# From von Neumann architecture and Atanasoffs ABC to Neuromorphic Computation and Kasabov's NeuCube: Principles and Implementations

Neelava Sengupta, Josafath Israel Espinosa Ramos, Enmei Tu, Stefan Marks, Nathan Scott, Jakub Weclawski, Akshay Raj Gollahalli, Maryam Gholami Doborjeh, Zohreh Gholami Doborjeh, Kaushalya Kumarasinghe, Vivienne Breen, and Anne Abbott

**Abstract** During the 1940's John Atanasoff with the help of one of his students Clifford E. Berry, at Iowa State College, created the ABC (Atanasoff-Berry Computer) that was the first electronic digital computer. The ABC computer was not a general-purpose one, but still, it was the first to implement three of the most important ideas used in computers nowadays: binary data representation; using electronics instead of mechanical switches and wheels; using a von Neumann architecture, where the memory and the computations are separated. A new computational paradigm, named as Neuromorphic, utilises the above two principles, but instead of the von Neumann principle, it integrates the memory and the computation in a single module a spiking neural network structure. This chapter first reviews the principles of the earlier published work by the team on neuromorphic computational architecture NeuCube. NeuCube is not a general purpose machine but is still the first neuromorphic spatio/spectro-temporal data machine for learning, pattern recognition and understanding of spatio/spectro-temporal data. The chapter further presents the software/hardware implementation of the NeuCube as a development system for efficient applications on temporal or spatio/spectro-temporal across domain areas, including: brain data (EEG, fMRI), brain computer interfaces, robot control, multisensory data modelling, seismic stream data modelling and earthquake prediction, financial time series forecasting, climate data modelling and personalised, on-line

N. Sengupta (e-mail: nsengupt@aut.ac.nz), J. I. E. Ramos, N. Scott, A. R. Gollahalli, M. G. Doborjeh, Z. G. Doborjeh, K. Kumarasinghe, V. Breen, A. Abbott
KEDRI, AUT, Auckland, New Zealand

S. Marks
Colab, AUT, Auckland, New Zealand

E. Tu
Rolls Royce@NTU-corporate Lab, NTU, Singapore

J. Weclawski
Warsaw University of Technology, Warsaw, Poland

risk of stroke prediction, and others. A limited version of the NeuCube software implementation is available from http://www.kedri.aut.ac.nz/neucube/

## 1 Introduction

The breakthrough work of Alan Turing in the 1940's, stating the possibility of using just 0's and 1's to simulate any process of formal reasoning [4] lead to massive development in the field of information theory and computer architecture. Simultaneously, significant progress was made by the neuroscientists in understanding the most efficient and intelligent machine known to man, the human brain. These parallel advancements in the middle of the last century had made man's imagination of creating 'intelligent' systems a possibility. These rational systems/agents were thought ideally to be able to perceive the external environment and take actions accordingly to maximise its goal, mimicking the human brain. The improvements in computer architecture with its advances in input/output, storage and processing power meant that the dream of artificial intelligence (AI) was now a reality and hugely reliant.

The field of AI and machine learning has grown strength to strength from the simple McColluch and Pitt's linear threshold based artificial neuron model [43] to the latest era of deep learning [37], which builds very complex models by performing a combination of linear and non-linear transformations. This is done using millions of neuron stacked in a layered fashion forming an interconnected mesh. The tremendous push of AI towards emulation of real intelligence has been sustained by the realisation of the Moore's law [54] which states that the processing power of the of central processing units (CPU) doubles in every couple of years. The scalable computer architecture proposed by John von Neumann in 1945 as part of the draft of EDVAC computer [53] had to play a substantial role in accomplishing the continuous miniaturisation of the CPU chips. In the more recent years, the CPU chip manufacturing companies have spent billions of dollars in CMOS technology to shrink the transistor size to a minuscule($\approx$ 14 nanometres) and thus keep Moore's law alive. It is evident that this is non-sustainable and as per well-supported predictions will reach its boundary in the next five years [61].

The saturation in the scalability of the von Neumann architecture led to new developments in computer and computing architectures. Neuromorphic computing coined by Carver Mead in the 1980s [44] and further developed recently is one of the paradigms of computing which has come into prominence. As the name 'neuromorphic' suggests, this paradigm of computing is inspired heavily by the human brain. Moreover, as the existence of AI is complimented by computing architectures and paradigms, having a real neuromorphic computer architecture oriented processing unit is a step towards the development of highly neuromorphic AI.

## *1.1 Neuromorphic computing beyond von Neumann architecture*

Throughout the continuous evolution of the traditional computers, von Neumann or the stored program architecture has continued to be the standard architecture for computers. It is a multi-modular design based on rigid physically separate functional units. It specifically consists of three different entities:

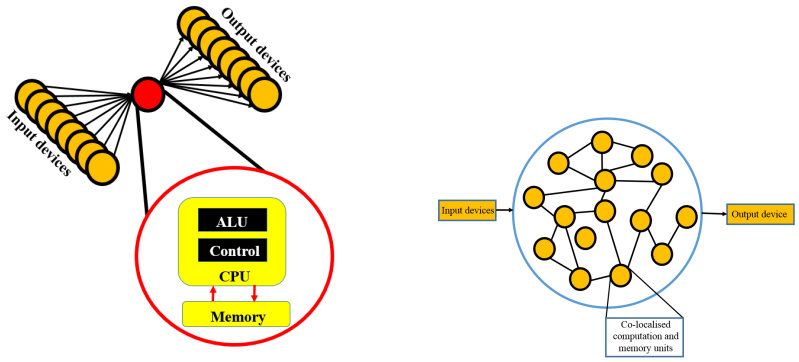
- Processing unit: The processing unit can be broken down into a couple of sub-units, the arithmetic and logical unit(ALU), the processing control unit and the program counter. The ALU compute the arithmetic logic needed to run programs. The control unit is used to control the flow of data through the processor.
- I/O unit: The i/o unit essentially encompasses all I/O the computer could possibly do (printing to a monitor, to paper, inputs from a mouse or keyboard, and others.).
- Storage unit: The storage unit stores anything the computer would need to store and retrieve. This includes both volatile and non-volatile memory.

Table 1: A comparison of the key contrasts between von Neumann and neuromorphic computing paradigm [34]

| | **von Neumann** | **Neuromorphic** |
|---|---|---|
| Representation of the data | Sequence of binary numbers | Spike(event) timings |
| Memory | 1. Volatile<br>2. Non-volatile | 1. Long term memory<br>2. Short term memory |
| Plasticity(Learning) | No | Adaptable via:<br>1. Long-term potentiation and depression<br>2. Short-term potentiation and depression |
| Processing | 1. Deterministic<br>2. Centralised<br>3. Sequential | 1. Stochastic<br>2. Decentralised<br>3. Parallel |

These units are connected over different buses like data bus, address bus and control bus. The bus allows for the communication between the various logical units. Though very robust, as shown in figure 1a, this architecture inherently suffers from the bottleneck created due to the constant shuffling of the data between the memory unit and the central processing unit. This bottleneck leads to rigidity in the architecture as the data needs to pass through the bottleneck in a sequential order. An alternate solution of parallelising the computers has been proposed where millions of processors are interconnected. This solution, though, increases processing power, is still limited by the bottleneck in its core elements [56].

The neuromorphic computing paradigm as shown graphically in figure 1b, on the contrary, draws great inspiration from our brain's ability to manage tens of billions of processing units connected by the hundreds of trillions of synapses using tens of watts of power on an average. The vast network of the processing units(neurons) in




(a) The von Neumann architecture used in traditional computers. The red lines depict the data communication bottleneck in the von Neumann architecture.

(b) A graphical representation of a general neuromorphic architecture. In this architecture, the processing and memory are decentralised across different neuronal units(the yellow nodes) and synapses(the black lines connecting the nodes), creating a naturally parallel computing environment via the mesh-like structure.

