Intro Lab

This lab is meant to expose workshop participants to examples of problems which can be applied to the SpiNNaker architecture.

Installation

Begin by installing the software which is used to control the SpiNNaker board and run Neural Networks on it. The software installation instructions can be found at the following link. Please install the PyNN 0.8 version of the software:

https://spinnakermanchester.github.io/latest/spynnaker_install.html

Please then also install the build tools which will be used later in the workshop. These include the C compiler, spinnaker_tools, spinn_common and the SpiNNFrontEndCommon and sPyNNaker source code:

https://spinnakermanchester.github.io/latest/spynnaker_extensions.html

In addition, the visualisers used in this lab require the following steps to be done:

1. Install the OpenGL libraries (depending on your platform):
    a. On Fedora Linux (replace dnf with yum on older platforms):
    ```
    sudo yum install freeglut3-devel
    ```
    b. On Ubuntu Linux:
    ```
    sudo apt-get install freeglut3-dev
    ```
    c. On Windows 64-bit, from an administrative console run:
    ```
    pip install
    https://github.com/SpiNNakerManchester/SpiNNakerManchest
    er.github.io/releases/download/v1.0-win64/PyOpenGL-3.1.1
    -cp27-cp27m-win_amd64.whl
    ```
    d. On Windows 32-bit from an administrative console run:
    ```
    pip install
    https://github.com/SpiNNakerManchester/SpiNNakerManchest
    er.github.io/releases/download/v1.0-win32/PyOpenGL-3.1.1
    -cp27-cp27m-win32.whl
    ```
    e. On Mac OS X no installation is required
2. Install the visualisers (using sudo if you have done a central installation, or --user if you have done a user-only installation):
    ```
    [sudo] pip install SpyNNaker-Visualisers [--user]
    ```

File download

All of these examples can be found here:
https://spinnakermanchester.github.io/latest/intro_lab.html

Please download and open a terminal at the top level of the folder.

Run Applications

Below is a list of applications with the corresponding folders and execution commands, please run each script as it currently stands, and attempt to understand what the application is doing.

1. Neural Network Synfire Chain
2. Sudoku Game Through Neural Network
3. Balanced Random Network
4. Simple Learning Network

Neural Network Synfire Chain

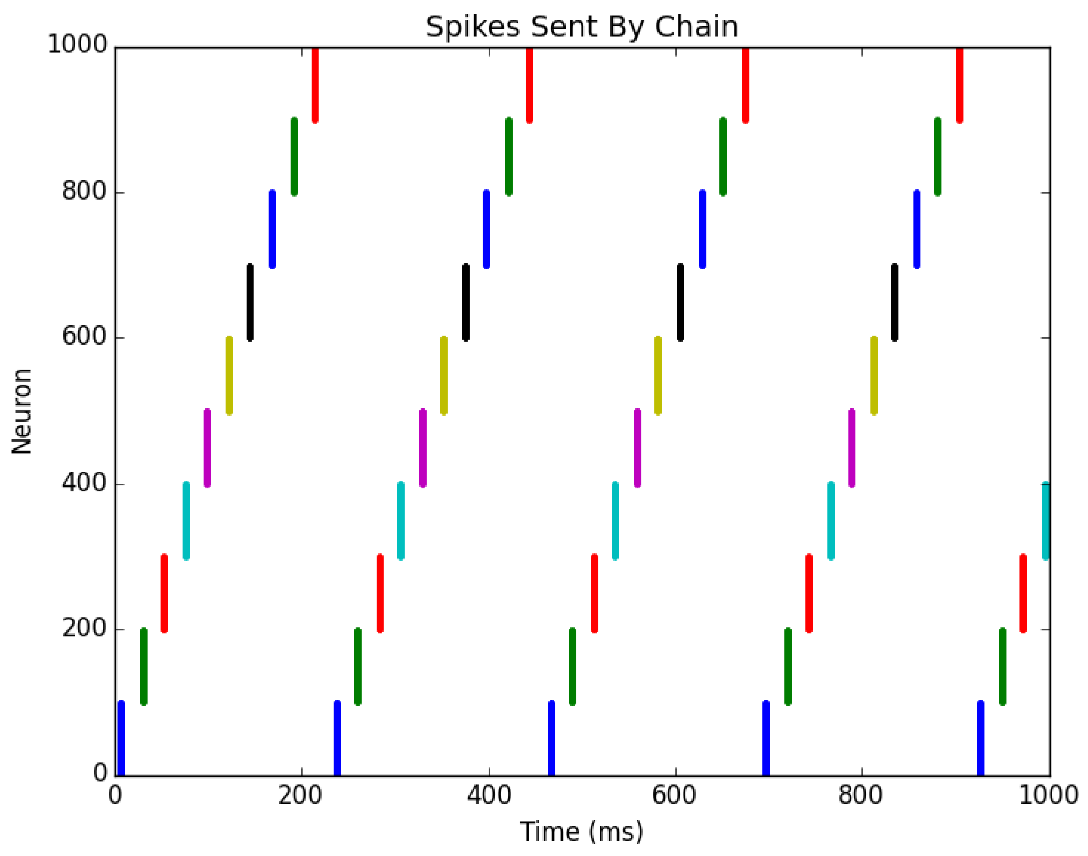


Figure 1: The output from a simple Synfire chain.

