NeuCube

Neurocomputing Software/Hardware Development Environment for Spiking Neural Network Applications in Data Mining, Pattern Recognition, and Predictive Data Modelling

Knowledge Engineering and Discovery Research Institute
(http://www.kedri.aut.ac.nz)

Auckland University of Technology,

Auckland, New Zealand

June 2016

Disclaimer

NeuCube is a modular development system that helps graduate students, researchers, and practitioners to create new, more efficient solutions to problems of data mining, pattern recognition, event prediction, and decision support when dealing with complex and large data, especially temporal or/and spatio-/spectro-temporal data (SSTD) across domain applications. NeuCube uses the third generation of neural networks – the spiking neural networks (SNN) and the available neuromorphic hardware.

NeuCube is being developed and owned by the Knowledge Engineering and Discovery Research Institute (KEDRI, www.kedri.aut.ac.nz) and funded by the Auckland University of Technology Strategic Research Investment Fund (SRIF).

Using NeuCube for teaching and research will require a licence and a small charge depending on the version, to recover the costs of its development and to allow the developers to further improve it.

Commercial use will require a special licence as NeuCube is patent protected. Communication for obtaining any licence should be done through the person in charge (see the NeuCube web page: http://www.kedri.aut.ac.nz/neucube).

CONTENTS

1. INTRODUCTION

NeuCube is a software/hardware development environment for spiking neural network (SNN) prototype systems for data mining, pattern recognition, and predictive data modelling with complex and large data, especially temporal or/and spatio-/spectro-temporal data (SSTD). An application system created with the use of NeuCube has the architecture of a spatio- temporal data machine (STDM). In this respect, NeuCube is not a 'magic bullet' that can easily solve any of the above problems. Instead, it is a sophisticated framework of methods that facilitates the design and the implementation of efficient solutions to these problems through careful and precise selection and testing of most suitable methods and parameters for an STDM. This process can be slow, but the results can be very impressive in both accuracy and data understanding.

An STDM has three parts:

- an input part to encode input data into spiking sequences
- an SNNcube that learns the input data in an unsupervised mode to capture spatio-temporal patterns, and
- an evolving output part for classification or regression tasks that is trained in an incremental, adaptive way in a supervised or in a semi-supervised mode to classify (calculate) the SNNcube patterns into output classes (or regression values).

Additional modules can be added, such as a dynamic parameter model (e.g. a gene-regulatory network), or a module for personalised modelling.

STDMs are based on the principles of evolving connectionist systems (ECOS) and neuromorphic computations. This architecture was first proposed in (Kasabov, 2014), initially for brain data modelling as shown in Figure 1. NeuCube is PCT patent protected (Kasabov et al, PCT patent, 2013).

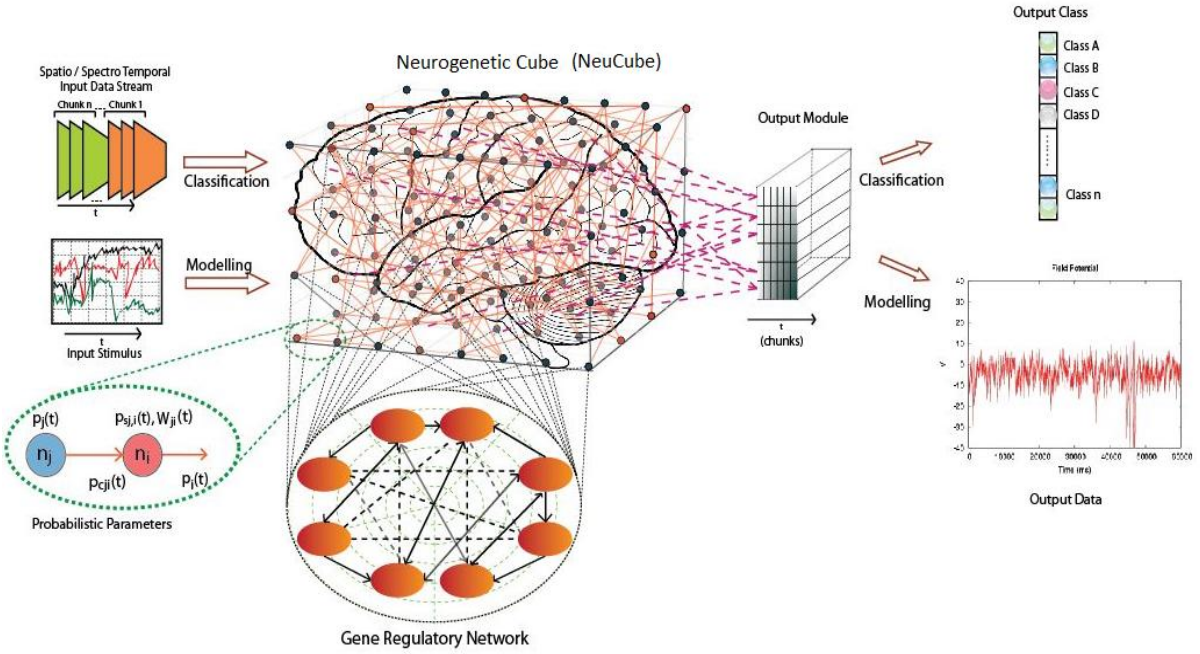

Figure 1: A functional diagram of the first NeuCube architecture introduced in (Kasabov, 2014; PCT 2013)

The above architecture was developed further as a multi-modular software/hardware development system for wider applications (Kasabov et al, 2015):

- Brain data modelling: EEG, fMRI, EMG, DTI, neurogenetic data, integrated brain data.
- Brain-Computer Interfaces.
- Robot control (e.g. neuro-rehabilitation robots with China Academy of Sciences).
- Prediction of neurodegenerative disease progression (e.g. Alzheimer's disease data).
- Personalised event prediction (e.g. stroke occurrence; CVD).
- Ecological event prediction from temporal climate data.
- Audio/Visual data processing.
- Moving object recognition.
- Hazardous environmental event prediction (e.g. risk of earthquakes).
- Radio astronomy data modelling (Pulsar detection) – SKA.
- Bioinformatics (e.g. spatio-temporal protein folding and functions).
- Financial and business data modelling and prediction.
- Others

Figure 2 presents an example of a multi-modular NeuCube development system, where different modules perform different tasks, but all of them are integrated through a common communication protocol. A brief description of the modules from Figure 2 is given below.

Module M1 is a generic prototyping and testing module, where an SNN application system (called here Prototype Descriptor) can be developed for data mining, pattern recognition, and event prediction from temporal or spatio-/spectro-temporal data (SSTD, or as it is called here Data) After creating and testing a prototype descriptor in M1, it is saved in the I/O **module (M5)** and can be used by the other modules.

Module M2 is a simulator for small and large-scale applications written in the PyNN programming language. It can read the Data created and tested in M1 or in other NeuCube modules and process it very fast. While this module can run independently, the use of the PyNN programming language also allows to run a Prototype Descriptor on specialised neuromorphic hardware (e.g. SpiNNaker; the INI ETH chip etc.), which is indicated in Figure 2 as **Module M3**. Module M3 makes it possible to execute a NeuCube Prototype Descriptor using parallel computing for fast on-line and real time applications.

Module M4 is a program written in Java for the 3D visualisation of a NeuCube Prototype Descriptor. It provides a virtual navigation through the 3D structure of a NeuCube model. It can also be used for visualisation through the specialised Oculus VR 'goggles'. Dynamic visualisation is possible when the SNN model is observed in action.

Module M5 is for the input/output of data and for interaction between all NeuCube modules.

Module M6 has all the functionality of module M1, but with specific additional functions for prototyping and testing of neurogenetic data models. These models can include relevant genes and proteins to study a problem in addition to only using brain data. This module is written in Java.

Module M7 facilitates the creation and testing of a personalised SNN system. It has all functionality of module M1, but also some special functions for personalised modelling (see Kasabov and Hu, 2010; Kasabov, USA patent 2008).

Module M8 has all functionalities of module M1, but it also includes some specific functions that support integrated brain data modelling, including EEG, fMRI, and DTI data analysis in combination.

Module M9 is an optional module for data encoding optimisation and event detection. This module includes state of the art data encoding techniques for mapping analogue signals to trains of spikes. It is specifically designed to optimise the input of data into the NeuCube.

Module M10 provides additional features for online learning and for real time data analysis and prediction, as it is commonly used in Brain-Computer Interface applications. Data can be added into the NeuCube constantly and it will produce results in real-time.

In this manual the terms "NeuCube v1.3" and "NeuCube-M1" are used interchangeably. The NeuCube v1.3 is an unrestricted version of the NeuCube Neurocomputing system and is a successor of NeuCube v1.2.

Further detailed information on the NeuCube architecture and its applications are given in (Kasabov et al, 2015). Various applications of NeuCube are published in the references provided at the end of the User Manual.

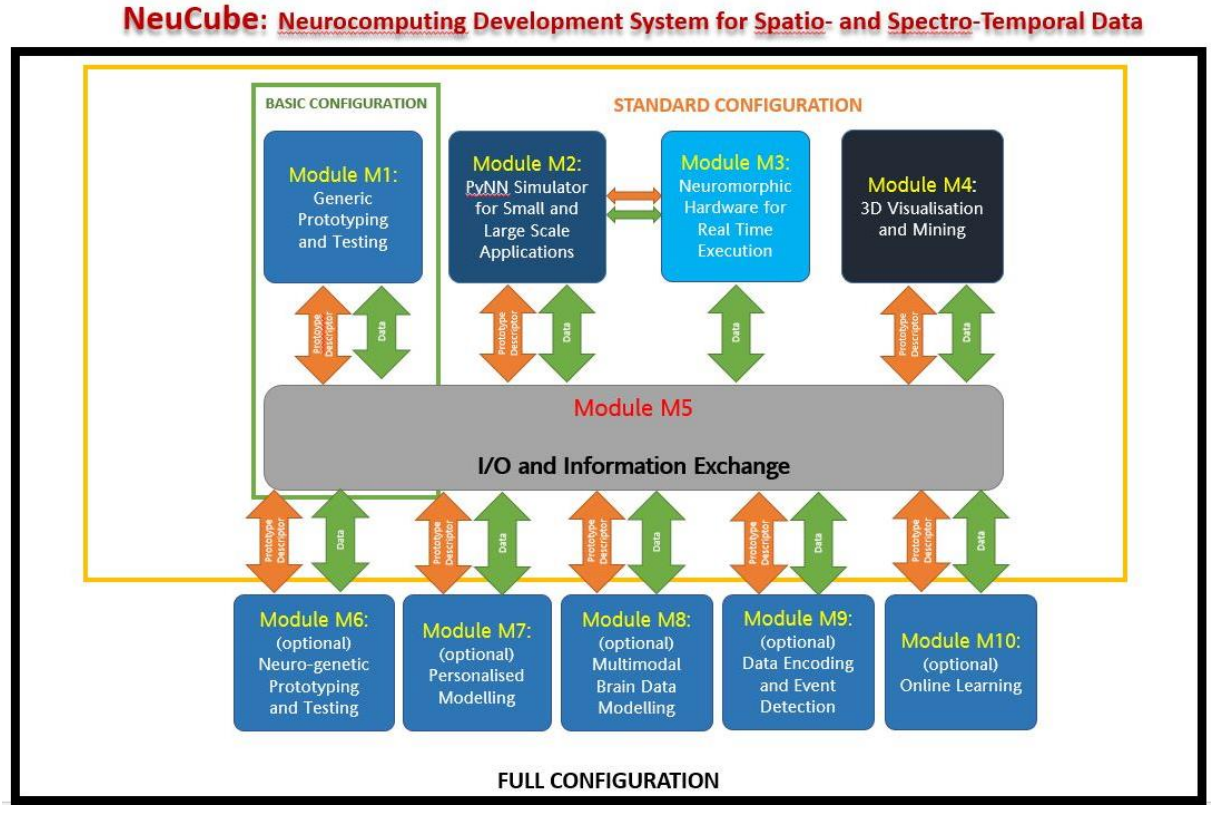

Figure 2: An example of a multimodular NeuCube development system for SNN applications

An example configuration of a NeuCube development system is shown in Figure 3, where modules M1, M2, M3 (a small SpiNNaker board), M4, and M10 are demonstrated.

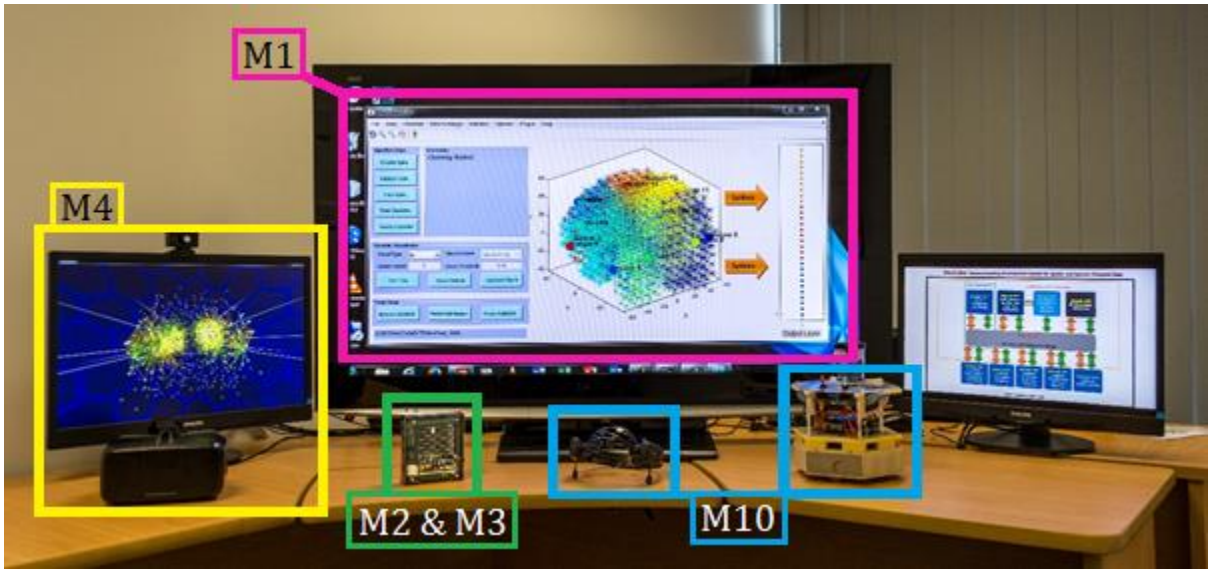

Figure 3: An exemplar configuration of a NeuCube development system is shown, where modules M1, M2, M3 (a small SpiNNaker neuromorphic hardware board) and M4 are demonstrated

NeuCube v1.1 was a limited trial version containing only Module M1.It is distributed as a windows executable file.

NeuCube v1.2 was an unrestricted version containing only Module M1 with some additional functionality. The unrestricted version does not have any software restrictions on the size of the data. It is distributed as a windows executable file, while NeuCube v1.3 and further versions (beyond 2016) will include additional functionalities.

NeuCube v1.3 is an unrestricted version containing Module M1. This module consists of several bug fixes and a significantly faster implementation of the learning algorithms.

NeuCube v2.1 will contain the Modules M1 and M4, while NeuCube v2.2 and further versions will have new functionalities added.

NeuCube v3.1 will contain the Modules M1, M4, M2 and M3 (optional), while NeuCube v3.2 and further versions will have new functionalities added.

NeuCube v4.1 will contain all modules shown in Figure 2, while further versions of it will have new functionalities added.

This manual briefly summarises the functionalities of the NeuCube-M1 module that constitutes the NeuCube v1.3.

2. NEUCUBE INSTALLATION

The NeuCube-M1 distribution can be installed and run on a Windows 7/Windows 8/windows 10 Operating System. A minimum memory of 4 GB is recommended. An internet connection is required for the installation.