Fig. 1: *A graphical comparison of the von Neumann and Neuromorphic architecture. [34]*

the brain is in a true sense a mesh. The data is transmitted over the network via the mesh of synapses seamlessly. Architecturally the presence of the memory and the processing unit as a single abstraction is uniquely advantageous leading to dynamic, self-programmable behaviour in complex environments [56]. The highly stochastic nature of computation in our brain is a very significant divergence from the bit-precise processing of the traditional CPU. The neuromorphic computing hence aspires to move away from the bit-precise computing paradigm towards the probabilistic models of simple, reliable and power and data efficient computing [8] by implementing neuromorphic principles such as spiking, plasticity, dynamic learning and adaptability. This architecture morphs the biological neurons, where the memory and the processing units are present as part of the cell body leading to decentralised presence of memory and processing power over the network. Table 1 lists down some of the fundamental characteristics of the von Neumann and neuromorphic architecture.

With significant commercial interest in sight, research community focused on the commercial scale development of the neuromorphic chips. The most prominent of the neuromorphic chips include the Truenorth [26, 45] from IBM, the Neurogrid [3] developed by the Stanford University and SpiNNaker chip [20] from the University of Manchester, the neuromorphic chip developed in ETH INI, Zurich [27] and others. All of these neuromorphic chips consists of programmable neuron and synapses and uses a multitude of CMOS technologies to achieve the neuromorphic

behaviours. The details of the neuromorphic chips are beyond the scope of this article and are well elaborated in [29].

Numerous research [39, 40] has focused on harnessing the theoretical powers of the spiking neural network (SNN). While the majority of the research focus on neurological simulations, its importance in the real world of engineering applications that deal with complex processes (and thus generate the spatio/spectro-temporal data), is yet to be identified. The importance of a computational model to capture and learn spatio/spectro-temporal (SSTD) patterns from data streams is henceforth very significant from an application perspective. Example of problems involving SSTD are: brain cognitive state evaluation based on EEG [9], fMRI data [14], moving object recognition from video data [11], evaluation of the response of a disease on treatment and others. In this article, we discuss a new neuromorphic artificial intelligence paradigm that uses SNN and the first spatio-temporal data machine called NeuCube [33]. This framework has been recently further developed and implemented as an SNN development system for applications on SSTD described in this chapter.

The rest of the article is organised into eight sections. Section 2, briefly describes the existing work on the neuromorphic software implementations including a brief review of the existing neural network simulators. Section 3, concisely presents the architecture of the NeuCube development system as a next-generation pattern recognition system. Section 4 describes the generic SNN prototyping and testing module M1 of NeuCube developed in Matlab. Further down, the implementation of the Neuromorphic hardware and virtual reality environment is presented in sections 5 and 6. Section 7 elaborates on the recent development of the NeuCube software in the state of the art Java model view control(MVC) framework. Finally, Section 8 summarises and concludes the chapter.

## 2 Related work

The number of software implementations that has appeared, as a result of ongoing research in the area of artificial neural networks, is ever growing. Majority of the neural network software is implemented to serve two purposes:

- Data analysis: These software packages are aimed at analysing real-world data derived from practical applications. The data analysis softwares use a relatively simple static architecture, hence are easily configurable and easy to use. Few examples of such software are: multilayer perceptron(MLP) [2], RBF network [49], Probabilistic network (PNN) [60], Self organizing maps(SOM) [36], Evolving connectionist systems, such as DENFIS and EFuNN [30]. These softwares are either available as independent packages, such as NeuCom [1], PyBrain(python) [55], Fast Artificial Neural Network(C++) [48], or as part of a data analytics software like Weka [24], Knime [5], Orange [12] and others.
- Research and development systems: As opposed to the data analysis softwares, they are complex in behaviour, and require background knowledge for usage and

configuration. the Majority of the existing SNN softwares, including NeuCube, belong to this class.

We have briefly reviewed some of the key features of the current SNN development systems below.

NEURON [25]: Neuron is aimed at simulating a network of detailed neurological models. Its ability to simulate biophysical properties such as multiple channel types, channel distributions, ionic accumulation and so on renders it well suited for biological modelling [6]. It also supports parallel simulation environment through (1) distributing multiple simulations over multiple processors, and (2) distributing models of individual cells over multiple processors.

PyNEST [17]: The neural simulation tool (NEST) is a software primarily developed in C++ to simulate a heterogeneous network of spiking neurons. NEST is implemented to ideally model neurons in the order of $10^4$ and synapses in the order of $10^7$ to $10^9$ on a range of devices from single core architectures to supercomputers. NEST interfaces with python via implementation of PyNEST. PyNEST allows for greater flexibility in simulation setup, stimuli generation and simulation result analysis. A node and a connection comprise the core elements of the heterogeneous architecture. The flexibility to simulate a neuron, a device or a subnetwork (which can be arranged hierarchically) as a node, provides a major improvement over [51]. Due to the bottom-up approach of network simulation, the software allows for individually configurable neuron states and connection setup.

Circuit Simulator [47]: The circuit simulator is a software developed in C++ for simulation of heterogeneous networks with major emphasis on high-level network modelling and analysis, as opposed to [25]. The C++ core of the software is integrated with Matlab based GUI, for ease of use and analysis. CSIM enables the user to operate both spiking and analog neuron models along with mechanisms of spike and analog signal transmission through its synapse. It also performs dynamic synaptic behaviour by using short and long-term plasticity. In 2009, circuit simulator was further extended to parallel circuit simulator (PCSIM) software with the major extension being implementation on a distributed simulation engine in C++, interfacing with Python based GUI.

Neocortical Simulator [16]: NCS or Neocortical Simulator is an SNN simulation software, mainly intended for simulating mammalian neocortex [6]. During its initial development, NCS was a serial implementation in Matlab but later rewritten in C++ to integrate distributed modelling capability [63]. As reported in [6], NCS could simulate in the order of 106 single compartment neuron and $10^12$ synapses using STP, LTP and STDP dynamics. Due to the considerable setup overhead of the ASCII-based files used for the I/O, a Python-based GUI scripting tool called BRAINLAB [16] was later developed to process I/O specifications for large scale modelling.

Oger Toolbox [50]: Oger toolbox is a Python-based toolbox, which implements modular learning architecture on large datasets. Apart from traditional machine learning methods such as Principal Component Analysis and Independent Component Analysis, it also implements SNN based reservoir computing paradigm for learning from sequential data. This software uses a single neuron as its building

block, similar to the implementation in [17]. A Major highlight of this software includes the ability to customise the network with several non-linear functions and weight topologies, and a GPU optimised reservoir using CUDA.

BRIAN [23, 22]: Brian is an SNN simulator application programming interface written in Python. The purpose of developing this API is to provide users with the ability to write quick and easy simulation code [22], including custom neuron models and architecture. The model definition equations are separated from the implementation for better readability and reproducibility. The authors in [23], also emphasises the use of this software in teaching a neuroinformatics course [13]. A major limitation of BRIAN is, however, the requirement of Python knowledge to run the simulation and the lack of GUI for the non-technical user community.

The aforementioned discussion of the existing software highlights the suitability for building highly accurate neurological models but lacks a general framework for modelling temporal or SSTD, such as brain data, ecological and environmental data. Further in the line of the neural network development systems, and more specifically for SNN, where not only an SNN simulator can be developed, but a whole prototype system (also called spatio-temporal data machine) can be generated for solving a complex problem defined by SSTD, the NeuCube framework was proposed [30].