To run this example, from the top level of the folder type:

```
cd synfire
python synfire.py
```

A plot like the above should appear.

This example shows a PyNN Neural Network with a chain of 10 populations of 100 neurons each, where 10 neurons from each population excite all the neurons in the next population in the chain. The first population is then stimulated at the start of the simulation to start the chain running.

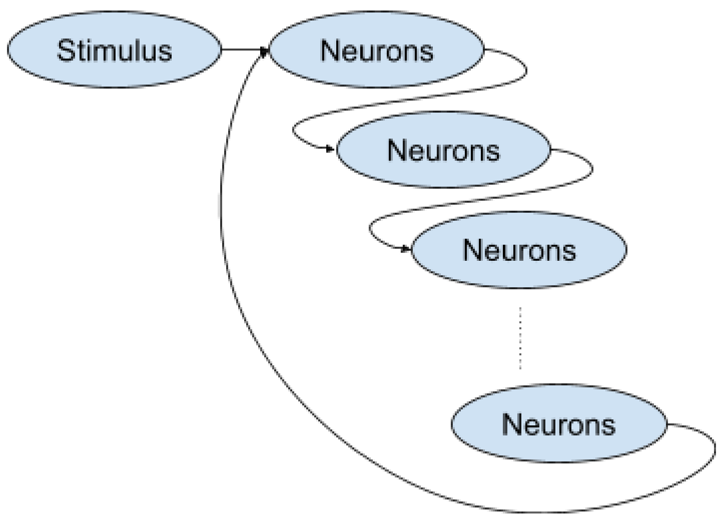
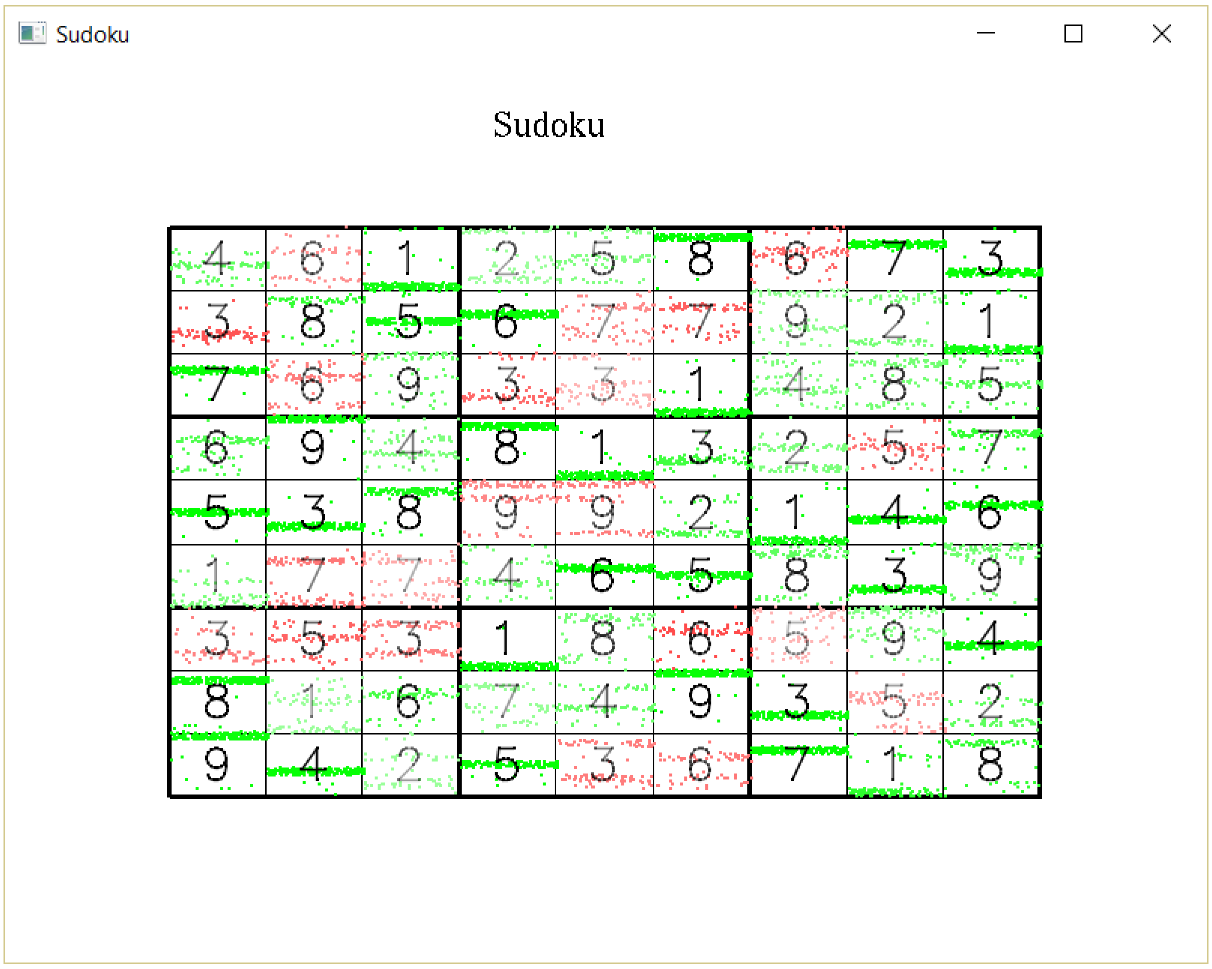