NeuCube-M1 can be installed by running the NeuCube(v1.3)_installer.exe from the distribution folder. Double clicking on NeuCube(v1.3)_installer.exe runs the installer. The steps are self-explanatory and can be performed following the step by step procedure. If you do not have the Matlab Compiler Runtime (8.2/2013b) already installed on your computer, the installer automatically downloads and installs the Matlab Compiler Runtime (MCR). MCR is a large file (approx. 486 MB), and it may take some time (depending on the internet connectivity) for the installation procedure to be completed. Once the installation is completed, you can run the NeuCube-M1 module by double-clicking NeuCube from the 'Start' menu or from your Desktop.

Note: In case you get an "Could not fiend version 8.2 of the CMR. Attempting to load mclmcrrt8_2.dll. Please install the correct version of the MCR. Contact your vendor if you do not have an installer for the MCR." Error, please manually add the path of the Matlab compiler runtime (traditionally located in C:\Program Files\MATLAB\MATLAB Compiler Runtime\v82\runtime\win64) to your user or system path.

3. NEUCUBE USER INTERFACE

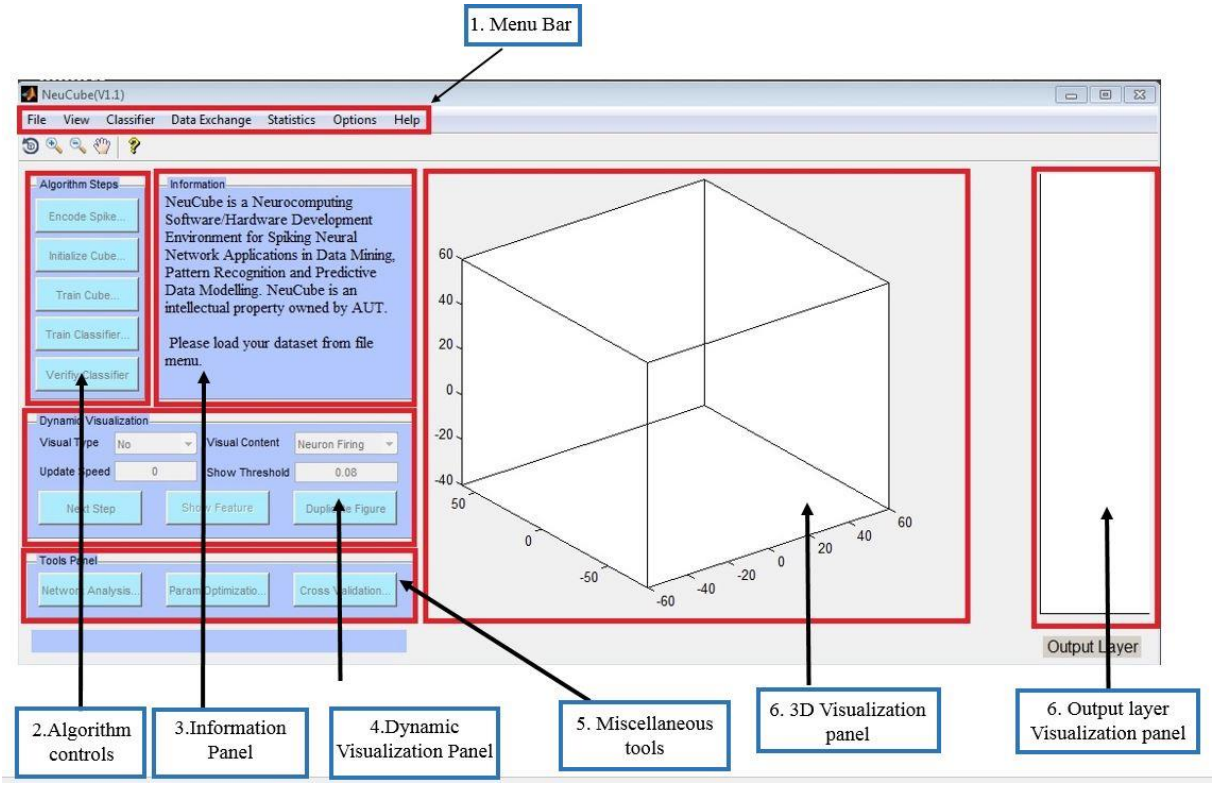

Figure 4: NeuCube-M1 module User interface and panel description

Figure 4 shows the user interface of the NeuCube-M1 module. It consists of seven different panels, which are:

1. **Menu bar**: The menu bar consists of several options for intra- and inter-modular information and data exchange, visualisation of the 3D reservoir, and the output layer.
2. **Algorithm controls**: Consists of controls to initiate the stepwise learning process in the design and the testing of an STDM.
3. **Information panel**: It is used to display information during a NeuCube model prototyping.
4. **Dynamic visualisation controls**: This panel consists of a set of UI controls which can be used to visualise the learning of the SNNcube (shown in panel 6) in real time during the 'train cube' phase of the process.
5. **Miscellaneous tools**: This panel initiates different functionalities, such as the network analysis toolbox, parameter optimisation, and cross validation.
6. **Visualisation panel**: This panel visualises the behaviour of the 3D SNNcube showing connections and neuronal spiking activity.
7. **Output layer visualisation panel**: This panel is used to visualise the behaviour of the output neurons created as a results of supervised training for classification or regression tasks.

4. DATA AND INFORMATION EXCHANGE

NeuCube supports several data formats which are used for intra- and inter-modular data exchange. Your choice of file format will influence the efficiency of NeuCube-M1.

The NeuCube-M1 interacts with the external environment using different data descriptors. The NeuCube architecture defines four types of data descriptors:

1. Dataset descriptor: The Dataset descriptor consists of the data (and the metadata) that is learned and analysed. In the majority of the cases, a dataset consists of a set of time series samples and the output label/value for the sample set. It is also possible to add miscellaneous information like 'feature name', 'encoding method' and other Meta information in the dataset.
2. Cube descriptor: The Cube descriptor contains all information about the structure and learning in NeuCube. Some of the most important information stored in this descriptor are the spatial information of the input and reservoir neurons, structural information of the SNNcube, and the state of the SNNcube during learning.
3. Parameter descriptor: The parameter descriptor stores all the user defined parameters including hyper-parameters of data encoding algorithms, SNNcube training algorithms, and classifier training algorithms.
4. Result descriptor: The Result descriptor stores information about the experimental results produced by NeuCube.

Descriptor type: The supported file formats are summarised in Table 1.	mat	JSON	CSV
Dataset	Yes	Yes	Yes
Cube	Yes	Yes	No
Parameter	Yes	Yes	No
Result	Yes	No	Yes

Table 1: Supported file format for descriptors

NeuCube-M1 supports three different file formats, mat (binary), JSON (structured text) and CSV (comma separated plain text). Table 1 shows the supported file formats for each of the descriptor type. As a heuristic rule, the mat format is recommended for achieving fast I/O. The CSV files are the recommended choice for the import/export of datasets and results. The JSON format is recommended for inter modular communication. The Dataset, Cube, and Parameter files can be imported or exported during the lifetime of the experiments run in NeuCube-M1.

5. PROROTYPE MODEL DESIGN AND TESTING, ILLUSTRATED WITH A DEMO PROBLEM

This section demonstrates the stepwise creation of a NeuCube prototype model design and testing. The data files that are used in this demo can be found in the folder data → wrist_movement_eeg.

Dataset description

The dataset used in this DEMO corresponds to a study participant moving their wrist either to the left or to the right, or holding their hand straight. This task was performed on a single subject and EEG data was sampled from 14 channels at a sampling rate of 128 Hz. 20 independent trials of 1 second duration were collected while the subject performed each movement task. The demo dataset consists of the following files:

- Sample files (mandatory): Each sample file (sam1.csv, sam2.csv, sam3.csv … sam60.csv in the example) contains data of one sample. Each sample corresponds to a data arranged in a matrix. The rows correspond to ordered time points, and the columns represent the features (in this case, EEG channels). Sample files in any user defined dataset should follow the following naming convention:
    - They should begin with the keyword 'sam' followed by a number which defines the order of the samples.
    - This can be followed by a __ character after which you can use any text.
    - The file must have a csv extension.
- Target file (mandatory): The target file stores the class label of each sample in a column, ordered in the same sequence as the numbers of the sample files. It should begin with the keyword 'tar' and have a csv extension. In this demo, the target file is called tar_class_labels.csv. A data folder should contain only one target file.
- SNNcube coordinate file (optional): This file describes the spatial coordinates of the neurons in the SNNcube. Every row in the SNNcube coordinate file contains the x, y, and z coordinates of a neuron. This file does not need a special name, but it has to be in csv format. In this demo, the brain_coordinates.csv file defines the coordinates of the spiking neurons according to the Talairach brain template; a template that makes it possible to represent any person's brain data in a standardised way. In this demo we use 1471 spiking neurons in the SNNcube representing $1cm^3$ spatial resolution.
- Input coordinate file (optional): This file describes the spatial location of the input neurons (features). Just like the SNNcube coordinate file, every row in the input coordinate file contains the x, y, and z coordinates of an input neuron. Again, it does not have to follow any naming conventions except for being a csv file. However, it is important to make sure that the input coordinates are a subset of the cube coordinates, so that they can be located in the cube and assigned their special role as input neuron. In this demo, the eeg_mapping.csv file is used as input coordinate file.
- Feature name file (optional): This file contains the names of the input features. In this example, it would contain a list of EEG channel names. This file should be a simple txt file with each line containing the feature name in the same order as they are used in the sample files. That way, it is easier to identify the input features during later analysis stages.

When creating your own set of files to use with the NeuCube, make sure they follow the same general structure of the required files as described above. Only the sample files and the target file have to be in the same folder, as they represent one dataset. The neuron coordinates and the feature names can be saved in another location on your computer for use in other projects.

Loading a dataset

NeuCube uses a stepwise process that begins by loading data from the File menu. For this you can choose between three options: classification, regression, or recall of an already trained model on new data. As shown in Figure 5, data can be loaded from the menu bar by clicking file → load dataset → classification/regression/recall.

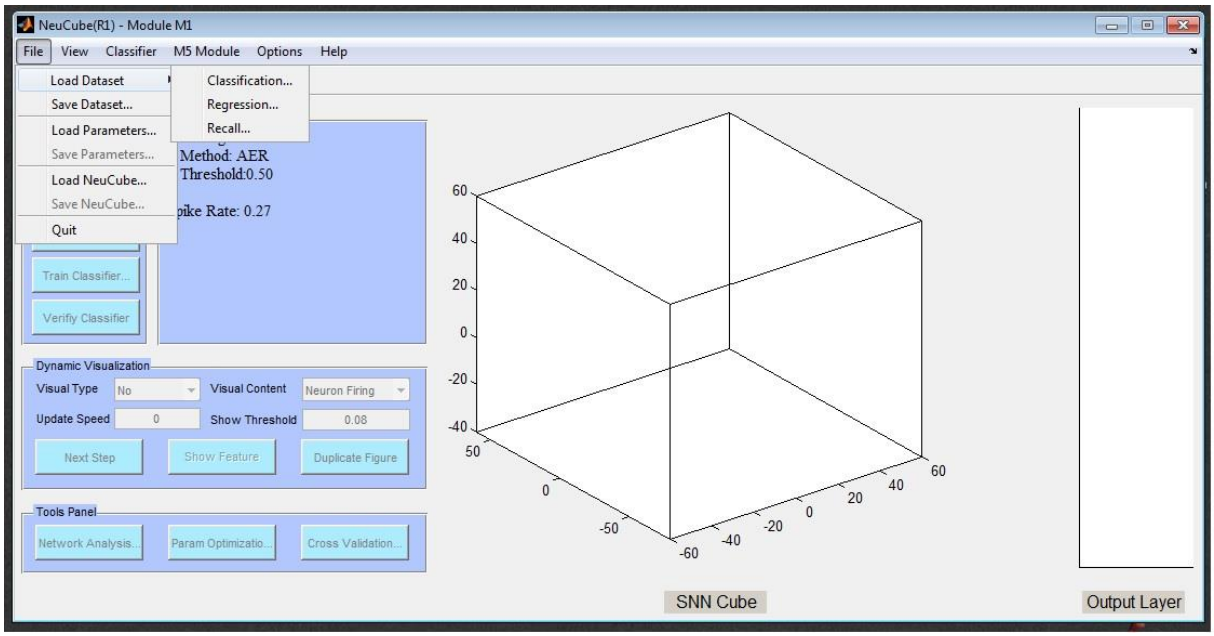

Figure 5. Loading a dataset

This example is a classification task, hence we click on 'classification'. This brings up the dialogue box shown in Figure 6. Click the 'browse' button to browse to the data folder containing the sample and target files. As described above, the data folder should contain one or more sample files and one target file. Please select the 'wrist_movement_eeg' folder and press 'OK' to load the sample into the NeuCube-M1 environment. Figure 7 shows how the metadata of the DEMO dataset is displayed in the information panel after the data was loaded successfully.

Figure 6: dataset parameters

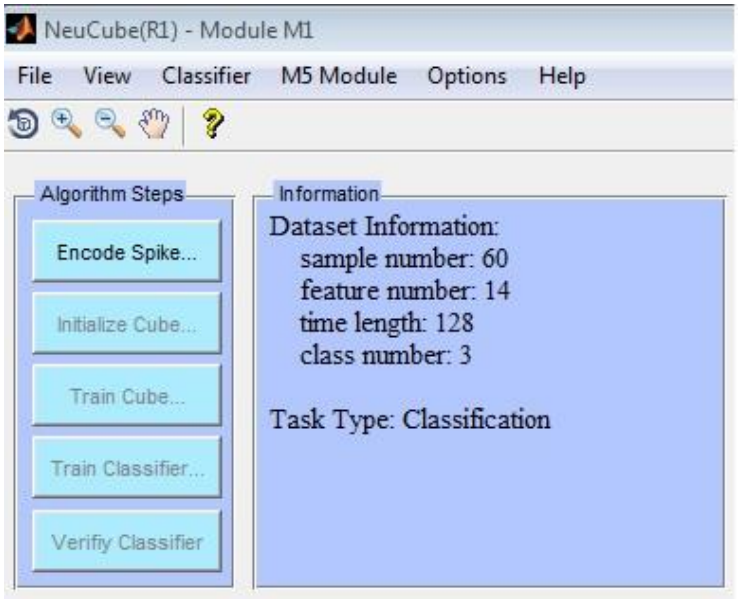

Figure 7. Information panel after data loading

In NeuCube v1.3 the size of each of the input data samples (the length of the temporal data points, or the number rows in a sample file) is fixed. For example, in this demo each sample is 128 time points long. However, in principle the NeuCube approach allows for samples of variable time length, for a flexible number of input variables (features) used in each sample, and for missing values. These additional options will be implemented in the next versions of the NeuCube v1.x.

Data encoding

The next step after data loading is to encode the real-value input data into trains of spikes, which will be used by the learning algorithms to learn spike patterns in the SNNcube (unsupervised learning), and after that also in the output module (classifier/regressor). Clicking the 'Encode Spike' button generates a new UI panel as shown in Figure 8. This panel is used as follows:

1. Choose the encoding algorithm: NeuCube-M1 implements different encoding methods, which can be chosen from the 'Encoding Method' dropdown menu. The hyper-parameters like 'Spike Threshold', 'Window Size' etc. can be tuned for the chosen algorithm.
2. Choose dataset split: This panel is also used to specify the random split of the dataset for training and validation (Training Set Ratio). The 'Training Time Length' and 'Validation Time Length' parameters define the percentage of the temporal length of the samples used for training and validation correspondingly (e.g. 0.8 means 80%; 0.7 means 70%, etc.). This option allows for a testing to be done on shorter temporal samples, when the model is expected to predict an event, for example based only on 70% of new input data.
3. Encoding visualisation: This panel is used for specifying options for data encoding visualisation. NeuCube v1.3 supports visualisation of raw and encoded data for one feature and one sample.

In this demo, the default values can be used. Please press 'OK' to encode the data. After completion, the data encoding is visualised in the visualisation panel as shown in Figure 9. The graph on top shows the raw input data for the chosen sample and feature, and the bottom graph shows the positive and negative spike trains generated from the raw data.