## 3 The NeuCube framework and system architecture

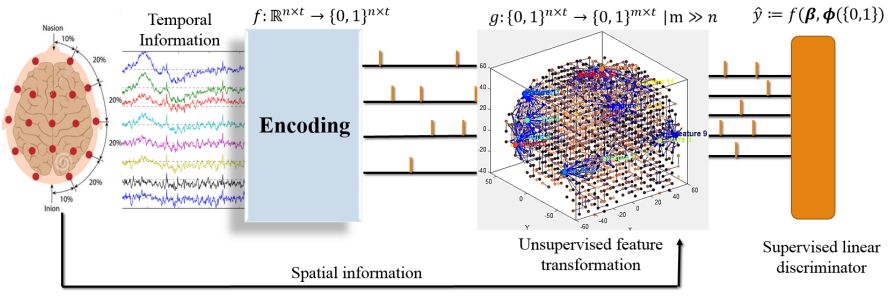



Fig. 2: *The NeuCube framework for SSTD. The brain, shown as a source of SSTD is only exemplary, rather than restrictive.*

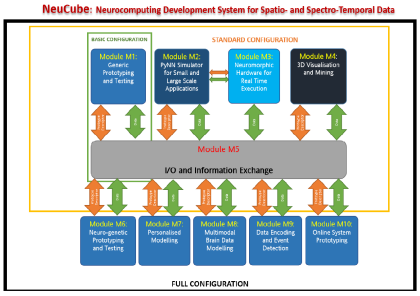
The NeuCube framework for SSTD, illustrated, but not restricted to brain data, is depicted in figure 2 [35] and explained below.

- Data encoding: The temporal information generated from the source(e.g. brain, earthquake sites) is passed through a data encoder component using a suitable encoding method, such as BSA [30], Temporal contrast, GAGamma[58]

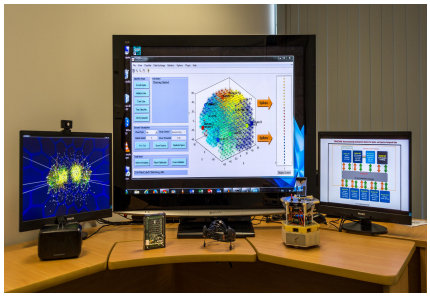
and others. It transforms the continuous information stream to discrete spike trains($f : \mathbb{R}^{n \times t} \rightarrow \{0,1\}^{n \times t}$).

- Mapping spike encoded data and unsupervised learning: The spike trains are then entered into a scalable three dimensional space of hundreds, thousands or millions of spiking neurons, called SNNcube (SNNc), so that the spatial coordinates of the input variables (e.g. EEG channels; seismic sites, and so on) are mapped into spatially allocated neurons in the Cube, and an unsupervised time-dependent learning rule [59][21] is applied ($g : \{0,1\}^{n \times t} \rightarrow 0,1^{m \times t} | m >> n$).
- Supervised learning: After unsupervised learning is applied, the second phase of learning is performed, when the input data is propagated again, now through the trained SNNc, and an SNN output classifier/regressor is trained in a supervised mode($\hat{y} := h(\beta, \phi(0,1))$)[31]. For this purpose, various SNN classifiers, regressors or spike pattern associators can be used, such as deSNN [31] and SPAN [46].

The NeuCube software development system architecture uses the above mentioned core pattern recognition block described in figure 2 as the central component and wraps a set of pluggable modules around it. The pluggable modules are mainly developed for: (1) Using fast and scalable hardware components running large scale applications; (2) Immersive model visualisation for in-depth understanding and analysis of the SSTD and its SNN model; (3) Specific applications like personalised modelling, brain computer interfaces and so on (4) Hyperparameter optimisation; and others.




(a) The NeuCube software development architecture for SNN applications on spatio/spectrotemporal data.



(b) NeuCube Modules M1, M2, M3 and M4 integrated in the KEDRI NeuLab, also showing some application oriented devices, such as Oculus for 3D visualisation, a SpiNNaker small neuromorphic board, Emotiv EEG device, an EEG-controlled mobile robot with Kyushu Institute of Technology

Fig. 3: *The NeuCube SNN development system for SSTD.*

Figure 3a shows the graphical representation of the NeuCube SNN development system for SSTD and fig. 3b shows the standard configuration in a real life setup

in the KEDRI NeuLab. Each module in figure 3a is designed to perform an independent task and in some instances, written in a different language and suited to the specific computer platform. All of the modules, however, are integrated via a common communication protocol in module M5. A brief description of the standard modules from figure 3a is given below:

**Module M1** It is a generic prototyping and testing module written in Matlab, where an SNN application system can be developed for data mining, pattern recognition and event prediction from temporal or SSTD. Additional functionalities like dynamic visualisation, network analysis toolbox, parameter optimisation are included in this module. Section 4 describes Module M1 in a more elaborate fashion.

**Module M2** is a python based simulator of NeuCube for large scale applications or implementation on a neuromorphic hardware (Module M3). This application is developed on top of PyNN package, which is a Python-based simulator-independent language for building SNN. The NeuCube-PyNN [57] module is not only compatible with existing SNN simulators described previously (e.g. Neuron, Brian), but can also be ported to a large neuromorphic hardware such as the SpiNNaker, or on any neuromorphic chip, such as the ETH INI chip, the Zhejiang University chip, and others. The advantage of such hardware lies in the extreme energy efficiency of computation, allowing for the large-scale massively parallel neuromorphic system to run more efficiently. **Module M3** is dedicated for such hardware implementations of NeuCube. Section 5 has a detailed description about Modules M2 and M3.

**Module M4** allows for a dynamic visualisation of the 3D structure and connectivity of the NeuCube SNN [41, 42]. Due to the 3-dimensional structure as well as the large number of neurons and connections within NeuCube a simple 2D connectivity/weight matrix or an orthographic 45-degree view of the volume is insufficient. We created a specialised visualisation engine using JOGL (Java Bindings for OpenGL) and GLSL (OpenGL Shading Language). This engine can render the structural connectivity as well as the dynamic spiking activity. Using 3D stereoscopic head-mounted displays such as the Oculus Rift, the perception and understanding of the spatial structure can be improved even further. Section 6 describes module M4 in detail.

**Module M5** is the input/output and the information exchange module. This module is responsible for binding all the NeuCube modules together irrespective of the programming language or platform. Experiments that are run on any module produces prototype descriptors containing all the relevant information, which are exported and imported as structured text files, and is compatible with all the modules. We have used language independent JSON (Javascript object notation) format as a structured text, which is lightweight, human readable and can be parsed easily. The present implementation of the I/O module supports the use of three types of data and SNN prototype descriptors. They are: (1) Dataset descriptor, which consist of all the information relevant to the raw and encoded dataset; (2) Parameter descriptor, which is responsible for storing all the user defined and changeable parameters of the software; and, (3) SNN application system descriptor, which stores information related to the NeuCube SNN application system.

**Module M6** extends the functionality of module M1, by adding functions for prototyping and testing of neurogenetic data modelling. These functions include models for genetic and proteomic influences in conjunction with brain data. This is provided as an optional feature for specific applications. This module is written in Java and can be further developed as an open source NeuCube SNN development system.

**Module M7** facilitates the creation and the testing of a personalised SNN system. It extends module M1 by including additional functionalities for personalised modelling which is based on first clustering of integrated static-dynamic data using new algorithm dWWKNN (dynamic weightedweighted distance K-nearest neighbours) and then learning from the most informative subset of dynamic data for the best possible prediction of output for an individual. This module is for optional use in the context of specific applications[32] (USA patent 2008). This module is for optional use in the context of specific applications [15].

**Module M8** is currently being developed as a plugin for multimodal brain data analysis. It aims to integrate different modalities of brain activity information (.g, EEG, fMRI, MEG) and structural Tractography (DTI) information, in NeuCube, for the purpose of better modelling and learning. This module is also bound to specific applications.

**Module M9** is an optional module for data encoding optimisation and event detection. This module includes several data encoding algorithms for mapping analog signal to spike trains based on different data sources[58].

**Module M10** provides an additional feature of online learning for real-time data analysis and prediction. In this module, continuous data streams are processed in the form of continuous data blocks.

## 4 NeuCube implementations for prototyping and testing

The generic prototyping and testing tools or Module M1 are the GUI based implementations for the rapid development of SNN application prototype systems for temporal or spatio/spectro-temporal data. It can also be used for research on SNN for general purpose pattern recognition from SSTD. Currently, there are two parallel implementations of the NeuCube system in Matlab and Java. This section describes the Matlab system in detail.