Figure 2: The Synfire Chain of Populations

Sudoku Game Through Neural Network



4	6	1	2	5	8	6	7	3
3	8	5	6	7	7	9	2	1
7	6	9	3	3	1	4	8	5
6	9	4	8	1	3	2	5	7
5	3	8	9	9	2	1	4	6
1	7	7	4	6	5	8	3	9
3	5	3	1	8	6	5	9	4
8	1	6	7	4	9	3	5	2
9	4	2	5	3	6	7	1	8

Figure 5: The output from the Sudoku game application

To run this example, from the top level of the folder type:

```
cd sudoku
python sudoku.py
```

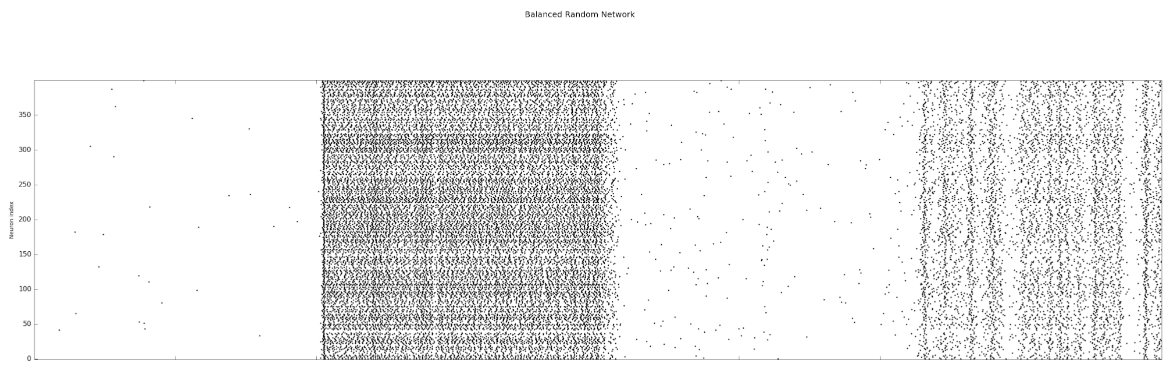
A visualiser will pop up, which is shown in Figure 5.

This example shows a PyNN neural network which describes a neural network for running sudoku problems. The spikes representing each cell are shown behind each number, with green output indicating that the value is valid according to the rules of Sudoku and red output indicating that the value is invalid. The problem to be solved is described near the top of the sudoku.py file, with 0s representing values to be computed. Note that on a small SpiNNaker board, the network is not always successful at solving the problem.