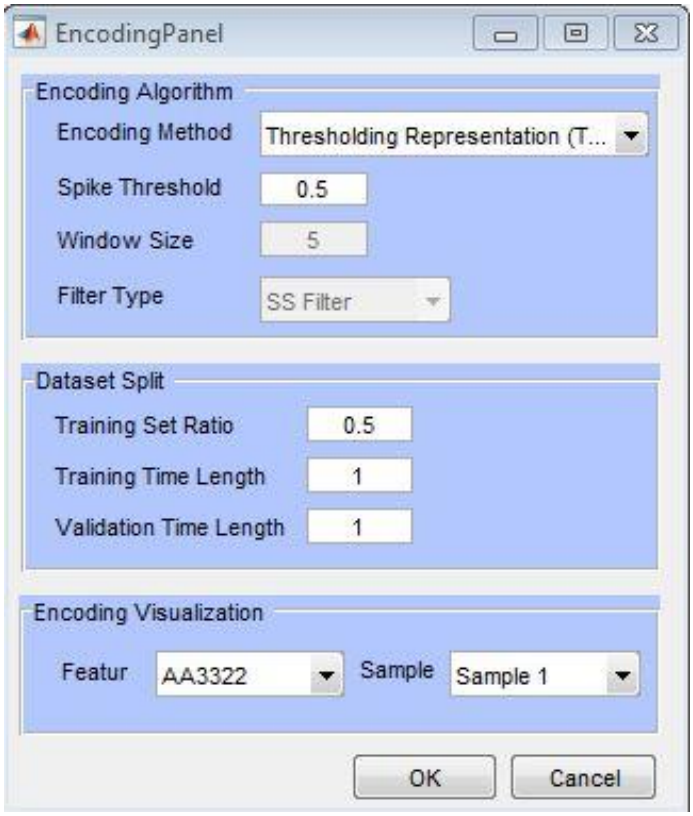

Figure 8. Encoding panel

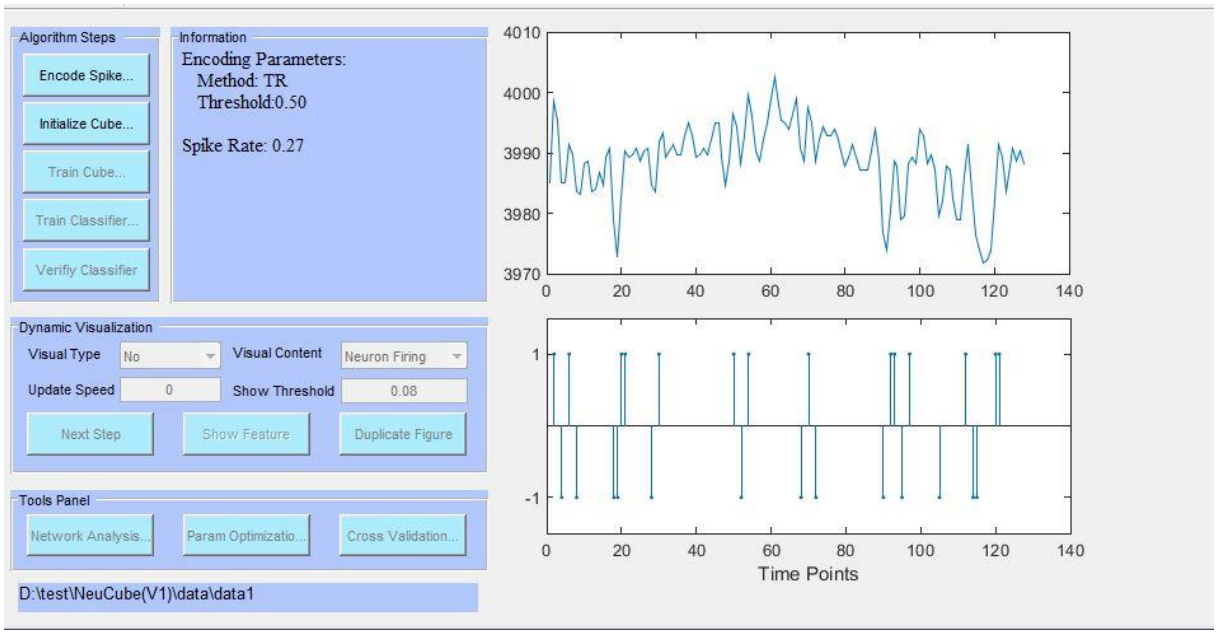

Figure 9: Spike encoding visualisation

Cube initialisation

Once the data is encoded, the next step is to initialise the SNNcube, which can be done by clicking the 'Initialise Cube' button. This evokes the cube initialisation panel as shown in Figure 10. The subpanel highlighted in red is used to configure the properties of the neurons in the cube. Coordinates of the neurons in the SNNcube can be defined automatically (using a graph method), manually (by specifying how many neurons shall be created for x, y, and z coordinates), or by

loading the coordinates from a file (described above as SNNcube coordinate file). Select your preferred option from the 'Neuron coordinates' dropdown menu. The subpanel highlighted in blue shows the coordinates of the input neurons. These coordinates can be mapped automatically, loaded from a file (described as Input coordinate file above), or defined manually by using the 'given by' dropdown menu.

The connection between the neurons in the SNNcube are initialised using the small-world connectivity (SWC) approach, where a radius is defined as a parameter for connecting neurons within this radius (SWC parameter), with small weight values attached to the connections which are 80% positive. Another parameter, LDC (long distance connectivity), can be used to initialise connections beyond the radius of SWC.

In this demo we load the mapping for the reservoir neurons using the 'load from a file' option and choosing the file 'brain_coordinates.csv'. Similarly, the input neuron mapping is chosen by loading the file 'eeg_mapping.csv'. Click 'OK' once the parameters are set for initialisation.

Once the initialisation is finished, the '3D visualisation' panel shows the initialised cube as shown in Figure 11. Clicking the 'Show feature' button shows the spatial location of the input neurons in the 3D SNNcube.

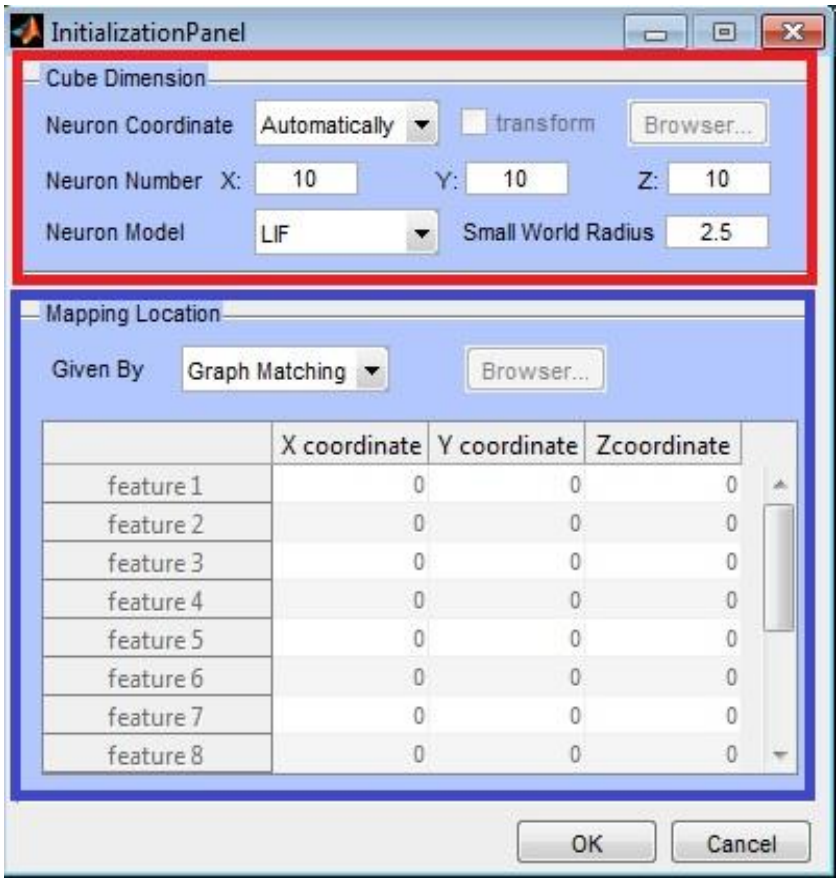

Figure 10. SNNcube Initialisation and mapping panel

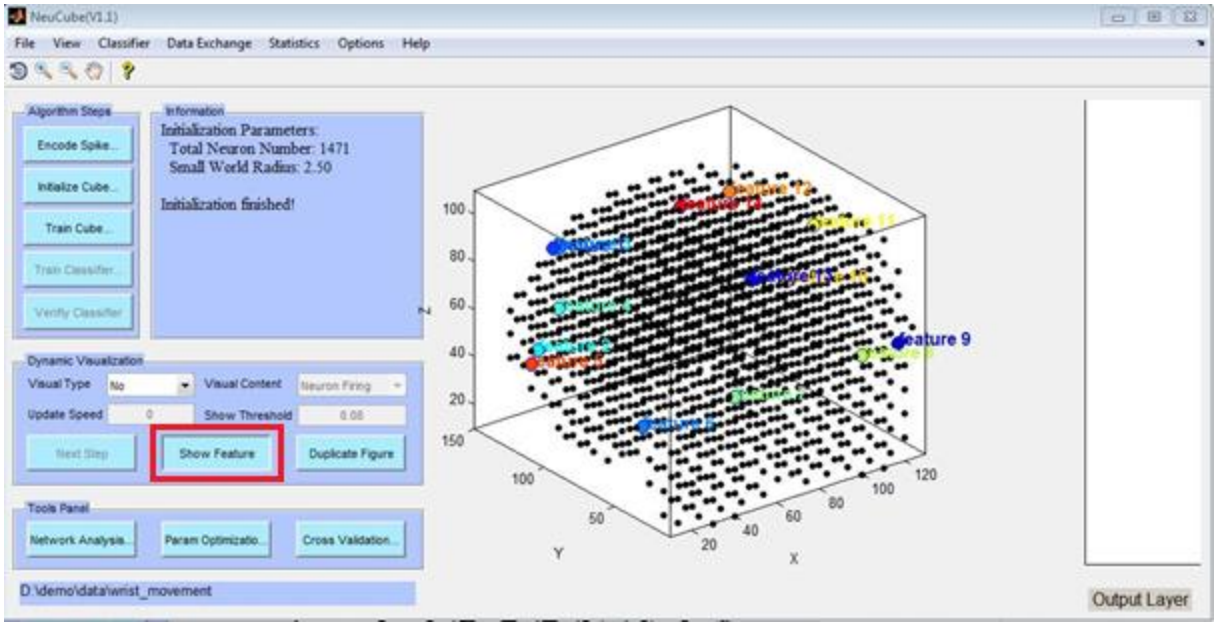

Figure 11. SNN cube after initialisation

Training of the SNN cube

The next step is the unsupervised training of the SNNcube that will create connections between the neurons based on the input spikes. Clicking on 'Train Cube' shows the UI for setting hyper-parameters for the training as shown in Figure 12. The hyper-parameters have the following meanings:

- Potential leak rate: This parameter defines the leak in membrane potential of a spiking neuron when the neuron does not fire.
- Threshold of firing: This parameter defines the threshold membrane potential beyond which the neuron fires a spike.
- Refractory time: This parameter defines the absolute time (in time units) during which the neuron will not fire. This refractory period begins after a neuron has fired a spike.
- STDP rate: This parameter defines the learning rate of the STDP learning.
- Training round: Describes the number of iterations for unsupervised learning in the cube.
- LDC probability: Defines the probability of creating a long distance connection.

Once the hyper-parameters for the unsupervised learning are set, click 'OK' to start the unsupervised learning. This step may take time based on the size of the dataset and the computer platform used.

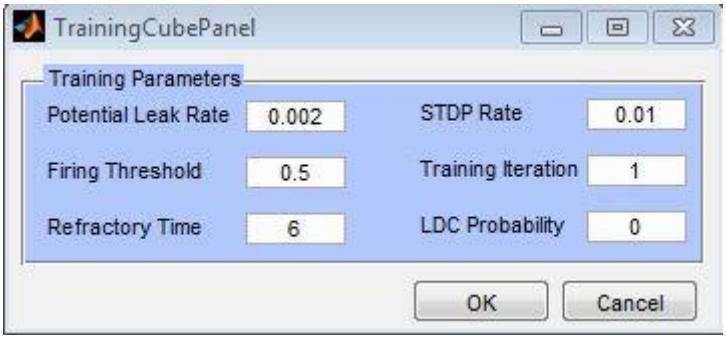

Figure 12. Training SNNcube panel

Dynamic visualisation of learning

The unsupervised learning process can be visualised dynamically while the system is learning, or can be saved as a movie file for later usage and analysis by using the 'dynamic visualisation' panel as shown in Figure 13. The visualisation can be displayed continuously or stepwise. The 'visual content' dropdown list specifies the type of activity to be rendered, like 'Neuron firing', 'Synaptic evolution', or 'Weight changing'. The 'Show Threshold' option sets the minimum connection weight for the connections to be shown.

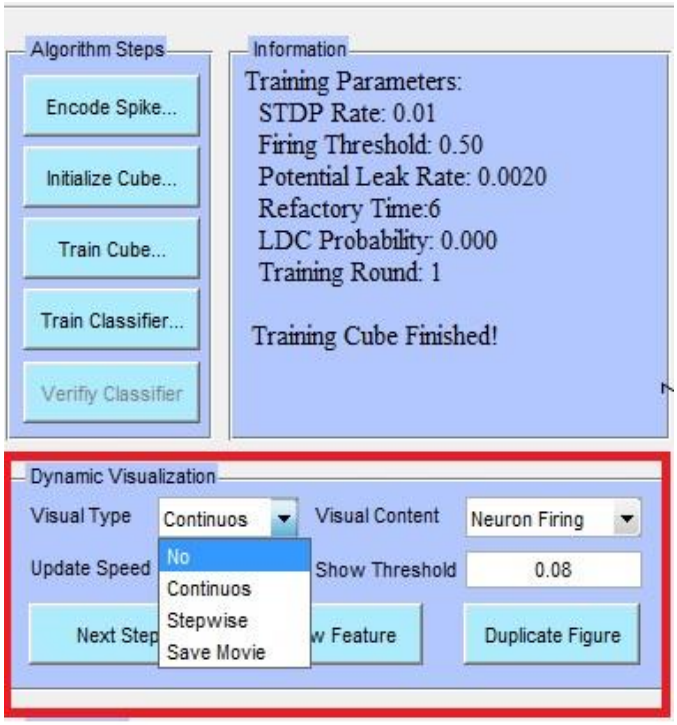

Figure 13. Dynamic visualisation panel

Analysis of the cube

Once the unsupervised learning finishes, there are several options in NeuCube for connectivity analysis and visualisation.

Analysis/visualisation of the SNNcube connectivity:

The final state of the cube at the end of learning can be analysed and visualised by clicking 'View' in the menu bar (Figure 14).

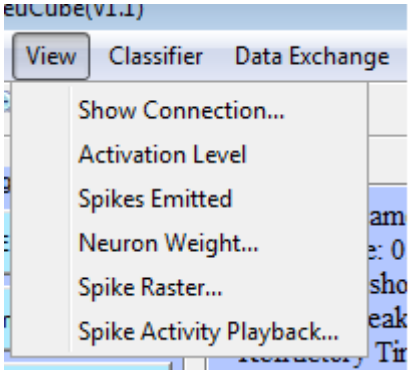

Figure 14. Module M1 visualisation menu

1. Clicking 'Show Connection' and choosing a threshold displays the connection above a threshold value, as shown in Figure 15 for the demo dataset.
2. Clicking 'Activation Level' shows the membrane potential of the neurons. Figure 16 shows the spike activation level of the neurons for the demo dataset. An explanation of the visual elements used in each figure can be found in the information panel.
3. Clicking 'Spikes Emitted' shows a histogram of positive and negative spikes emitted by all SNNcube neurons. Figure 17 shows the spike emission histogram generated for the demo dataset.
4. The 'Neuron Weight' option allows the user to specifically choose a neuron ID and visualise the connection weights of all neurons connected to the chosen neuron.
5. The 'Spike Raster' option plots a temporal raster plot of the training data. The raster plot takes the sample number as input and generates the raster plot for the specific sample as shown in Figure 18.
6. The 'Spike Activity Playback' option allows the user to dynamically visualise the spike dynamics over time. Clicking on the 'play' button plays the spike activity from beginning of the simulation till the end. The scrollbar can be used to see the spike activity in forward or backward order in stepwise manner.