The Matlab implementation is designed as a set of continuous signal processing steps as shown in figure 2. The user interface of NeuCube M1, as shown in figure 4, is built as a logical stepwise process.

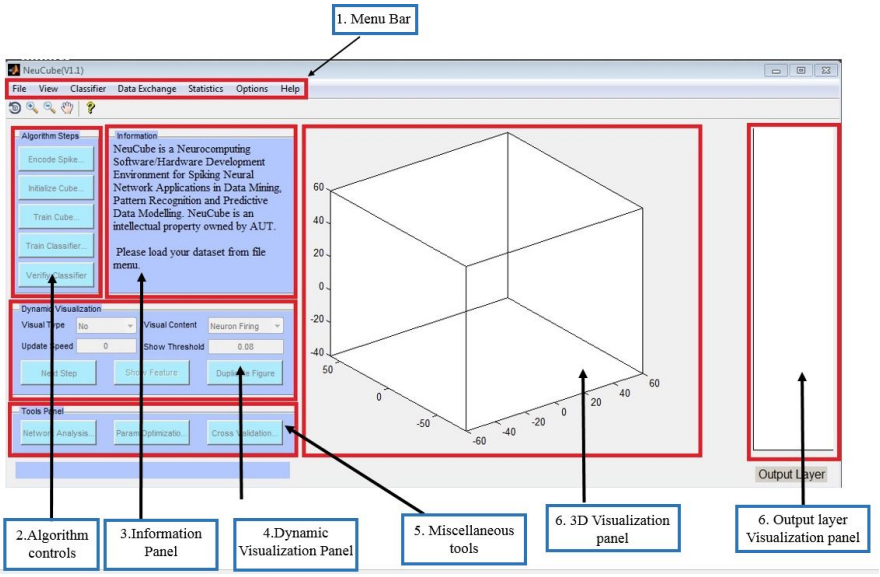



Fig. 4: *NeuCube-M1 user interface and panel descriptions*

## *4.1 Data exchange*

The data exchange component is used to import or export user defined information to and from the software, which includes temporal or SSTD data, already developed SNN systems, parameters and results. The NeuCube-M1 interacts with the external environment using four data descriptors. They are the following:

- Dataset descriptor: The Dataset descriptor consists of the data (and the metadata), that is to be learned and analysed. In the majority of the cases, a dataset contains 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.
- SNNc descriptor: The SNNc descriptor contains all information related to the structure and learning of the SNN. Some of the most important information stored in this descriptor are the spatial information of the input and reservoir neurons,structural information of the SNNc and the state of the SNNc during learning.
- Parameter descriptor: The parameter descriptor stores all the user defined parameters including hyperparameters of data encoding algorithms, the unsupervised learning algorithm and the supervised learning algorithm.
- Result descriptor: Result descriptor stores information about the experimental results produced by NeuCube.

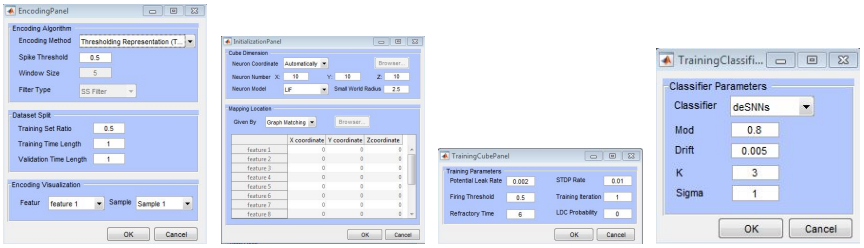
| Descriptor type | Mat | JSON | CSV |
|---|---|---|---|
| Dataset | yes | yes | yes |
| Cube | yes | yes | no |
| Parameter | yes | yes | no |
| Result | yes | no | yes |

Table 2: Supported file format for descriptors

NeuCube-M1 supports three different file formats, Mat (binary), JSON (structured text) and CSV (comma separated plain text). Table 2 describes the supported file formats for each of the descriptor type. As a heuristic rule, mat format is recommended for achieving fast I/O. The CSV files are the recommended choice for import/export of dataset and results for later analysis. The JSON format is recommended for inter-modular communication. Loading a dataset (temporal information) is the entry point to the software, and can be done by loading a csv or a mat file from the file menu. The import and export of all the descriptors can be performed throughout the lifetime of the experiment.

## *4.2 Algorithm interactions*

NeuCube-M1, being a general purpose pattern recognition software, allow users to interact with the pattern recognition and signal processing algorithms via the algorithm controls panel, shown in figure 4. The algorithm control panel includes a set of buttons for configuring and running the step by step process of data encoding, network initialisation, unsupervised learning(by training the SNNc) and supervised learning. The software uses a guided approach for performing the algorithmic steps by enabling or disabling buttons after every operation. Figure 5a, 5b, 5c and 5d shows the individual user interaction panels for encoding, initialisation, unsupervised and supervised learning respectively. Each panel allows users to choose from a set of algorithms and corresponding hyperparameters. For example, the data encoding panel(5a), that encode the real-valued signal to spike trains, provide the option of choosing from a set of encoding algorithms and its hyperparameters from the drop down menu. The initialisation panel, as shown in figure 5b, initiates the SNNc. NeuCube allows making use of the natural spatial ordering (if any) of the features of the data as input to the system. This can be achieved by loading an external csv coordinate file from mapping location option in the encoding panel. If a natural spatial ordering is not present in the data, NeuCube uses the graph matching technique, to enforce a spatial arrangement of the input variables based on their temporal similarities. All the panels for algorithmic control interacts with the visualisation panels, for visualisation and analysis of the data and the models.




(a) data encoding panel (b) SNNc initialisation panel (c) Unsupervised learning panel (d) Supervised learning panel

Fig. 5: *Algorithmic controls through the NeuCube-M1 UI's*

## *4.3 Integrated visualisation and network analysis*

visualisation and model analysis is an integral and unique feature of the NeuCube architecture. As discussed in [35], a NeuCube model acts as a white box, *i.e.*, a learned NeuCube model outputs analysable and interpretable spatio-temporal patterns for knowledge discovery. The visualisations in the M1 module are rendered in the '3D visualisation panel' and 'output layer visualisation panel' and can be manoeuvred by using the controls under the 'dynamic visualisation panel,'miscellaneous tools' and 'menu bar' in the user interface of Module M1.

Visualisation capabilities of NeuCube M1 module includes: comparative display of real and encoded data; online dynamic visualisation of unsupervised learning; static visualisation of the SNNc model and the output readout layer and visualisation; and SNNc network analysis.

### 4.3.1 Visualisation of data encoding:

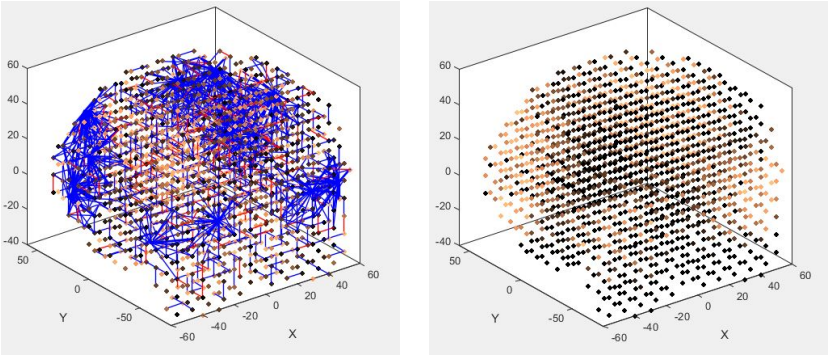
The Current version of the M1 module includes an offline encoding and visualisation scheme, but real-time online encoding and visualisation for streamed data is part of another specialised module - Module M10.

