clicking on an input square will add 0.5 positive activation to that square. The other squares will update automatically. Right-Clicking on a square will subtract 0.5 activation (i.e. add 0.5 negative activation) to that square.

7.3 Camera controls

The arrow keys orbit the camera around the origin, while the ’q’ and ’e’ keys zoom in and out, respectively.

7.4 Changing individuals

The ’-’ and ’=’ keys will move backwards and forwards through the runs, respectively. The ’[’ and ’]’ keys will backtrack/advance the current run by one generation, while the ’{’ and ’}’ keys will backtrack/advance the run by ten generations. The ’,’ and ’.’ keys will backtrack/advance the current individual by one, and the ’<’ and ’>’ keys will backtrack/advance the current individual by ten.

7.5 Dumping images

The ’a’ key will dump images to the output directory. It will dump connection weights from the node that the mouse is currently pointing to (if the mouse is pointing to one) and it will dump activation levels of all sheets.

8 Contact Info

Please post comments/questions to the NEAT users group:

http://tech.groups.yahoo.com/group/neat/

The best way to reach Jason Gauci or Kenneth Stanley is through email (jgauci@eecs.ucf.edu and kstanley@eecs.ucf.edu respectively).