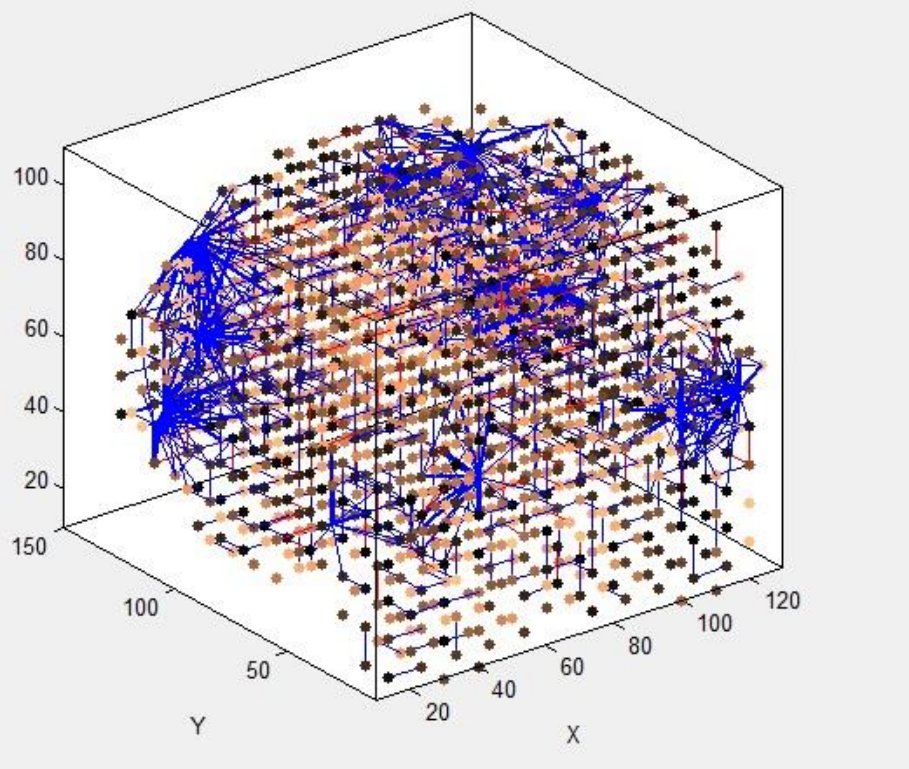

Figure 15. A snapshot of the connnections in the SNNcube after unsupervised learning. The connections represent spatio-temporal relationships between input data variables over time.

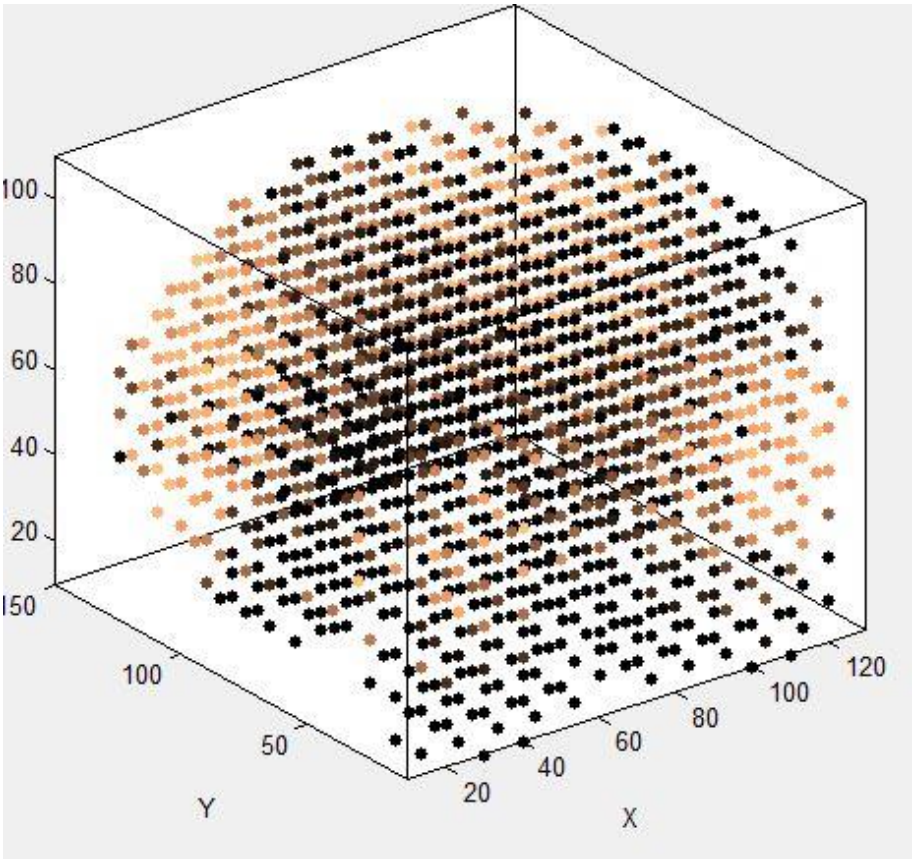

Figure 16. Activation level of the neurons in the SNNcube after unsupervised learning. The brighter the colour of a neuron, the higher its activation level is in terms of number of spikes emitted.

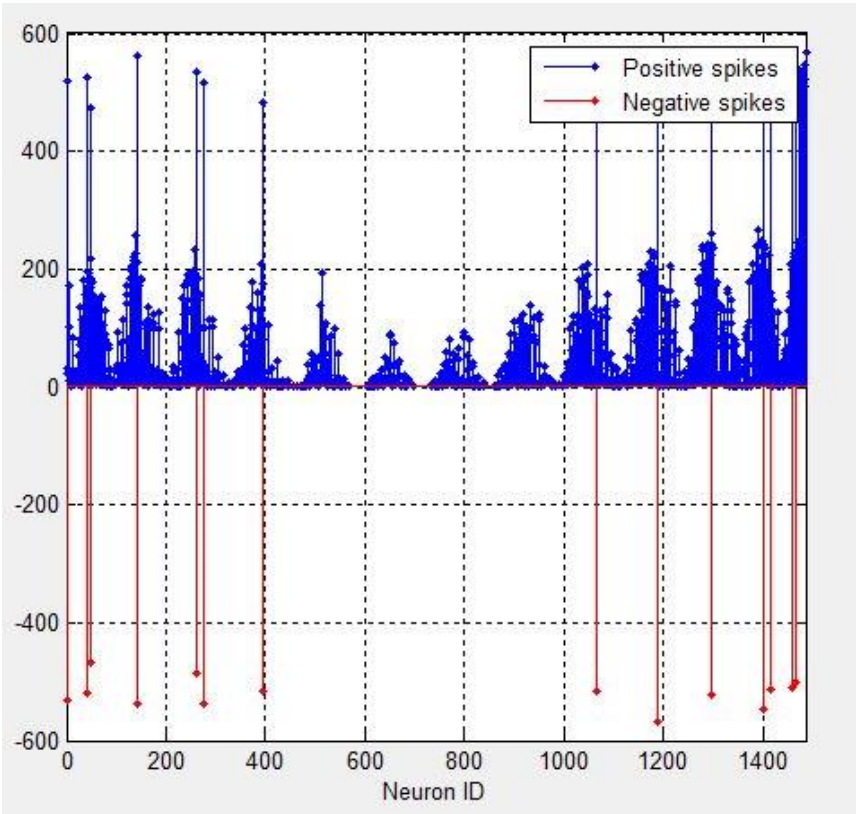

Figure 17: Positive and negative spike emission histogram for all the SNNcube neurons

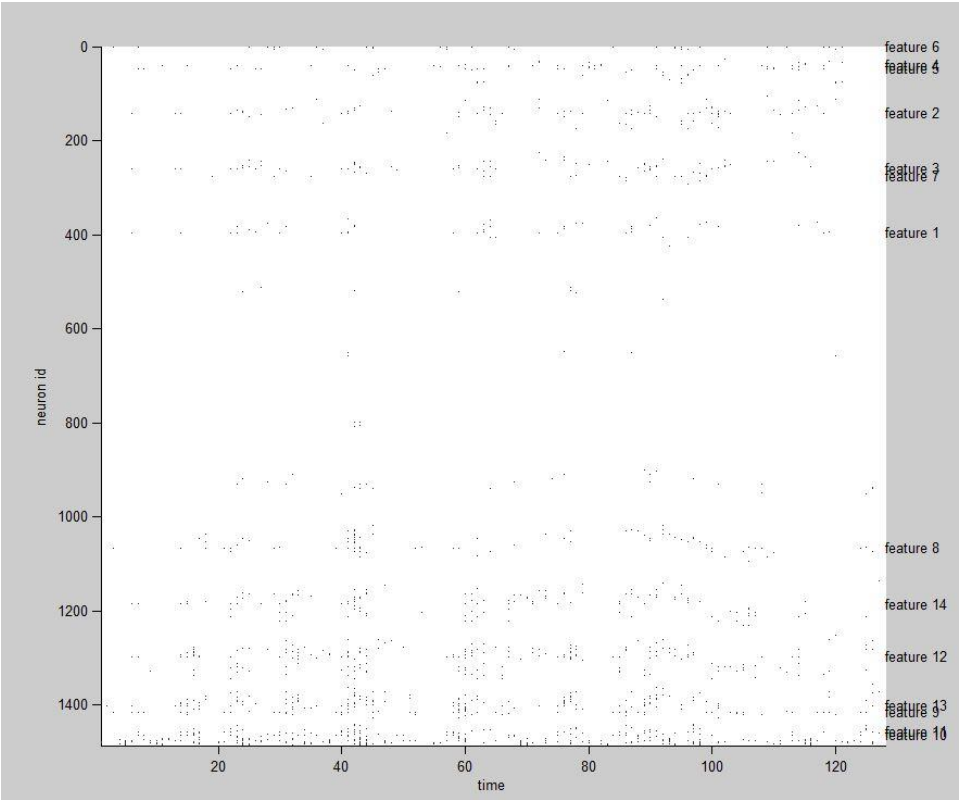

Figure 18: Raster plot of sample 7 spike output by SNNcube

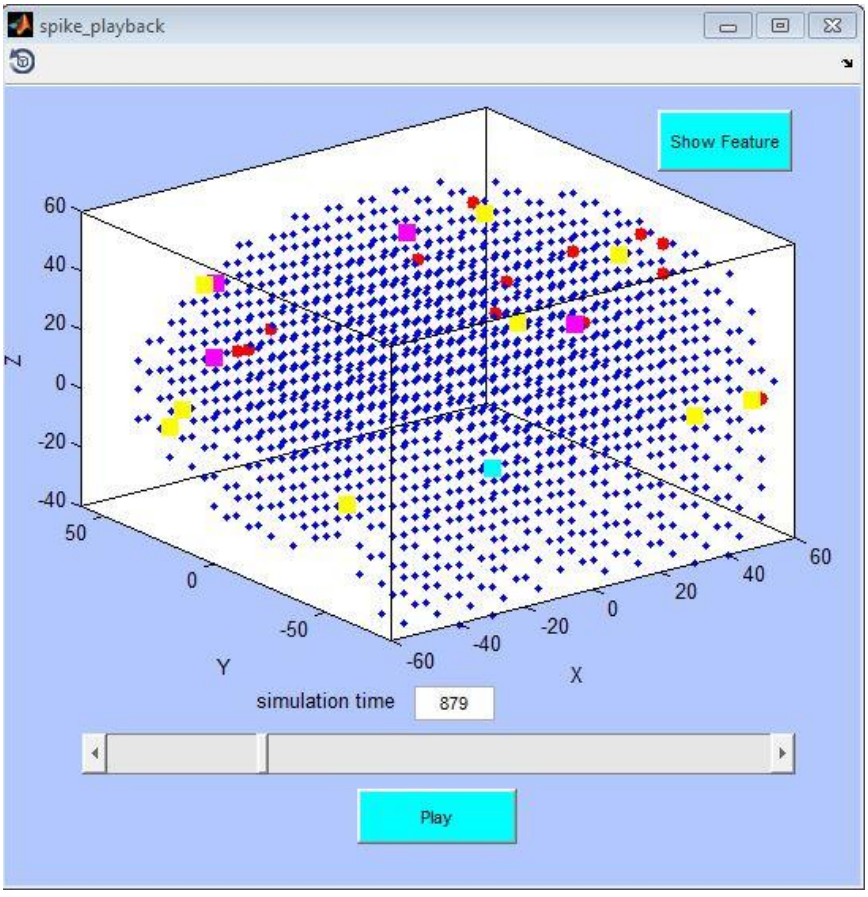

Figure 19: Spike playback panel.

Analysis through the 'network analysis panel:

Analysis of the learned SNNcube network can be performed using the network analysis toolbox, which is initiated by clicking the 'Network Analysis' button in the tools panel. Figure 20 shows the UI for the network analysis panel. The network analysis toolbox offers several methods of analysis, which are described in detail below:

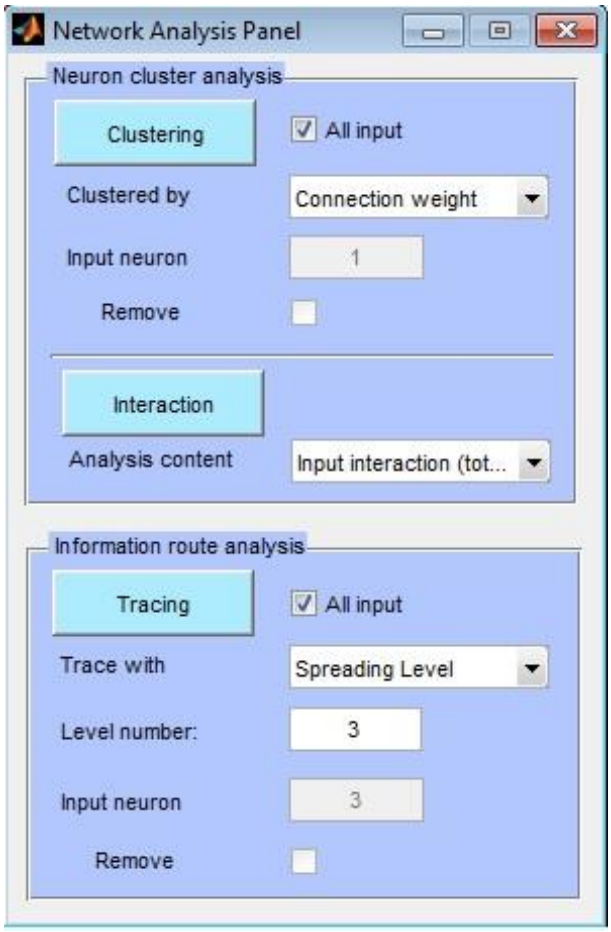

Figure 20: Network analysis panel

- **Neuron cluster analysis**: This option is used to analyse clusters of neuron-surrounding input neurons. The clustering of the neurons can be done by two methods.
    - Connection weight: This is the synaptic weight between a pair of neurons. It is adjusted during unsupervised learning to reflect the interaction between the neurons. Figure 21 shows the clustering by connection weights for the learned cube with the demo dataset.