### 4.3.2 Visualisation of 3D SNNc model:

The SNNc in the NeuCube architecture learns the spatio-temporal patterns over time using the spikes transmitted by the spiking neurons. The SNNc also forms relationships by regulating the connection strengths between the neurons. The visualisation and analysis of this learning process are of utmost importance for knowledge discovery. The unsupervised learning process can be visualised dynamically online, while the system is learning, or can be saved to a movie for later usage and analysis by using dynamic visualisation panel. The plots can be rendered in a continuous, or

stepwise fashion. It is also possible to specify the type of activity to be rendered on the go, such as the spiking behaviour, evolution of connection, and others.

The static visualisation of the SNNc represents a snapshot of the final state of the cube at the end of the learning period. The option for static visualisation of the SNNc can be found in the view drop down under the menu bar. Figure 6a shows an example of the static visualisation of the spatial relationships formed via the connections below a defined threshold. The colour of the neurons in figure 6b, on the other hand, represent the spike emission count.




(a) Visualisation of relationships formed in the brain shaped SNNc at the end of unsupervised learning. The blue lines represent positive connections, and the red lines represent negative connections

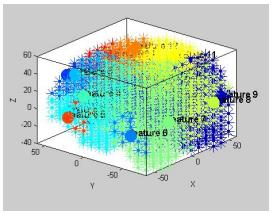
(b) Visualisation of spike emission density. The brighter neurons represent higher spike count

Fig. 6: *Static visualisation of SNNc model*

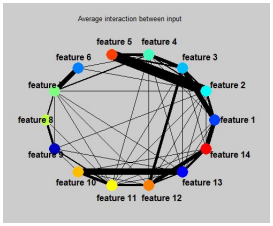
#### 4.3.3 SNNc (network) analysis:

The network analysis panel can be used for analysis of the SNNc network content. The network analysis consists of two major functionalities: (1) Neuron cluster analysis; (2) Information route analysis. An example of neuron cluster analysis visualisation is shown and described in figure 8a, 8b and 8c. Information route analysis is used for analysing information propagation route of the spikes generated by the spiking neurons. This analysis is based on the concept of the rooted tree. The type of information can be chosen by selecting the trace with drop down. Different methods of analysis is described below in brief:

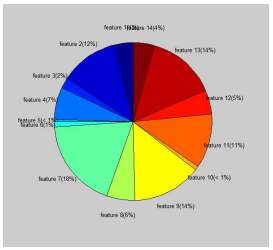
- Max spike gradient: Shows a tree rooted by the input neuron, where a child neuron is chosen to be connected to a parent neuron, if it receives spike from its parents.



(a) Example of neuron clustering based on connection weights of the network




(b) average one to one interaction between the input neuron clusters shown in figure 7a. The thicker lines signify more interaction.



(c) Pie chart depicting the number of neurons belonging to an input cluster group

Fig. 7: *Neuron cluster analysis by network analysis toolbox*

- Spreading level: 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.
- Information amount: 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 defined percentage of spikes from its parent neuron. The percentage can be specified by the information box, where 0.1 means 10% spikes.

#### 4.3.4 Output layer visualisation:

The output layer visualisation is concerned with the K-nearest neighbour based deSNN discriminator used as a supervised learning algorithm in the M1 module. The output layer renders a set of spiking neurons, where each output neuron represents the class label (or the regression value) assigned to one input SSTD sample.

### *4.4 Parameter optimisation*

Parameter optimisation is developed to allow users to search for the optimal set of hyperparameters that minimises the test accuracy of the NeuCube prototype system (model), either for classification or regression. The computational time for parameter optimisation depends on the number of parameters to be optimised and the size of the NeuCube model. Parameter optimisation in NeuCube Module M1 can be performed using various methods, such as: Grid search; Genetic Algorithm; Differential Evolution; Quantum-Inspired Evolutionary Algorithms, PSO, and so on.

The current release of NeuCube M1 includes two methods (Grid search and Genetic algorithm).

## 5 Implementations on hardware and neuromorphic chip

The NeuCube is also appropriate for implementation on dedicated neuromorphic hardware systems, or distributed computing platforms. Due to the fact that the model is by nature highly scalable, it requires a highly scalable computation platform.

As traditional von-Neumann computational architectures reach their limits [18, 52] in terms of power consumption, transistor size, and communication, new approaches must be sought. Neuromorphic hardware systems, especially designed to solve neuron dynamics and able to be highly accelerated compared to biological time, are a response to these concerns. Systems such as analog VLSI or the SpiNNaker are advantageous by comparison to software based simulations on commodity computing hardware in areas such as biophysical realism; density of neurons per unit of processing power; and significantly lowered power consumption [19, 28]. This is not to say that simulations of the NeuCube cannot occur on traditional computing architectures; merely that dedicated hardware is advantageous in these areas and may be more appropriate for large-scale modelling.

To address this opportunity, a cross-platform version utilising the PyNN API in Python has been written to extend the modular framework established in [57]. This version is targeted primarily towards neuromorphic hardware platforms but is also applicable to commodity distributed hardware systems depending on the simulation backend chosen.

PyNN [10] is a generic SNN simulation markup framework that allows the user to run arbitrary SNN models on a number of different simulation platforms, including software simulators PyNEST and Brian, and some neuromorphic hardware systems such as SpiNNaker and FACETS/BrainScaleS. It provides a write once, run anywhere (where anywhere is the list of simulators it supports) facility for the development of SNN simulations.

One of the possible neuromorphic platforms for the implementation of a NeuCube SNN prototype system developed in module M1 or in any other modules of the NeuCube architecture, is the SpiNNaker device, currently in development at the University of Manchester. SpiNNaker is a general-purpose, scalable, multichip, multicore platform for the real-time massively parallel simulation of large-scale SNN [19]. Each SpiNNaker chip contains 18 ARM968 subsystems responsible for modelling up to one thousand neurons per core, at very low power consumption. These chips communicate through a custom multicast packet link fabric, and an arbitrary number of these chips can be linked together, with the assumption that the networks simulated exhibit some kind of connection locality. The small-world connection structure used in the NeuCube and its scalable nature are appropriate for implementation on this type of hardware.

Alternative implementations of the NeuCube on neuromorphic hardware are currently being pursued on the INI Neuromorphic VLSI systems and the Zhejiang University FPGA system.

## 6 Dynamic Immersive visualisation and interpretation

The complexity of the network within NeuCube, with regards to the sheer number of neurons and connections, and their 3-dimensional structure requires a specialised visualisation system. These connections and their evolution over time is a significant source of information about the data processed therein. A mere orthographic 45-degree view or a 2D connectivity/weight matrix or of the volume would be insufficient for this amount of data. For this reason, we created a specialised visualisation engine for NeuCube datasets that use JOGL (Java Bindings for OpenGL) and GLSL (OpenGL Shading Language) shaders, enabling us to render up to 1.5 million neurons and their connections on relatively recent graphics cards with 60 frames per second or more.

For efficiency, neurons are rendered as stylised spheres that change the size and the colour based on spiking activity. Connections are rendered as lines with different colours depending on their weight (excitatory: green, inhibitory: red). Spiking activity is also visualised as white pulses travelling along these connections. Additional visualisation modes include detection of find hot paths, connection length, and the ability to view the 3D structure in slices. Using a 3D cursor, neurons can be viewed individually, including their configuration parameters and their current activation potential. There are also controls for playing back, pausing, and changing the speed of spiking activity during the training process as well as during online processing. This is a powerful mechanism for further understanding the temporal behaviour or NeuCube.

By using display hardware such as 3D stereoscopic head-mounted displays (HMD) like the Oculus Rift or the HTC Vive, the perception of the spatial structure of the network and the neuron positions is increased. The full potential of the visualisation unfolds in a motion capture space, where the camera perspective and the cursor node position and orientation are controlled by markers that are attached on the actual HMD and a cursor implement (see fig. 8). This facility enables the user to literally 'walk through' NeuCube, to observe structural and dynamic behaviour, and to select individual neurons with the cursor in a natural manner using a joystick.