Balanced Random Network

To run this example, from the top level folder type:

```
cd balanced_random
python balanced_random.py
```

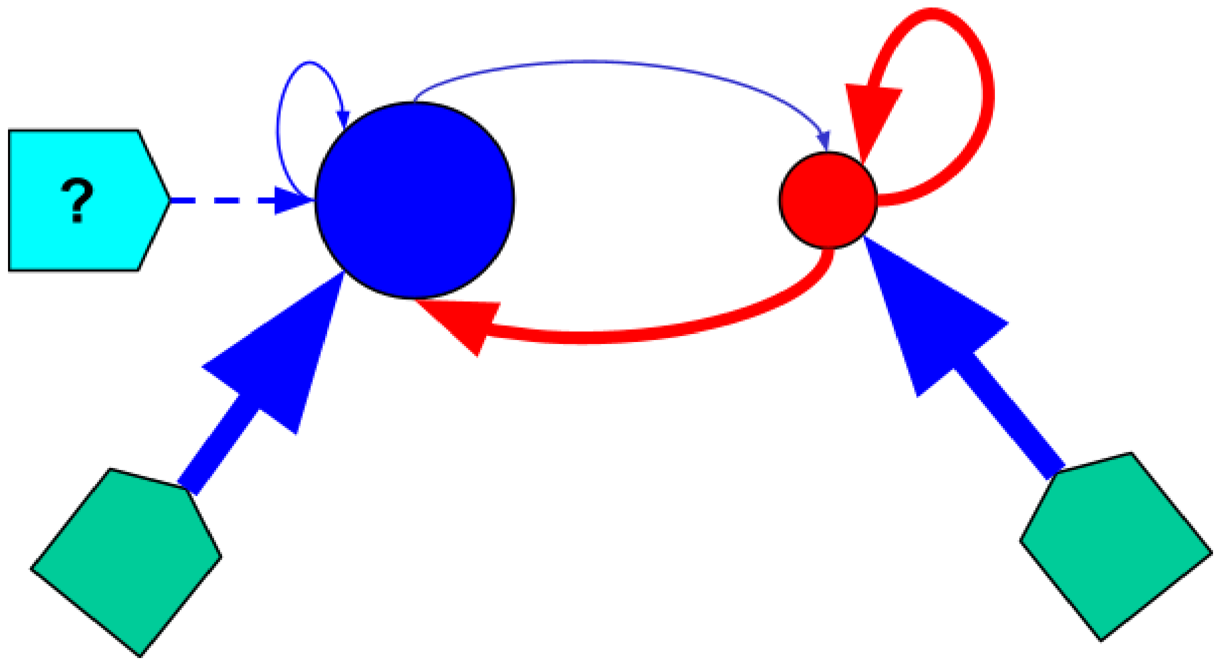


A plot like the above should appear.

This example shows a type of neural network known as a “Balanced Random Network”, which is believed to be representative of some areas of the brain. It is a combination of two populations, E and I, each stimulated to cause some random activity. These are then coupled to each other so that E *increases* the activity in I and I *decreases* the activity in E. Additionally, E *increases* the activity in itself and I *decreases* the activity of itself.

Without any external stimulation, random spikes are generated. If external stimulation is also added to E, it will spike even more. With a high rate of external input, the activity of E is high and the with a low rate, the activity is low, but with an intermediate rate, the network shows a more interesting pattern, with bursts of activity occurring at random times in the simulation.

The rates are changed in between runs of the simulation. You can change the existing rates and see the effect, or add additional runs with different rates.



Simple Learning Network

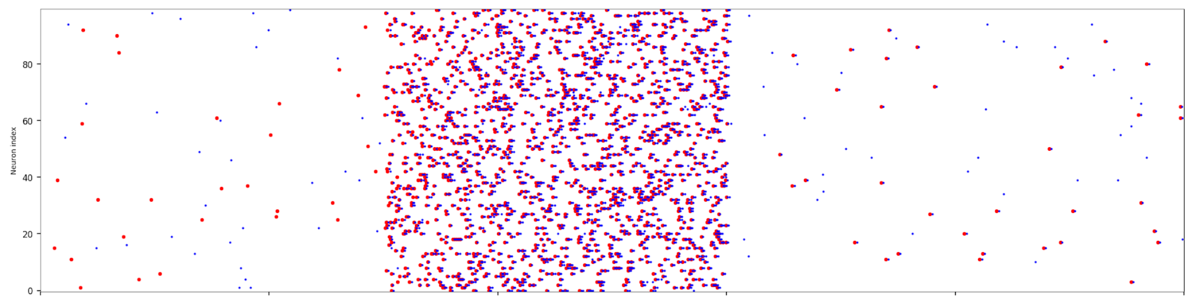


Figure 7: The output from the learning application

To run this example, from the top level folder type:

```
cd learning
python stdp.py
```

A plot like the above should appear.

This example shows the spike outputs from two populations of neurons that are connected together with a learning connection. At the start, both of the populations spikes regularly but the output is uncorrelated. In the middle, some learning is done; a training signal is applied to both populations and so both populations spike at a higher rate. At the end, whenever there is a red spike, there is a blue spike; the blue spikes still occur by themselves as well.