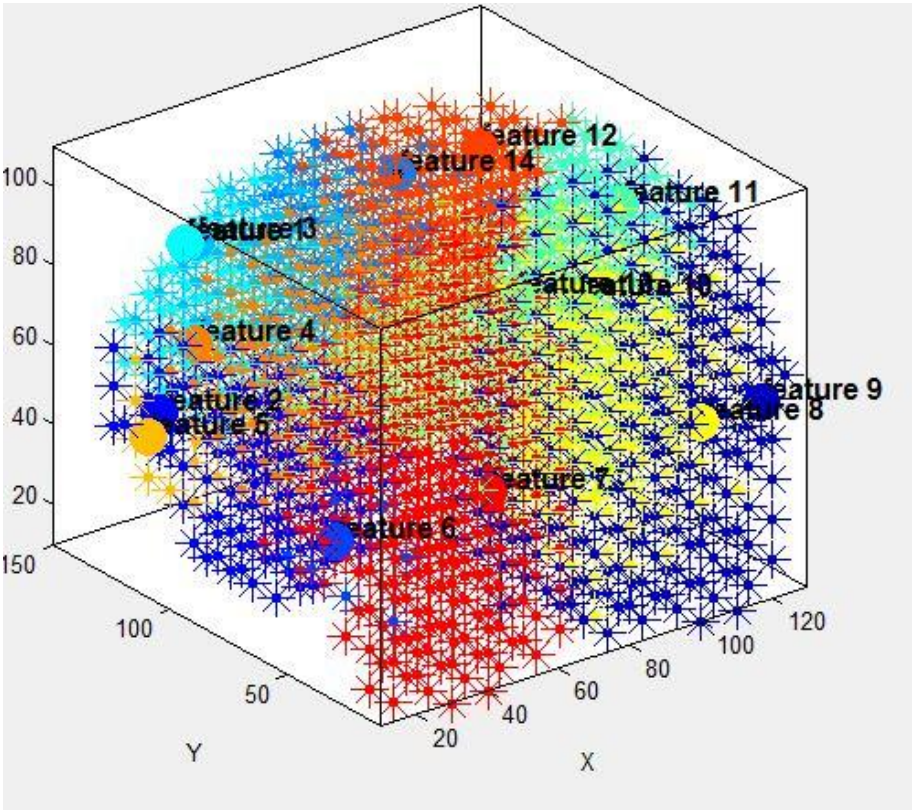

Figure 21. Clustering by connection weights for each input variable that represents a cluster centre.

- Spike communications: This is the spike amount communicated between a pair of neurons. Figure 22 shows the clustering by spike communications for the learned cube with the demo dataset.

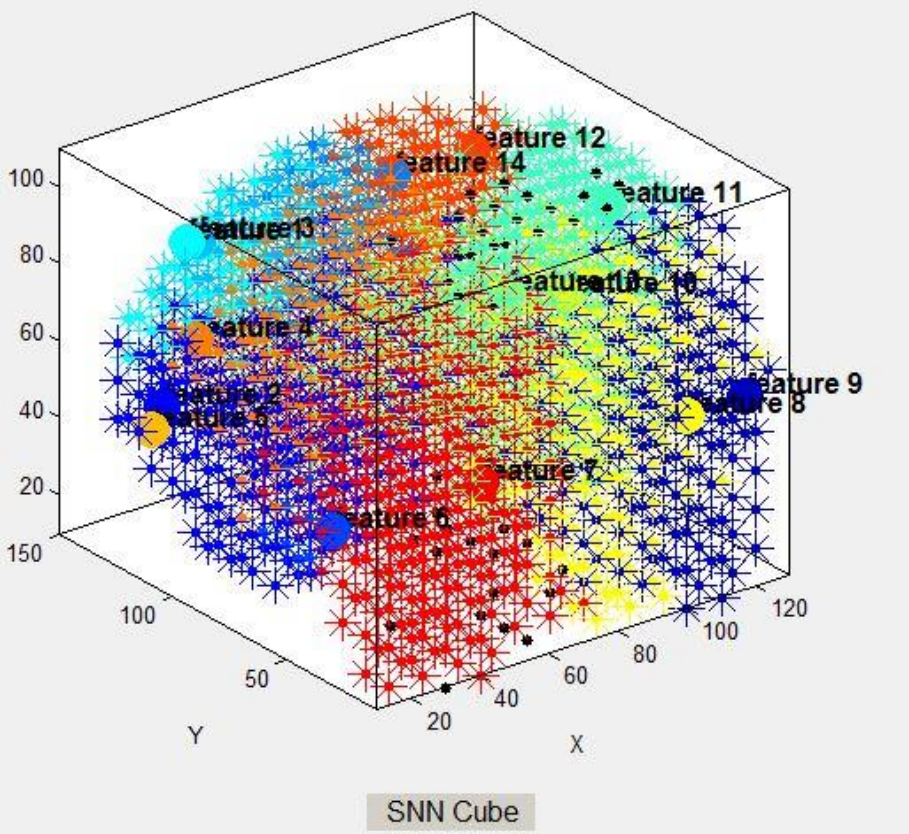

Figure 22. Clustering by spike communication

Another analysis option is to show the interactions between the neurons by choosing from the 'analysis content' dropdown and clicking the 'Interaction' button. Interactions are analysed using the following metrics:

- Input interaction (total): Shows the total interaction between the input neuron clusters given by the cluster analysis (connection weight/spike communication) as explained previously.
- Input interaction (average): Shows the average interaction between the input neuron clusters given by the cluster analysis (connection weight/spike communication) as explained previously.
- Neuron proportion: Shows the percentage of neurons in the cube which belong to an input neuron cluster.

Figure 23 shows the average one-to-one interaction between the input neurons based on average input interaction for the demo dataset. Thicker lines indicate more interaction.

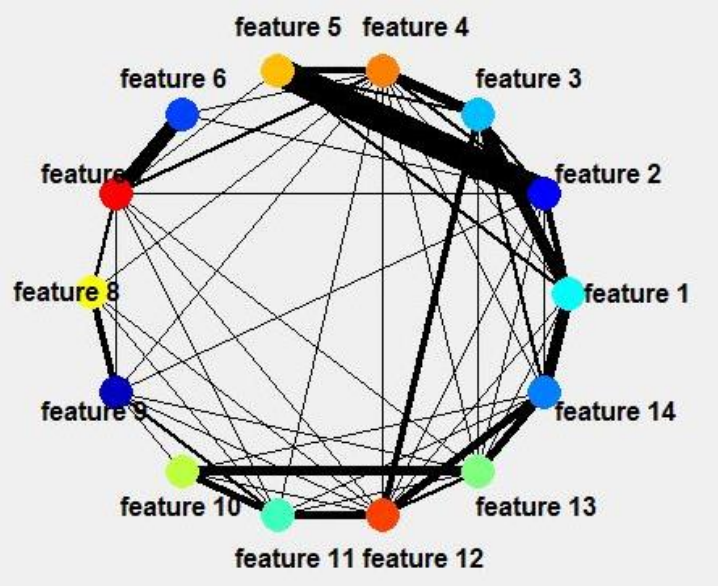

Figure 23. Spike interaction based on 'average input interaction'

Information route analysis: This option is used for analysing the information propagation route of the spikes. This analysis is based on the concept of a rooted tree structure. A rooted tree is defined as a directed tree having a single root node (neuron). Figure 24 shows an example of a rooted tree. The neuron in yellow is the root neuron. A neuron's 'parent neuron' is defined as a neuron which is one step higher in hierarchy and lying on the same branch. For example, the parent neuron of any green neuron is the red neuron to which it is connected. Similarly a 'child neuron' is defined as a neuron which is one step lower in hierarchy and lying on the same branch. For example, the child neuron of any red neuron is the green neuron to which it is connected.

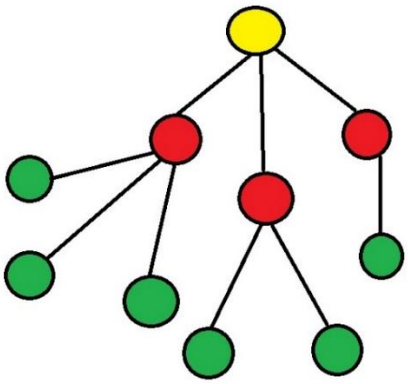

Figure 24: Rooted tree structure

The type of information to be shown can be chosen by selecting an option from the 'trace with' dropdown menu. Different methods of analysis are available:

- Max spike gradient: This visualisation shows a tree rooted by the input neuron, where a child neuron is chosen to be connected to a parent neuron if it receives spike from its parents.
- Spreading level: This visualisation shows a tree from the input neuron to its neighbourhood which reflects the spreading of the spikes. The 'level number' parameter

defines the neighbourhood of spread. For example, setting this parameter to 2 will show the spike distribution from the input neuron to two layers of neighbouring connected neurons. Figure 25 shows an example of spreading visualisation for input neuron/feature 3 with a spread up to level 3.

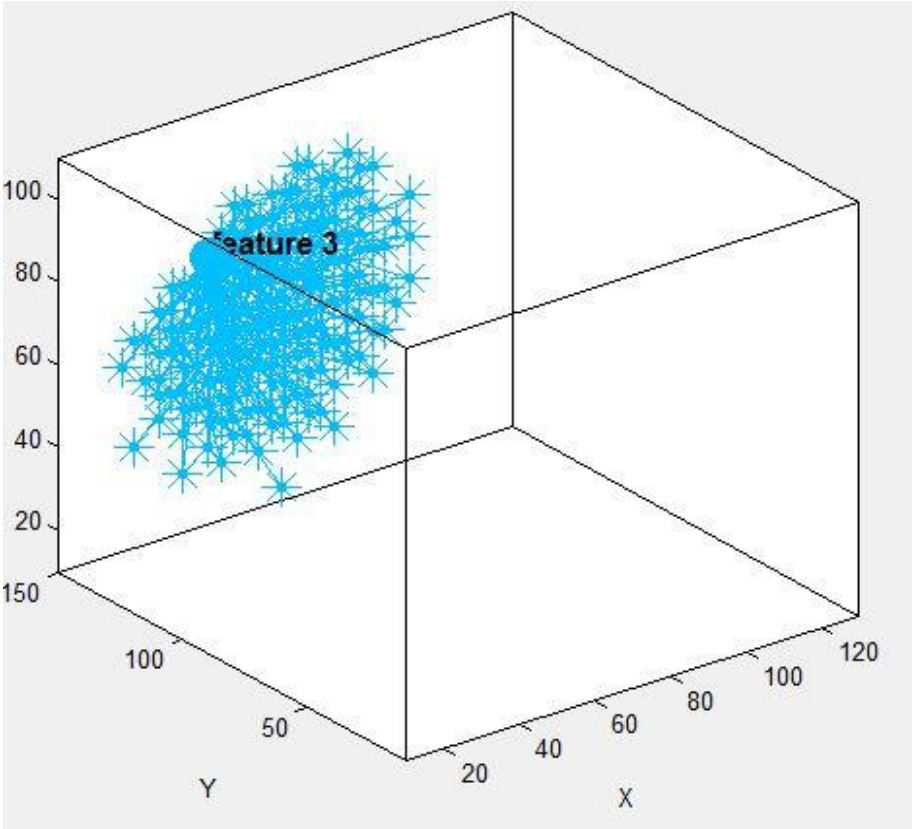

Figure 25. Information route analysis traced by 'spreading level' for input neuron number 3

- Information amount: This visualisation shows a tree rooted by the input neuron where a child neuron is chosen to be part of the tree only if it receives a minimum percentage of spikes from its parent neuron. The percentage is specified as decimal value (0.1 means a minimum of 10% spikes). Figure 26 shows a neuronal cluster from input neuron number 5, where every child neuron has received at least 10% of spikes from the parent neuron.

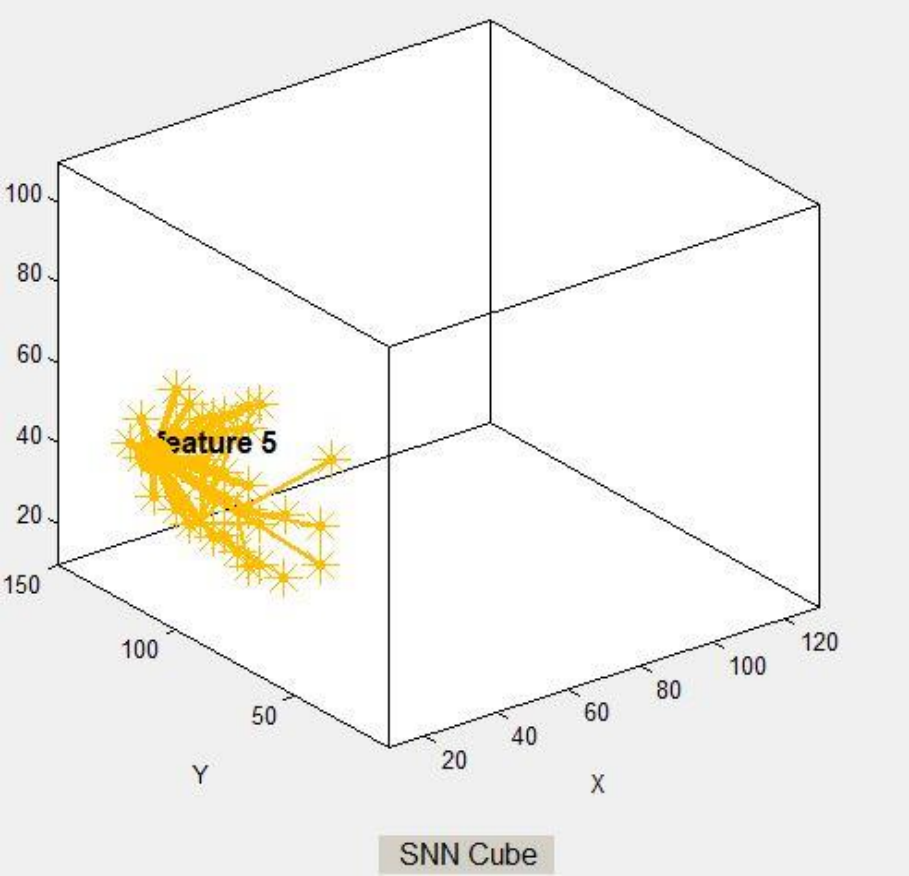

Figure 26. Information amount cluster for input neuron 5.

Training classifier

This step trains a model that takes the output spikes of the trained SNN cube as input and performs supervised learning to perform classification and regression. The UI for training classifier is started by clicking the 'Train Classifier' button. Figure 27 shows the train classifier panel in which the classification algorithm and corresponding hyper-parameters can be defined. For this demo the default parameter set is kept. Click 'OK' to start supervised learning. As for the unsupervised training of the cube, this step can be dynamically visualised (see page 17).

Figure 27. Classifier/regressor training panel

Verify classifier

This step is used to verify the accuracy of the model built by deSNN learning. Clicking on 'Verify Classifier' begins the verification procedure. At the end of verification the output result is visualised graphically as shown in Figure 28. The 2D plot shows the sample ID against the class label, and the legend describes the actual and predicted class labels. The 'result information' table lists these labels for each sample. Overall and class-wise accuracies are shown at the end. The true and predicted class labels can be exported to a csv file by clicking 'Export Results'.