Our solution [41, 42] differs from other neural network specific visualisation tools such as BrainGazer [7] and Neuron Navigator (NNG) [38] in that the user can naturally navigate through the 3D space by simply walking and interact with it by gesturing instead of having to use the mouse and memorise keyboard shortcuts. Closer to our visualisation is the work of [62], who are using a Computer Assisted Virtual Environment (CAVE) to visualise the spatial structure and activity of an SNN. However, due to the limited space within a CAVE, navigation by simply walking is not possible and requires indirect interaction, e.g., by using a controller.

These technologies also do not allow for multiple concurrent users, whereas our system is limited only by the available hardware (HMDs) and can easily be extended due to its modular software architecture.

On the numerous occasions where Module M4 has been used at our facility, we observed that people are quickly starting to walk around and to look at structures. The visualisation and interaction metaphors such as selecting and monitoring individual neurons via the 3D cursor are very intuitive for new and experienced users. Overall, M4 is a vital and very useful tool for facilitating a better understanding of NeuCube through 3D rendering of and intuitive navigation within the structure.

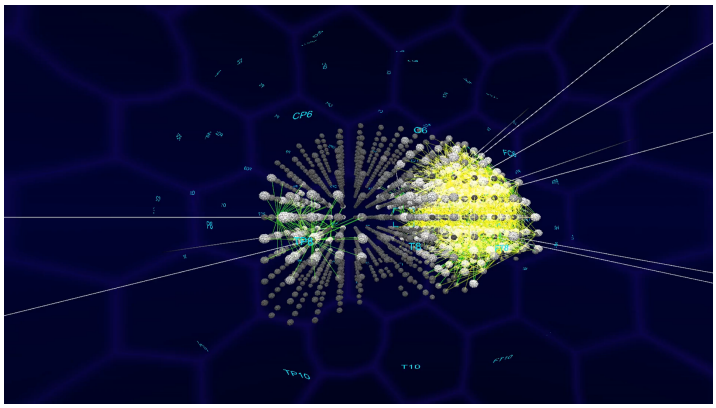


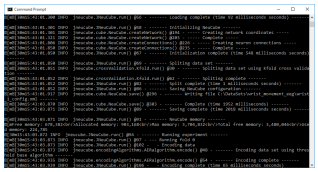

Fig. 8: *3D visualisation of the spatial and dynamic structure of a NeuCube application model.*

## 7 Java implementation of the NeuCube architecture

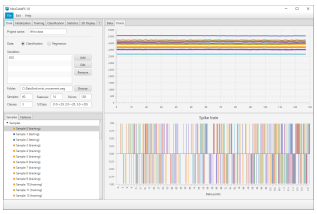
The Java implementation integrates the modules M1, for generic prototyping; M4, for a dynamic 3D visualisation of the NeuCube SNN; M5, for data exchange; M6 for neurogenetic data modelling and; M10, for online learning for real time data analysis. Besides the off-line mode, this version incorporates new methods for modelling large and fast on-line multisensory spatiotemporal stream data. Additionally, it integrates a novel approach to map and analyse data that arises from diverse input variables captured by each element of a sensor network.

The Java implementation is based on the model-view-controller (MVC) design pattern. The model, which captures the behaviour of the NeuCube, is developed in Java 8 (JNeuCube) and can be executed in a command line as shown in figure 9a; and the view and the controller (user interface), in JavaFX 8 (NeuCubeFX) as shown

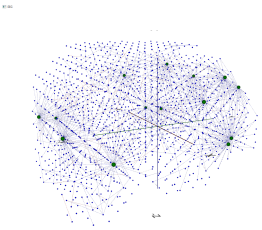
in figures 9b and 9c. Similar to the Matlab implementation, both Java versions are built as a logical step wise processes that can be analysed individually.



(a) JNeuCube execution through a command line.



(b) Graphic user interface.



(c) Dynamic 3D visualisation.

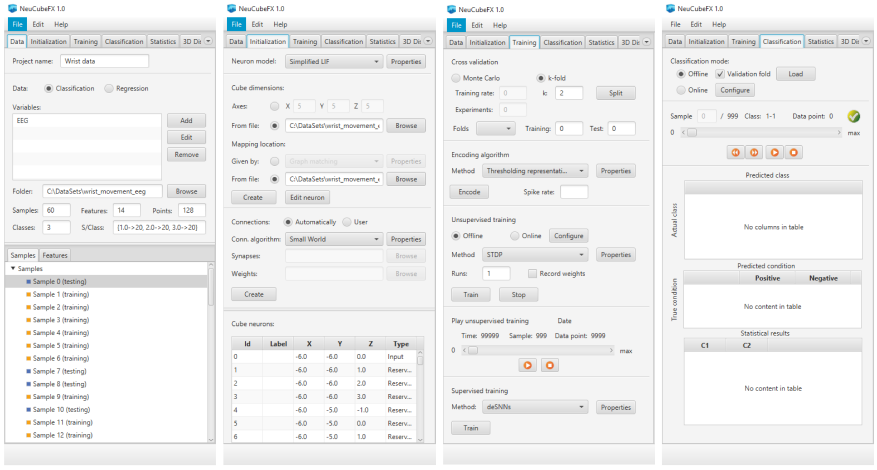
Fig. 9: *JNeuCube MVC architecture.*

## *7.1 Graphic user interface (GUI)*

The software environment is formed by two main sections: the workflow section, which contains NeuCubes functionality, and the visualisation and network analysis section, which allows analyse and interpret spatiotemporal patterns.

### 7.1.1 Workflow

This section is formed by four main modules (see figure 10) that allow users to interact with the algorithms and methods for solving pattern recognition tasks.



(a) Data (b) Initialisation (c) Training (d) Classification

Fig. 10: *Workflow controls in JNeuCube. Every module involves several steps that can be controlled separately.*

### Data module

In this module, the user can import the SSTD and define the type of pattern recognition problem, classification or regression. Here, the user also defines the number of spatiotemporal variables to work with, e.g. the air speed, atmospheric pressure, temperature, and so on. Every variable can be separately analysed in a different 3D SNN that interact together for solving the problem in hand.

The data exchange (import and export) is a pivotal feature for information analysis. In its architecture, this version has an I/O layer that allows the user to process information in an offline mode (data modelling and prototyping), and in an online mode (online learning and recall). It can also store its state at every step (persistence of the model) and brings the possibility of data transfer to and from the Matlab version. Saving a NeuCube project into a file involves storing all information related to the set of time series, the algorithms and methods utilised for building the structure of the SNN, the validation methods, the training algorithms for both unsupervised and supervised learning, the spiking activity and the connection weights before and after the unsupervised learning, the information about the experimental results, and the parameter values for 3D visualisation.

The current version can support three different file formats for different modules: CSV (comma separated plain text), for loading a data set (temporal information) to work in an offline mode for data modelling and prototyping, and for exporting the SNN structure (spiking activity and connectivity); and XML (structured data and communication protocol), for persistence of the NeuCube at every stage. One of the

new features implemented in this Java version is the reading and writing modules for online stream data processing. These modules allow users to implement methods to communicate with any data source and send information to any device.