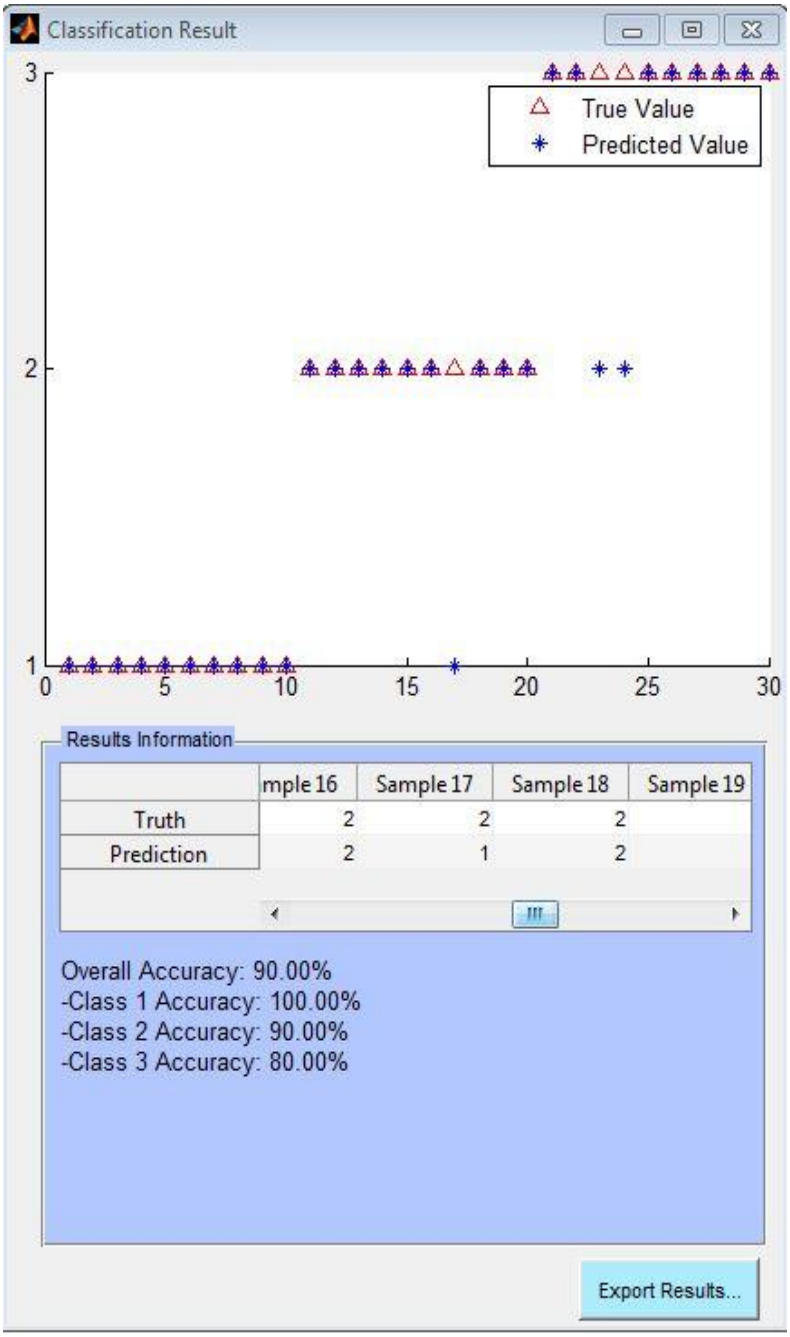

Figure 28. Output classification result panel for the verification of the classifier on the testing data.

Output layer visualisation

This option can be accessed through the menu item 'Classifier' as shown in Figure 29. It offers five methods for analysis as described below: The 'output layer visualisation' panel can be controlled

from the 'classifier' tab from the menu bar as shown in Figure 29. Each option under 'classifier' tab is described below:

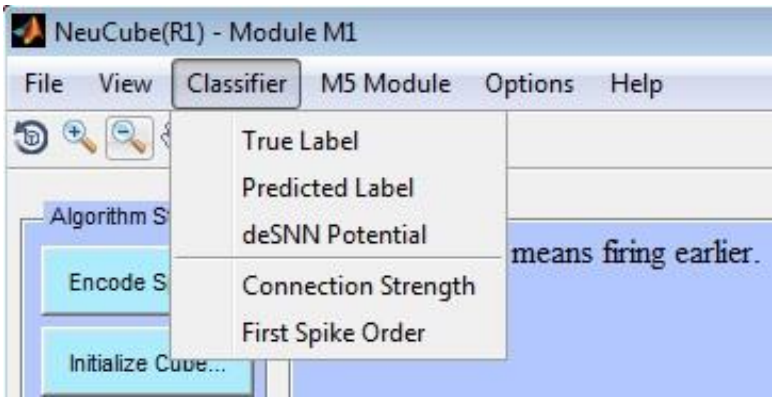

Figure 29. Output layer visualisation control

- *True label*: This option displays the true (actual) label of each sample by using a different colour for each class. The samples are ordered by their number (see description of Sample files) from bottom to top. Figure 30 shows the true labels for the demo dataset. The first sample corresponds to the first blue neuron at the bottom, and the last sample corresponds to the top red neuron. A red dot represents class 1, a green dot class 2, and a blue dot class 3.

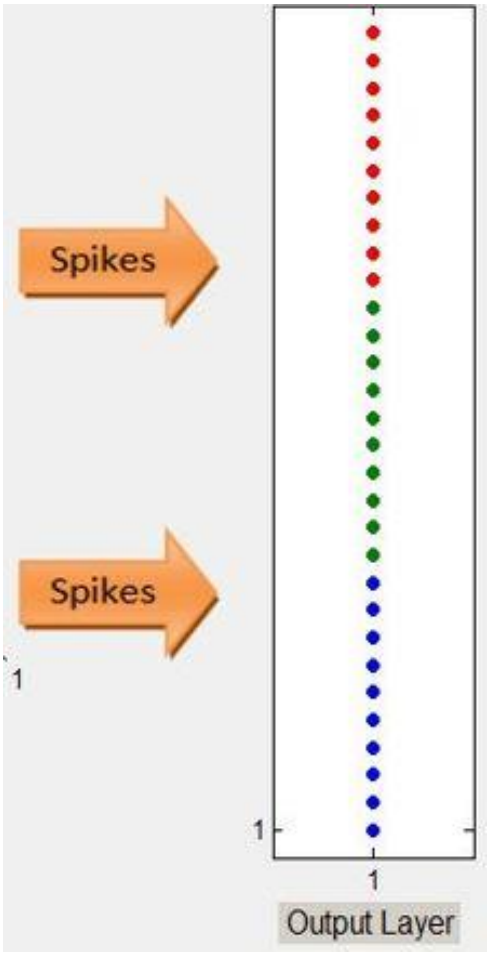

Figure 30. True labels shown in output layer

- *Predicted label*: This option displays the predicted label of each sample from the test/validation data set in the same way as for the true labels. Figure 31 shows the predicted labels for the demo dataset, where red is predicted class 1, green is predicted class 2, and blue is predicted class 3. In the demo we used 50% of the data (30 samples, 10 of each class) for training and 50% for testing (This was selected in the initialisation menu – Train Set Ratio parameter). In the example, two test samples that belong to class 1 (red) are wrongly classified into class 2 (green).

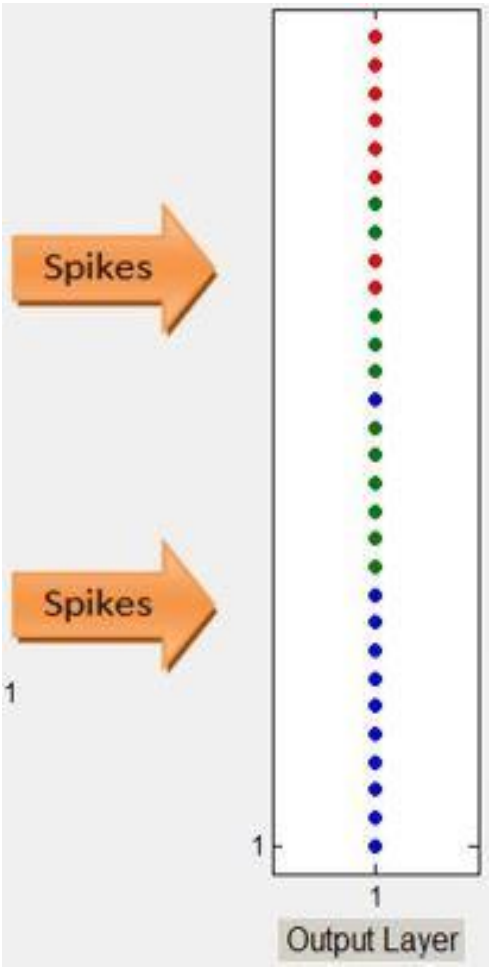

Figure 31. Classification output results after testing the trained classifier.

- *deSNN potential*: This displays the membrane potential of the output neuron per sample. As shown in Figure 32, a brighter neuron signifies higher potential.

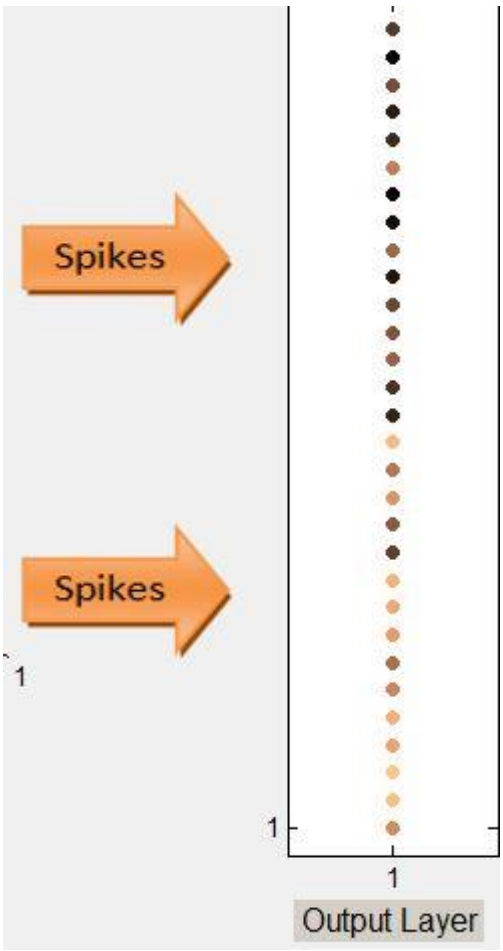

Figure 32. deSNN membrane potential of each output neuron in the output layer after supervised training

- *Connection strength*: This option enables the user to visualise the strength of connections between the SNNcube neurons for every output neuron (sample). As shown in Figure 33, clicking on one of the neurons in the output layer shows the connection strength of the neurons in the cube for that particular output neuron. Brighter neurons are more strongly connected than darker neurons.

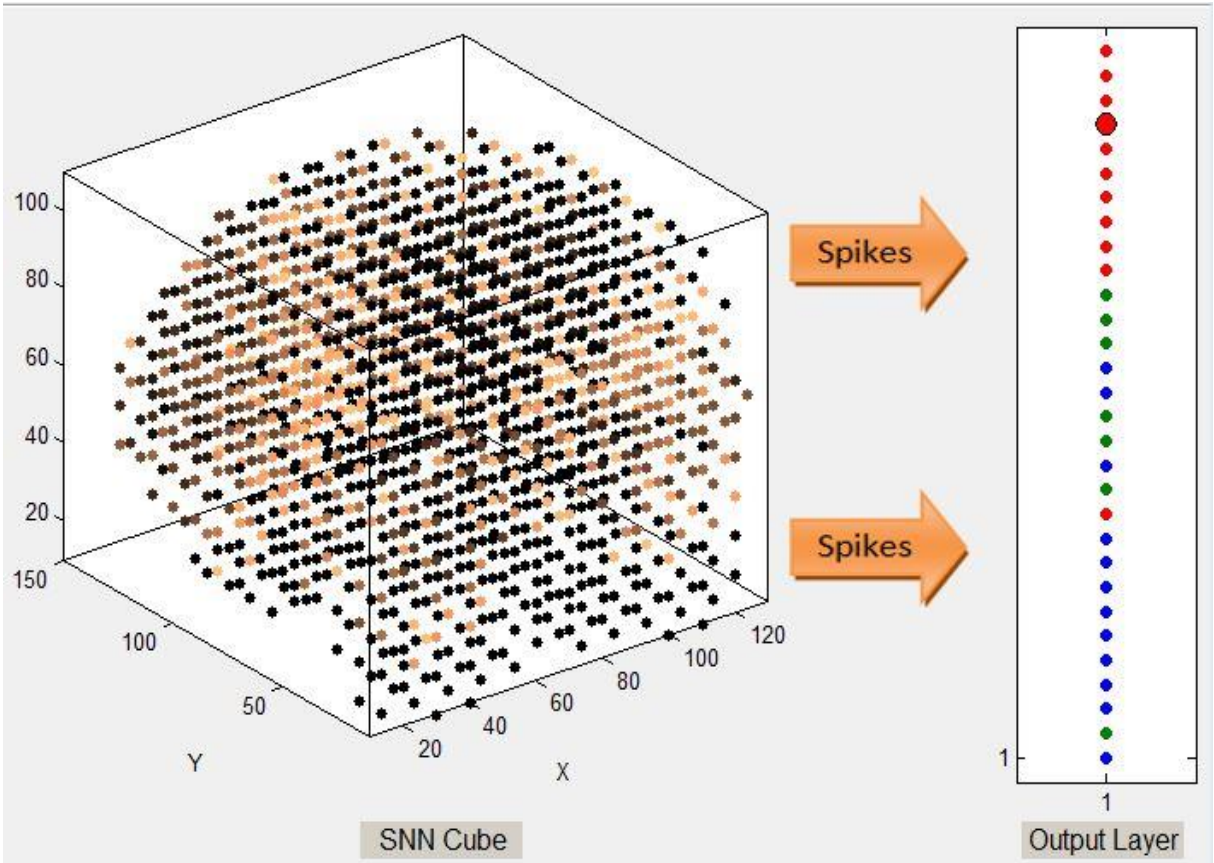

Figure 33: Connection strength for the fourth sample (highlighted by the big neuron in the output layer)

- *First spike order*: This option enables the user to visualise the spiking order of the neurons in the SNNcube for each output neuron (sample). As shown in Figure 34, clicking on one of the neurons in the output layer shows the firing order of the neurons in the cube for that particular output neuron. Brighter neurons fire earlier than darker neurons.

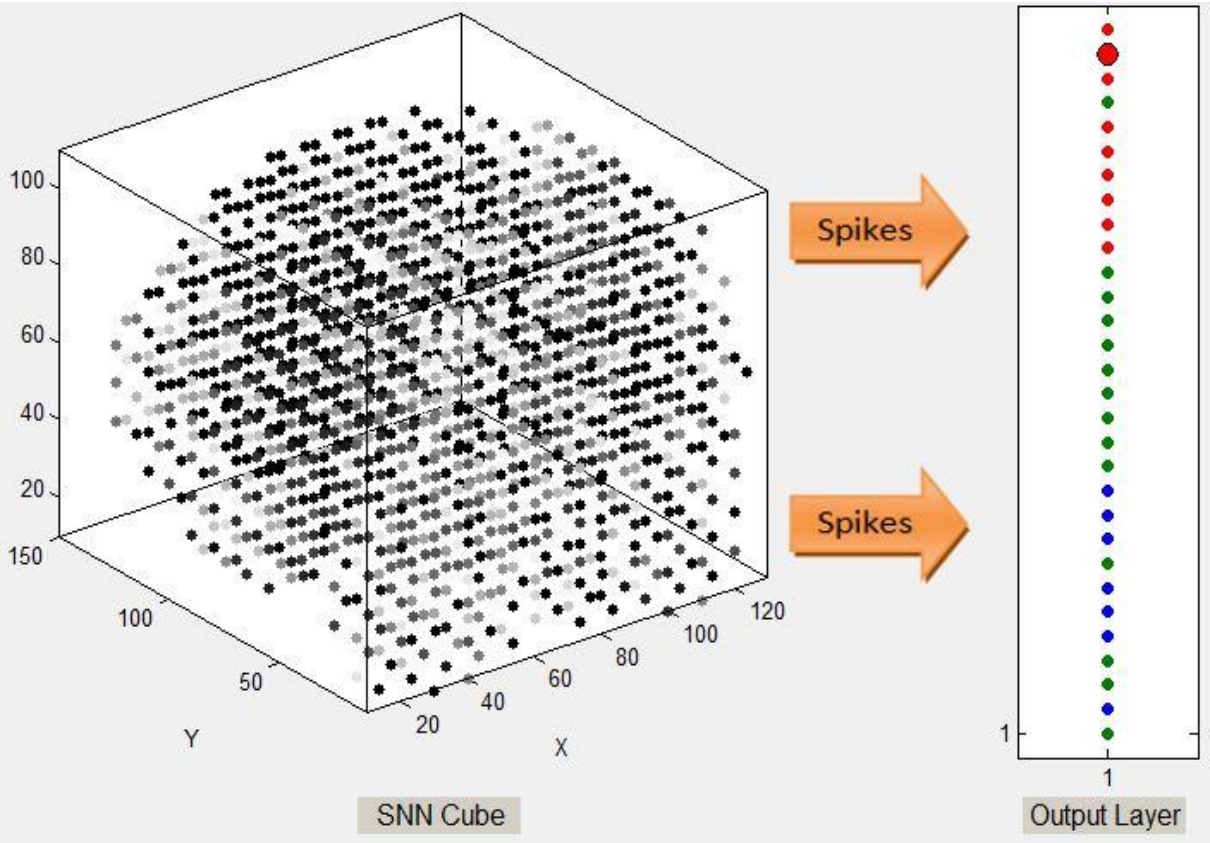

Figure 34. Firing order for the second sample (highlighted by the big neuron in the output layer)

Cross validation and parameter optimisation

Cross validation is a function that is wrapped around the unsupervised and supervised learning. At every fold the cube is initialised, trained unsupervised, and trained supervised with different combinations of data. Cross validation can be performed by clicking on the 'Cross Validation button' in the tools panel. This shows the popup window in Figure 35. The fold number parameter defines the number of iterations of training and validation cycles.