### Initialisation module

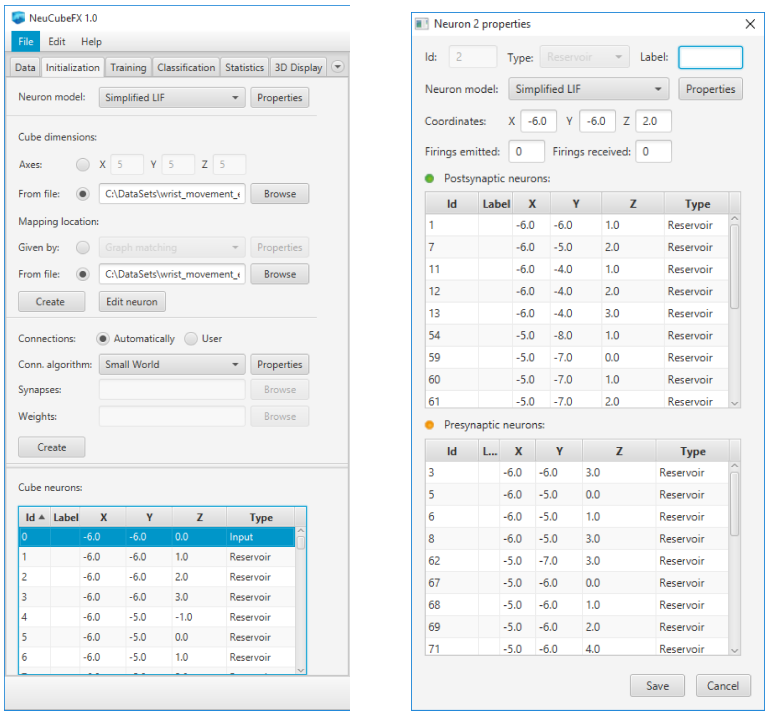
This module allows the user configure the 3D structure of the SNN through three simple steps. In the first step the user can select and configure the properties of the spiking neuron model, so far a simplification of the leaky integrate and fire model, and the Izhikevich model are implemented. However, any other model can easily be implemented. In the second step, the user can define the dimensions of the SNN by choosing the number of neurons per axis (x, y, z) or by choosing a file containing the coordinates of a map (e.g. the Talairach atlas which maps the location of brain structures). A third step is enabled for defining the network connectivity, either using and parameterising a connection algorithm or importing the connections and synaptic weights from files. Finally, in the fourth step, the user can visualise a list with the location and the type (input or reservoir) of all neurons in the SNN. By selecting a neuron in the list, the software shows its details like the number of firings emitted and received, and the pre and post synaptic neurons connected as shown in figure 11.

### Training module

After initialization, the user can start to perform experiments. The first step is configuring the cross validation (Monte Carlo or k-fold) to assess the generalisation of the model to an independent data set. Then, selecting the encoding algorithm transforms the spatio-temporal data into spike trains that NeuCube can process. Hidden patterns can be extracted through an offline or online unsupervised training. In an offline mode, it is possible to record every state of the NeuCube during learning and analyse its behaviour at any point of interest. In an online mode, only the states before and after learning can be analysed. Currently, the spike time dependent plasticity (STDP) rule is implemented, but other learning rules can also be implemented. The last step of this module corresponds to the supervised training. This learning method recognises the temporal pattern produced after the unsupervised learning for classification or regression tasks. Similar to previous modules, different training methods can be implemented; at present, only the deSNN is available.

### Classification

This is the last module that directly involves the algorithms and methods utilised by the NeuCube for pattern recognition. Here, an offline and an online classification of new data can be performed. It is also possible to visualise the spiking activity




(a) Initialisation module (b) Neuron's details

Fig. 11: *Selection and visualisation of the detail of a neuron in the reservoir of JNeuCube.*

produced while the data is introduced to the SNN. Finally, the user can visualise statistical results such as the confusion matrix and its derivations for generalisation analysis.

### 7.1.2 Visualisation and network analysis

The aim of this section is to communicate information clearly and efficiently via statistical graphics, plots, and a 3D dynamic visualisation of the NeuCube SNN. This is an effective module that helps users analyse and reason about data, spatiotemporal patterns and learning processes.

### Dataset visualisation

With this module, users can analyse every sample and feature of a data set. The NeuCubeFX makes data more accessible, understandable and usable by including tables and charts to visualise raw and encoded (spike trains) data as shown in figure 12.

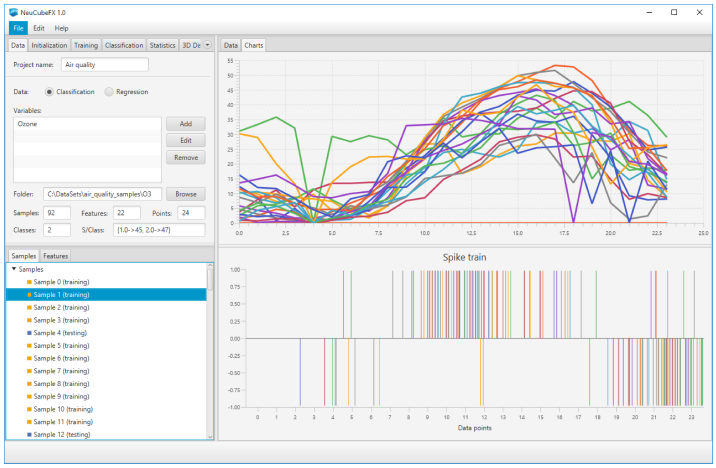


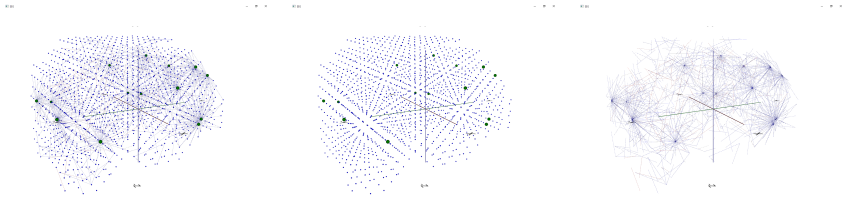

Fig. 12: *JNeuCube Data visualisation before (top chart) and after (bottom chart) encoding.*

### 3D dynamic visualisation

A characteristic feature of the NeuCube is the 3D dynamic visualisation module. Here, research scientists can explore how the data is being processed and interpreted by the NeuCube model.

The unsupervised learning can be dynamically visualised while the system is learning in an online mode (streamed data in real time). In an offline mode, every moment and state of the NeuCube can be saved during the learning process. Then, the user can go to any moment of such process and explore the spiking activity and/or the evolution of connections. At this point, the user can also rotate the NeuCube and analyse different areas of interest, study the properties of a particular neuron, or how the connection between two neurons changed from one time to another.

The recall process can also be dynamically visualised, during the stimulation process, it is possible observe the spiking activity produced by the data sample pattern. Figure 13 shows the 3D visualisation of a brain shaped SNNc.



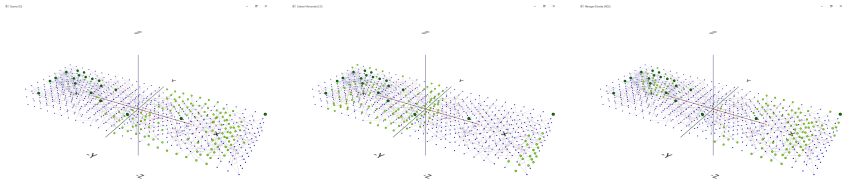
(a) Shows the complete spiking neural network.

(b) Shows the reservoir (blue spheres) and input neurons (dark green spheres).

(c) Shows the connectivity among neurons. The blue and red lines respectively represent positive and negative connections.

Fig. 13: *Snapshots of the JNeuCube dynamic 3D visualisation of the NeuCube model using the Talairach atlas for EEG data processing.*

Mapping and visualising multiple input variables is a novel feature included in this version. For example, in sensor network applications, a node can provide various types of output (temperature, humidity, air pressure, and so on.) at the same time. Here, we decompose a multidimensional SNNc model into several SNNc models, each of them representing one input variable. In Figure 14 we show an example of modelling a sensor network. There, every node measures the concentration of three different greenhouse gases.



(a) $O_3$ (b) $CO$ (c) $NO_2$

Fig. 14: *Example of modelling a greenhouse gases sensor network. Different spiking activity and connection evolution are produced by every input variable (ozone, carbon monoxide and nitrogen dioxide).*

Any component of the 3D visualisation such as neurons, connection or the scale of the network, can be controlled through the 3D display tab (see Figure 15).