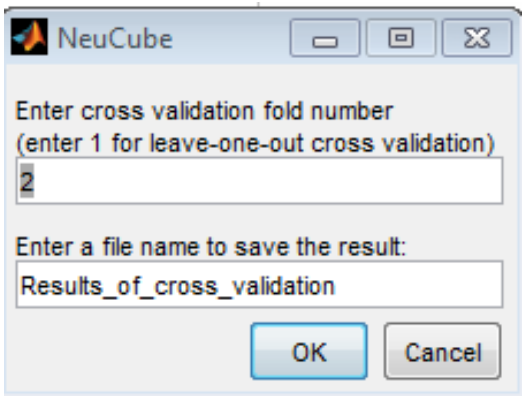

Figure 35. Cross validation UI

Parameter optimisation can be used to search for an optimal set of hyper-parameters that minimises the test accuracy of the model (either for classification or for regression). The computational time for parameter optimisation depends on the number of parameters to be

optimised and the size of the NeuCube model. Clicking on 'Param Optimisation' initialises the UI for parameter optimisation (Figure 36).

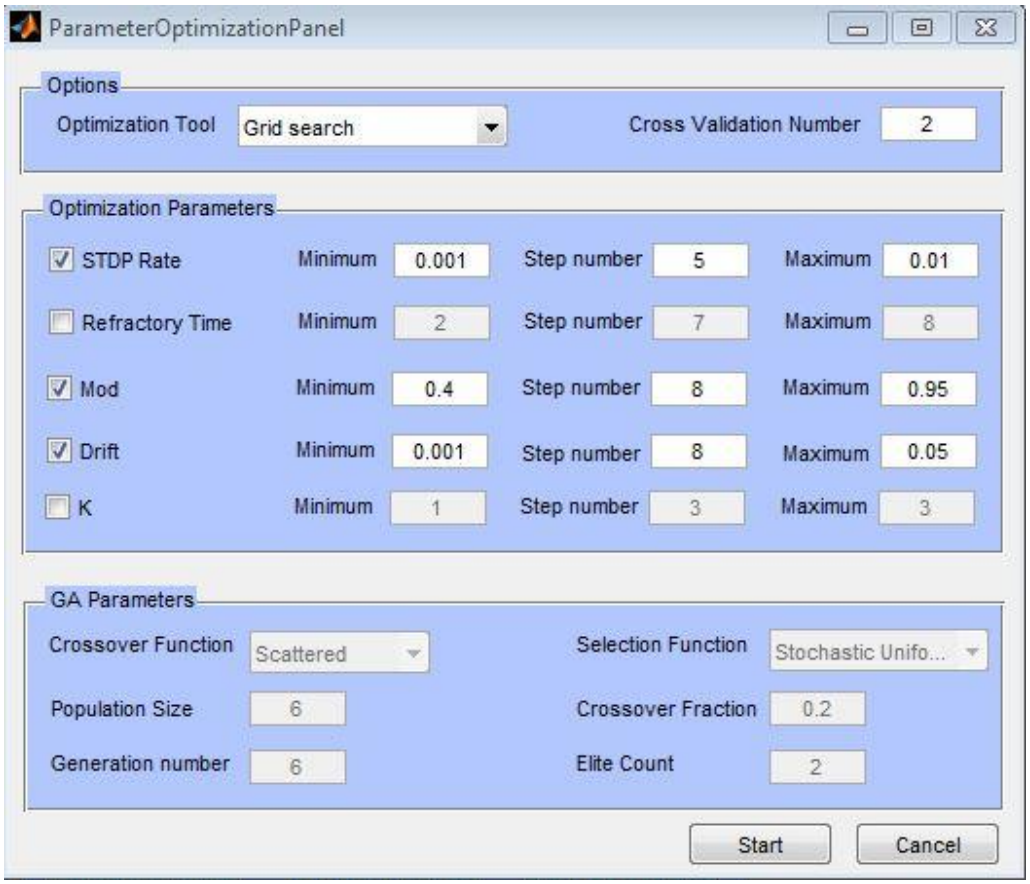

Figure 36. Parameter optimisation panel

Parameter optimisation in NeuCube M1 can be performed using various methods, such as grid search, Genetic Algorithms, Differential Evolution, Quantum Inspired Evolutionary Algorithms, PSO, etc. In NeuCube v1.3, two methods (grid search and Genetic Algorithms) are implemented and can be chosen from 'Optimisation Tool' dropdown menu. NeuCube v1.3 offers five parameters for optimisation; however the later release will include more parameters for optimisation.

1. **Exhaustive grid search**: This is an exhaustive search method using a grid-based combination of parameters. The 'Optimisation parameters' subpanel can be used to specify the parameters to be optimised by enabling the checkboxes. For example, in Figure 36 the three parameters STDP Rate, Mod, and Drift were chosen to be optimised. Each parameter is searched within a range, specified by the 'Minimum' and 'Maximum' values. The 'step number' specifies the number of steps to be used for moving from minimum to maximum. Once these parameters are set, clicking 'Start' begins the parameter optimisation by grid search.
2. **Genetic algorithm (GA)**: This is a nature inspired algorithm that employs the workings of genetic recombination in living beings as they happen in nature. Choosing this option enables the parameters for genetic algorithms at the bottom of the window. Similar to exhaustive grid search, the parameters can be chosen by checking the checkboxes, and bounds of the parameters can be set by using 'maximum' and 'minimum' values. Contrary to exhaustive search, though, GA is not a fixed-step approach and it does not require the

step number to be specified. The following parameters can be used to modify the behaviour of the GA:

- Crossover function: specifies how the genetic algorithm combines two individuals, or 'parents', to form a crossover 'child' for the next generation. An individual represents the set of parameters (called here genes) of a NeuCube model (called 'chromosome').
    - Scattered: creates a random binary vector and selects the parameters (genes) where the vector is a 1 from the first parent, and the genes where the vector is a 0 from the second parent, and combines the genes to form a child's chromosome.
    - Single point: chooses a random integer n between 1 and the number of variables and then
        - Selects vector entries numbered less than or equal to n from the first parent.
        - Selects vector entries numbered greater than n from the second parent.
        - Concatenates these entries to form a child's chromosome vector.
    - Double point: chooses a random integer n between 1 and the number of parameters and then:
        - Selects vector entries numbered less than or equal to n from the first parent.
        - Selects vector entries numbered greater than n from the second parent.
        - Concatenates these entries to form a child vector.
- Selection function: specifies how the genetic algorithm chooses parents for the next generation.
    - Stochastic uniform: lays out a line in which each parent corresponds to a section of the line of length proportional to its scaled value. The algorithm moves along the line in steps of equal size. At each step, the algorithm allocates a parent from the section it lands on. The first step is a uniform random number less than the step size.
    - Remainder: Remainder selection assigns parents deterministically from the integer part of each individual's scaled value and then uses roulette selection on the remaining fractional part.
    - Uniform: Uniform selection chooses parents using their evaluated fitness value (i.e., the classification accuracy of the NeuCube model). Uniform selection is useful for debugging and testing, but is not a very effective search strategy.
    - Roulette: Roulette selection chooses parents by simulating a roulette wheel in which the area of the section of the wheel corresponding to an individual is proportional to the individual's expectation. The algorithm uses a random number to select one of the sections with a probability equal to its area.
    - Tournament: Tournament selection chooses each parent by choosing Tournament size players at random and then choosing the best individual out of that set to be a parent.
- Population size: specifies how many individuals (NeuCube models) will be created and tested for fitness at each generation. With a large population size, the genetic algorithm searches the solution space more thoroughly, thereby reducing the chance that the algorithm returns a local minimum that is not a global

minimum. However, a large population size also causes the algorithm to run for longer.

- Generation number: specifies the maximum number of generations for the genetic algorithm to run.
- Crossover fraction: specifies the fraction of the next generation, other than elite children, that are produced by crossover.
- Elite count: specifies the number of individuals that are guaranteed to survive to the next generation. Set Elite count to be a positive integer less than or equal to the population size.

Figure 37 shows an example plot of fitness value (error) vs number of generations. It can be clearly seen how the model accuracy increases over time (generation).

Once the optimisation method and parameters are chosen, clicking 'Start' runs the parameter optimisation. During this process the running time and estimated time remaining for the optimisation can be visualised in information panel as shown in Figure 38. This is dynamically updated periodically. At the end of parameter optimisation, the best parameters (stored as a Matlab file) and the log file (stored as a text file) inside your current directory of execution (If you are executing NeuCube from desktop it will be saved on desktop). The matlab parameter file can be used and loaded into the NeuCub-M1 environment later. Optionally, you can also use the log file to capture the best parameters.

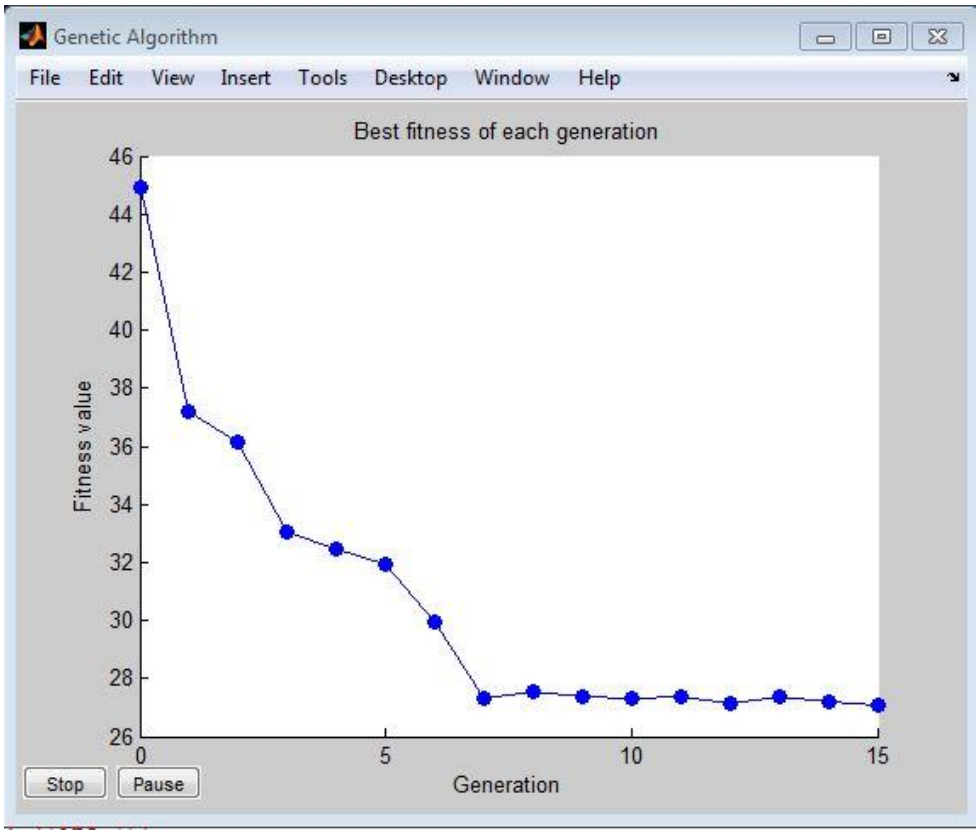

Figure 37. Example of parameter optimisation of a NeuCube model for classification using a GA optimiser. The classification error of the model (related to fitness) is decreasing with every generation of the GA where different parameter values are selected for the model.

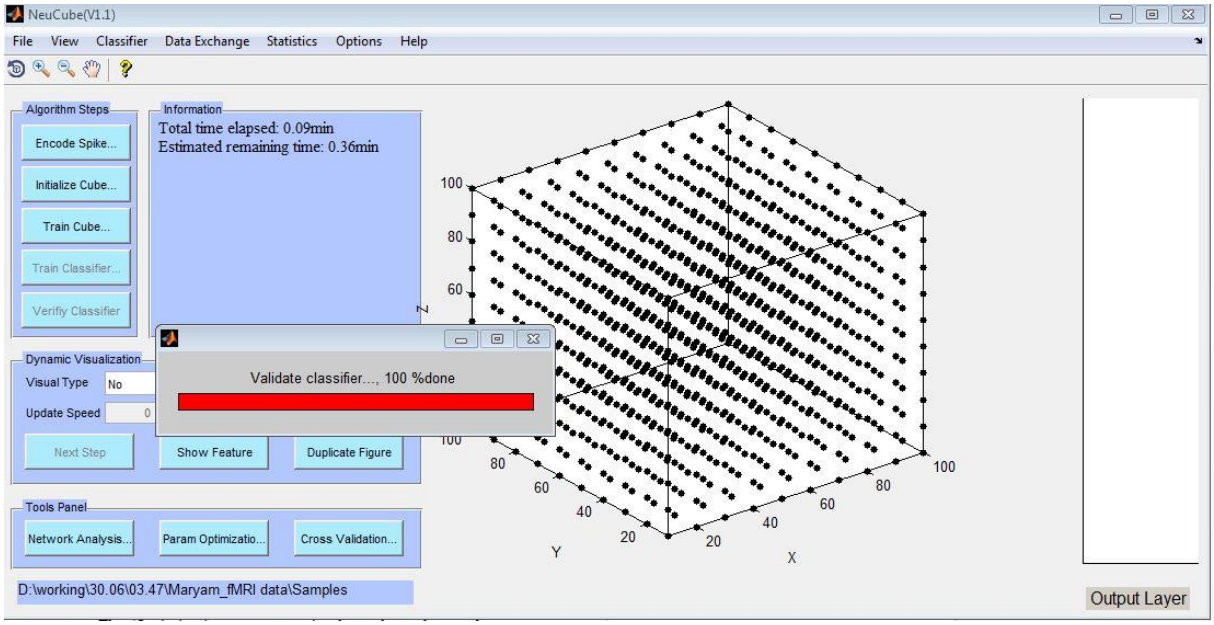

Figure 38: Example of parameter optimisation running in NeuCube-M1. The information panel shows the total time elapsed and estimated remaining time for the optimisation

Exporting statistics and results

It is possible to export some information as numbers for further analysis by clicking 'Statistics' in the menu bar as shown in Figure 39. It is possible to download SNNcube connection weights, output connection weights, SNNcube neuron activation levels, and spike emission statistics by clicking 'Cube Weight', 'Outlayer Weight', 'Activation Level' and 'Spike Emitted', respectively. The statistics are exported as csv files to a location specified by the user.

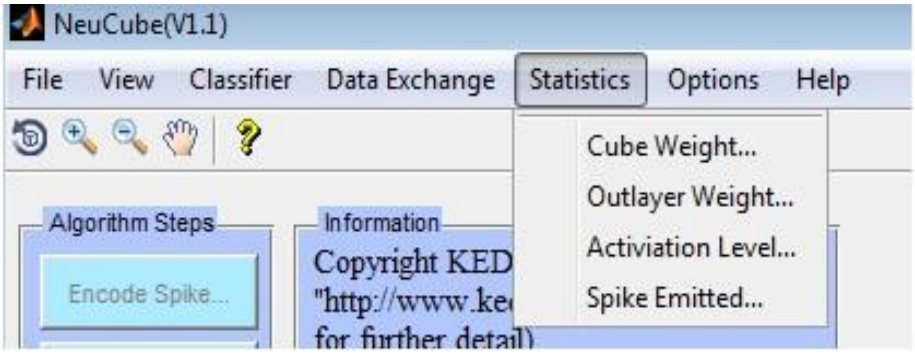

Figure 39: Statistics options

6. EXAMPLE OF REGRESSION ANALYSIS

A demo dataset for regression analysis can be found in the folder data → share_price. This dataset consists of 50 samples. Each sample consists of 100 timed sequences of daily closing prices for six different shares (Apple Inc., Google, Intel Corp, Microsoft, Yahoo, and NASDAQ). The target values representing the closing price of NASDAQ at the next day are arranged in a column in the target file. For dataset like this financial dataset that does not have any natural spatial ordering, NeuCube automatically assigns spatial location based on a graph matching algorithm. Hence, this dataset does not require any additional coordinate files. Data can be loaded from File → Load data → Regression by choosing the financial_dataset folder. Once the data is loaded, the steps of the algorithm can be performed as discussed in section 5. For initialisation of the SNNcube, choose the options 'Automatically' and 'Graph matching' from the dropdown menus 'Neuron Coordinates' and 'Given By', as shown in Figure 40.

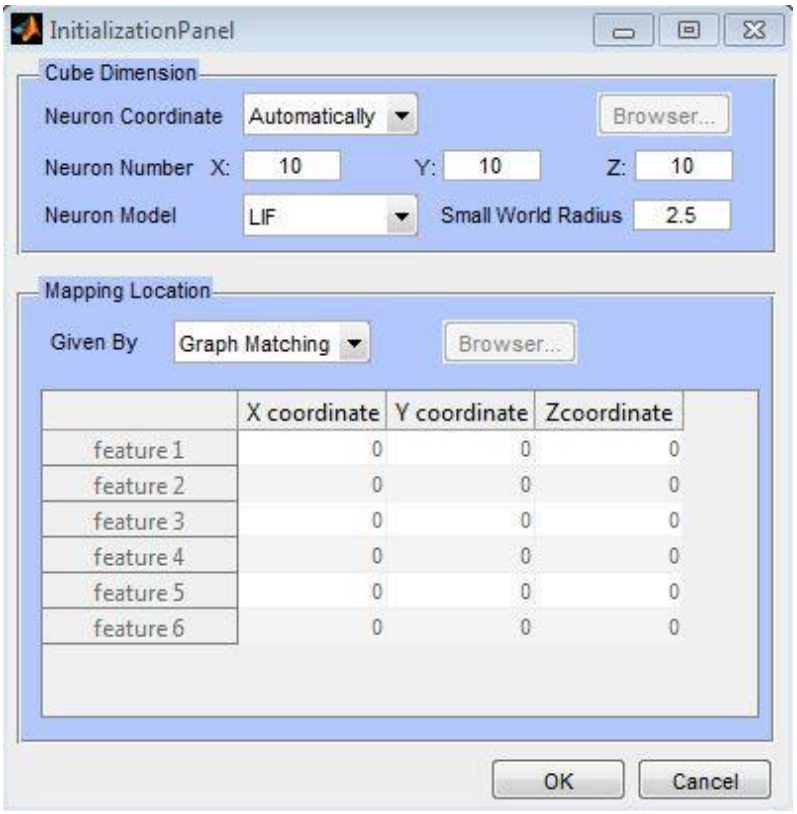

Figure 40: Initialisation options for regression DEMO

Figure 41 shows the regression result produced by NeuCube-M1 on the demo regression dataset. The graph plots the true and predicted value of the validation samples. It also provides the Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) as measures of performance on the validation set.

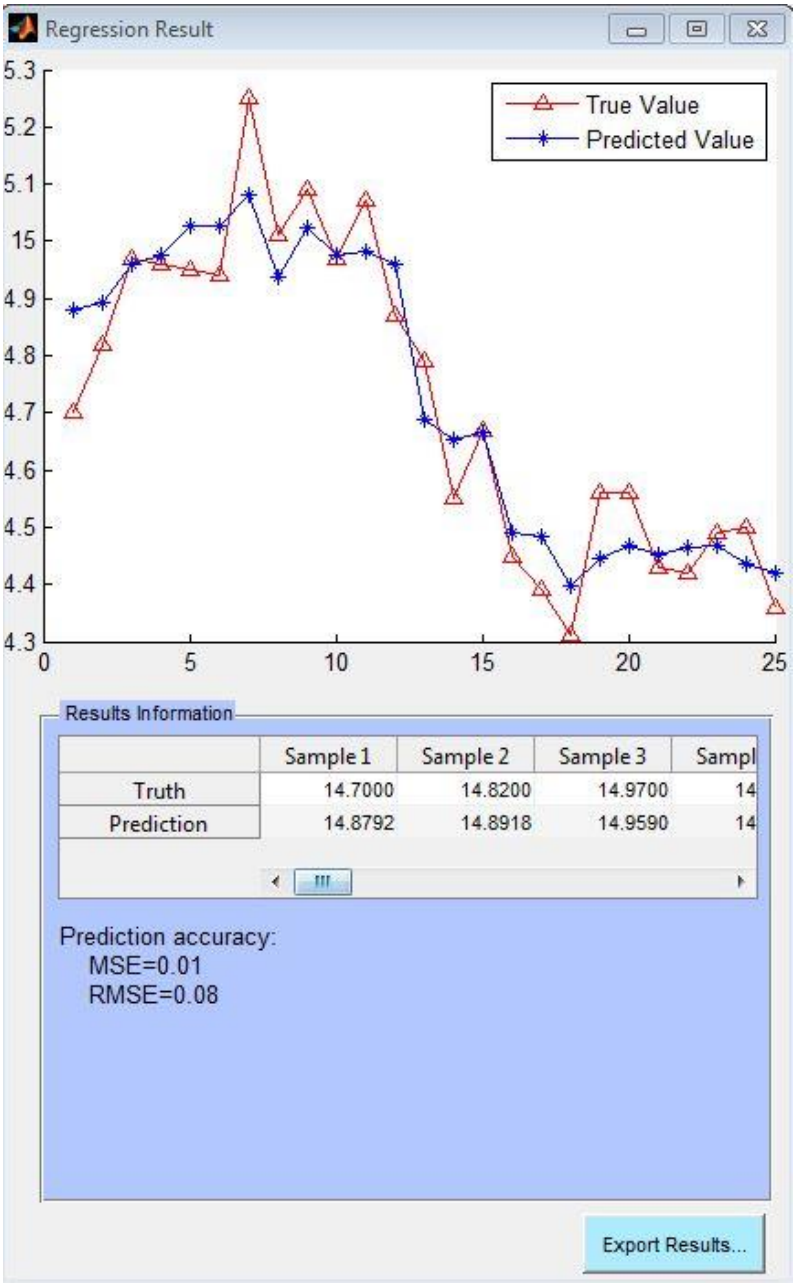

Figure 41: Regression result panel

7. RECALL

The NeuCube-M1 module can be used to perform recall on new samples. For the purpose of recall, the following resources are necessary:

1. Data Folder: A dataset for recall should only have a set of samples in one folder. If a target file is present in the folder, the software ignores it.
2. Cube file: Recall requires a cube file. The cube file contains the description of the NeuCube model. A cube file can be exported from the File menu → save NeuCube after the classifier has been trained.
3. Parameter file: Recall requires the parameter file used during training of the model. A parameter file can be exported from the File menu → save Parameter after the classifier has been trained.

A recall operation can be performed in following steps:

1. Load the recall data from the File menu → Load Data → Recall.
2. Once the dataset is successfully loaded, perform the spike encoding step by clicking 'Encode Spike'.
3. Load the cube and the parameter file saved previously from the File menu using the entries 'Load NeuCube' and 'Load Parameter', respectively.
4. Click on 'Verify Classifier' to predict the output on the recall samples.

8. REFERENCES

1. N. Kasabov et al, Design methodology and selected applications of evolving spatio- temporal data machines in the NeuCube neuromorphic framework, Neural Networks, 2015
2. Kasabov, N. Evolving connectionist systems for adaptive learning and knowledge discovery: Trends and Directions, Knowledge Based Systems, 2015, (2015), http://dx.doi.org/10.1016/j.knosys.2014.12.032.
3. Elisa Capecci, Grace Y. Wang , Nikola Kasabov, Analysis of connectivity in a NeuCube spiking neural network trained on EEG data for the understanding and prediction of functional changes in the brain: A case study on opiate dependence treatment, Neural Networks, (2015), http://dx.doi.org/10.1016/j.neunet.2015.03.009.
4. Maryam Gholami Doborjeh, Grace Y. Wang, Nikola Kasabov, A Neucube Spiking Neural Network Model for the Study of Dynamic Brain Activities during a GO/NO_GO Task: A Case Study on Using EEG Data of Healthy Vs Addiction Treated Subjects, IEEE Trans. NNLS, submitted, 2015.
5. Nikola Kasabov, Maryam Gholami Doborjeh, Spatio-Temporal Brain Data Mining with a NeuCube Evolving Spiking Neural Network Model on the fMRI Case study, IEEE Transactions of Neural Networks and Learning Systems, submitted 2015.
6. Enmei Tu, Nikola Kasabov, and Jie Yang, Mapping Temporal Variables into the NeuCube Spiking Neural Network Architecture for Improved Pattern Recognition, Predictive Modelling and Understanding of Stream Data, IEEE Transactions of Neural Networks and Learning Systems, submitted, 2014Kasabov, N., E.Capecci, Spiking neural network methodology for modelling, classification and understanding of EEG spatio-temporal data measuring cognitive processes, *Information Sciences*, 294, 565-575, 2015, DOI: 10.1016/j.ins.2014.06.028, 2014..
8. Kasabov, N. NeuCube: A Spiking Neural Network Architecture for Mapping, Learning and Understanding of Spatio-Temporal Brain Data, *Neural Networks* vol.52 (2014), pp. 62-76, http://dx.doi.org/10.1016/j.neunet.2014.01.006
9. Tu, E., Cao, L., Yang, J., & Kasabov, N. (2014). A novel graph-based k-means for nonlinear manifold clustering and representative selection. Neurocomputing. doi:10.1016/j.neucom.2014.05.067
10. Kasabov, N., Feigin, V., Hou, Z. -G., Chen, Y., Liang, L., Krishnamurthi, R., Parmar, P. (2014). Evolving spiking neural networks for personalised modelling, classification and prediction of spatio-temporal patterns with a case study on stroke. Neurocomputing, 134, 269-279. doi:10.1016/j.neucom.2013.09.049
11. N. Murli, N. Kasabov, and B. Handaga, Classification of fMRI Data in the NeuCube Evolving Spiking Neural Network Architecture, Proc. ICONIP 2014, Springer LNCS, 2014..
12. M. G. Doborjeh, E. Capecci and N. Kasabov, Classification and Segmentation of fMRI Spatio-Temporal Brain Data with a NeuCube Evolving Spiking Neural Network Model, Proc. SSCI, IEEE Press, 2014.
13. E. Tu, N. Kasabov, M.Othman, Y. Li, S.Worner, J.Yang and Z. Jia, NeuCube(ST) for Spatio-Temporal Data Predictive Modelling with a Case Study on Ecological Data, Proc. WCCI 2014, Beijing, 7-13 July 2014, IEEE Press.
14. D. Taylor, N.Scott, N. Kasabov, E.Capecci, E. Tu, N. Saywell, Y. Chen, J.Hu and Z.Hou, Feasibility of NeuCube SNN architecture for detecting motor execution and motor intention for use in BCI applications, Proc. WCCI 2014, Beijing, 7-13 July 2014, IEEE Press.
15. M. Othman, N.Kasabov, E.Tu, V. Feigin, R.Krishnamurthi, Z.Hou, Y. Chen and J.Hu, Improved Predictive Personalized Modelling with the use of Spiking Neural Network System and a Case Study on Stroke Occurrences Data, Proc. WCCI 2014, Beijing, 7-13 July 2014, IEEE Press.
16. Hu, J., Hou, Z., Chen, Y., Kasabov, N., & Scott, N. (2014). EEG-Based Classification of Upper-Limb ADL Using SNN for Active Robotic Rehabilitation. In 2014 5th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (pp. 409-414). Sao Paolo, Brazil: IEEE. doi:10.1109/BIOROB.2014.6913811
17. N. Kasabov, J.Hu, Y. Chen, N.Scott, and Y. Turkova, Spatio-temporal EEG data classification in the NeuCube 3D SNN Environment: Methodology and Examples, Proc. ICONIP 2013, Springer LNCS, vol.8228, pp.63-69.

18. Y.Chen, J.Hu, N.Kasabov, Z. Hou and L.Cheng, NeuroCubeRehab: A Pilot Study for EEG Classification in Rehabilitation Practice Based on Spiking Neural Networks, Proc. ICONIP 2013, Springer LNCS, vol.8228, pp.70-77.
19. N. Scott, N. Kasabov, and G.Indiveri, NeuCube Neuromorphic Framework for Spatio-Temporal Brain Data and Its Python Implementation, Proc. ICONIP 2013, Springer LNCS, vol.8228, pp.78-84..
20. N.Kasabov, V.Feigin, Z.Hou, Y.Chen, Improved method and system for predicting outcomes based on spatio/spectro-temporal data, PCT patent, WO2015/030606 A2, priority date: 26.08.2013.
21. N.Kasabov, Data Analysis and Predictive Systems and Related Methodologies – Personalised Trait Modelling System, PCT/NZ2009/000222, NZ Patent, USA Patent 13/088,306, Filed: April 15, 2011, Priority: Sept.2008.
22. Kasabov, N., & Hu, Y. (2010, December). Integrated optimisation method for personalised modelling and case studies for medical decision support. International Journal of Functional Informatics and Personalised Medicine, 3(3), 236-256. doi:10.1504/IJFIPM.2010.039123

9. DEVELOPER TEAM AND CONTACT PERSONS

Prof. Nik Kasabov, Director KEDRI – overall design of the NeuCube architecture.

Dr. Enmei Tu, Research Fellow – developer of the main NeuCube-M1 module.

Neelava Sengupta (nsengupt@aut.ac.nz) – contact person to report errors in the NeuCube software and the Manual and also developer of module M1, M8 and M9.

Nathan Scott – developer of modules M2 and M3 for neuromorphic implementation.

Dr. Stefan Marks – developer of module M4 for 3D dynamic visualisation in a VR scenario.

Israel Espinosa Ramos – developer of module M6 for neurogenetic modelling.

Elisa Capecci – Co-developer of module M6 for neurogenetic modelling.

Vivienne Breen – Co-developer of module M7 for personalised modelling.

Maryam Gholami Doborjeh (mgholami@aut.ac.nz) – contact person for tutoring on NeuCube and for EEG and fMRI data modelling.

Akshay Raj Gollahalli – Co-developer of Module M10.

Reggio Hartono – Co-developer of Module M10.

Joyce D'Mello (jdmello@aut.ac.nz) – KEDRI Admin Manager

Dr. Enrico Tronchin (etronchin@aut.ac.nz) – AUT Commercialisation Manager

Address for correspondence:

2 Wakefield Street, AUT Tower, 7th floor, KEDRI

Phone: +64 9 9219504;

Website: http://www.kedri.aut.ac.nz/neucube

10. ACKNOWLEDGEMENTS

The NeuCube development system is funded by the Auckland University of Technology SRIF fund and partially by the MBIE of New Zealand for strategic alliances with China. The intellectual property of NeuCube is owned by AUT. Other researchers who took part in the early development of NeuCube and its pilot applications are: Nelson Chen, James Hu, Professors Z. Hou, J. Yang, V. Feigin, D. Taylor, Dr. G. Wang, Anne Wendt, M. Othman, N. Murli, F. Alvi, M. Fanghella, W. Bhattacharjee, L. Zhou, J. Weclawski and C. McNabb.