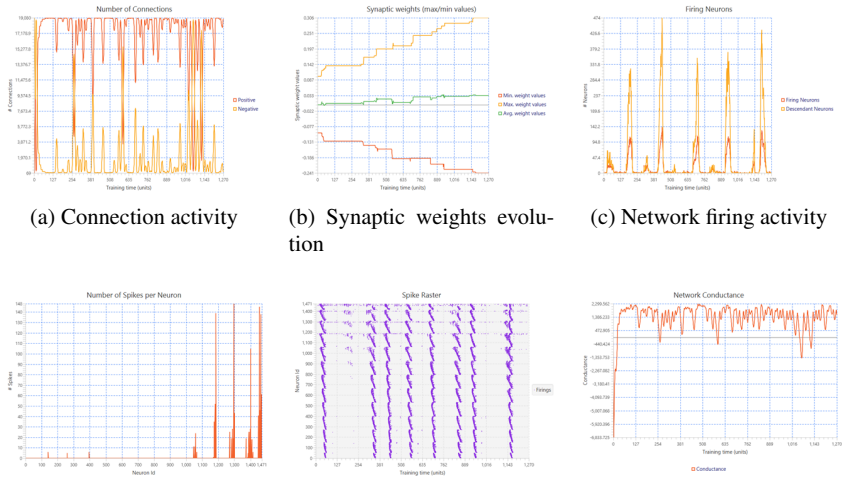

Fig. 15: *3D display control in JNeuCube.*

## *7.2 Network analysis*

Analysing and interpreting spatiotemporal patterns produced during learning is a unique feature that makes NeuCube acts as a white box. The statistics module allows the user to discover new knowledge related to: the number of positive and negative connections (figure 16a), the minimum and maximum values of the synaptic weights (figure 16b), the number of fired neurons and their descendants (figure 16c), the number of spikes produced by each neuron (figure 16d), the spike raster (figure 16e), and the network conductance described (figure 16f).

# 8 Conclusion

The chapter describes the main principles and applications of the first neuromorphic spatio-temporal data machine NeuCube. NeuCube is also a development system for a wide scope of applications. Its current implementation consists of 10 modules, written in 3 languages that are compatible through a common interface and shared data and structure formats. A free copy of the main NeuCube module as a limited and trial version is available from: **http://www.kedri.aut.ac.nz/neucube/**.




(a) Connection activity (b) Synaptic weights evolution (c) Network firing activity

(d) Firing activity per neuron (e) Spike raster (f) Network conductance

Fig. 16: *Network analysis in JNeuCube.*

**Acknowledgements** The NeuCube development is funded by the Auckland University of Technology SRIF grant. Nikola Kasabov and Giacomo Indiveri from ETH and University of Zurich were granted an EU Marie Curie grant in 2011-2012 to start a preliminary research on SNN for spatio-temporal data. The research groups lead by Zeng-Guang Hou and Jie Yang from China contributed to the earlier software implementation of the NeuCube development system.

## References


1. Neucom. http://www.theneucom.com. Accessed: 2015-08-15.
2. Eric B Baum. On the capabilities of multilayer perceptrons. *Journal of complexity*, 4(3):193–215, 1988.
3. Ben Varkey Benjamin, Peiran Gao, Emmett McQuinn, Swadesh Choudhary, Anand R Chandrasekaran, Jean-Marie Bussat, Rodrigo Alvarez-Icaza, John V Arthur, Paul A Merolla, and Kwabena Boahen. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, 102(5):699–716, 2014.
4. David Berlinski. *The advent of the algorithm: the 300-year journey from an idea to the computer*. Houghton Mifflin Harcourt, 2001.
5. Michael R Berthold, Nicolas Cebron, Fabian Dill, Thomas R Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Christoph Sieb, Kilian Thiel, and Bernd Wiswedel. Knime: The konstanz information miner. In *Data analysis, machine learning and applications*, pages 319–326. Springer, 2008.
6. Romain Brette, Michelle Rudolph, Ted Carnevale, Michael Hines, David Beeman, James M Bower, Markus Diesmann, Abigail Morrison, Philip H Goodman, Frederick C Harris Jr, et al. Simulation of networks of spiking neurons: a review of tools and strategies. *Journal of computational neuroscience*, 23(3):349–398, 2007.
7. S. Bruckner, V. Solteszova, E. Groller, J. Hladuvka, K. Buhler, J. Yu, and B. Dickson. BrainGazer - visual queries for neurobiology research. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1497–1504, 2009.
8. Andrea Calimera, Enrico Macii, and Massimo Poncino. The human brain project and neuromorphic computing. *Functional neurology*, 28(3):191–196, 2013.
9. Elisa Capecci, Nikola Kasabov, and Grace Y Wang. Analysis of connectivity in neucube spiking neural network models trained on eeg data for the understanding of functional changes in the brain: A case study on opiate dependence treatment. *Neural Networks*, 68:62–77, 2015.
10. Andrew P Davison, Daniel Brüderle, Jochen Eppler, Jens Kremkow, Eilif Muller, Dejan Pecevski, Laurent Perrinet, and Pierre Yger. PyNN: A Common Interface for Neuronal Network Simulators. *Frontiers in Neuroinformatics*, 2:11, January 2008.
11. T Delbruck and Patrick Lichtsteiner. Fast sensory motor control based on event-based hybrid neuromorphic-procedural system. In *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, pages 845–848. IEEE, 2007.
12. Janez Demšar, Blaž Zupan, Gregor Leban, and Tomaz Curk. *Orange: From experimental machine learning to interactive data mining*. Springer, 2004.
13. Markus Diesmann, Marc-Oliver Gewaltig, and Ad Aertsen. Stable propagation of synchronous spiking in cortical neural networks. *Nature*, 402(6761):529–533, 1999.
14. Maryam Gholami Doborjeh, Elisa Capecci, and Nikola Kasabov. Classification and segmentation of fmri spatio-temporal brain data with a neucube evolving spiking neural network model. In *Evolving and Autonomous Learning Systems (EALS), 2014 IEEE Symposium on*, pages 73–80. IEEE, 2014.
15. Maryam Gholami Doborjeh and Nikola Kasabov. Personalised modelling on integrated clinical and eeg spatio-temporal brain data in the neucube spiking neural network system. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 1373–1378. IEEE, 2016.
16. Rich Drewes. *Brainlab: a toolkit to aid in the design, simulation, and analysis of spiking neural networks with the NCS environment*. PhD thesis, University of Nevada Reno, 2005.
17. Jochen Martin Eppler, Moritz Helias, Eilif Muller, Markus Diesmann, and Marc-Oliver Gewaltig. Pynest: a convenient interface to the nest simulator. *Frontiers in neuroinformatics*, 2, 2008.
18. Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. *ACM SIGARCH Computer Architecture News*, 39(3):365, July 2011.
19. Steve Furber. To Build a Brain. *IEEE Spectrum*, 49(8):44–49, August 2012.

20. Steve B Furber, David R Lester, Luis A Plana, Jim D Garside, Eustace Painkras, Steve Temple, and Andrew D Brown. Overview of the spinnaker system architecture. *IEEE Transactions on Computers*, 62(12):2454–2467, 2013.
21. S Fusi. Spike-driven synaptic plasticity for learning correlated patterns of mean firing rates. *Reviews in the Neurosciences*, 14(1-2):73–84, 2003.
22. Dan FM Goodman. Code generation: a strategy for neural network simulators. *Neuroinformatics*, 8(3):183–196, 2010.
23. Dan FM Goodman and Romain Brette. The brian simulator. *Frontiers in neuroscience*, 3(2):192, 2009.
24. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.
25. Michael L Hines and Nicholas T Carnevale. The neuron simulation environment. *Neural computation*, 9(6):1179–1209, 1997.
26. Jeremy Hsu. Ibm's new brain [news]. *IEEE Spectrum*, 51(10):17–19, 2014.
27. Giacomo Indiveri, Bernabé Linares-Barranco, Tara Julia Hamilton, André Van Schaik, Ralph Etienne-Cummings, Tobi Delbruck, Shih-Chii Liu, Piotr Dudek, Philipp Häfliger, Sylvie Renaud, et al. Neuromorphic silicon neuron circuits. *Frontiers in neuroscience*, 5:73, 2011.
28. Giacomo Indiveri, Bernabé Linares-Barranco, Tara Julia Hamilton, André van Schaik, Ralph Etienne-Cummings, Tobi Delbruck, Shih-Chii Liu, Piotr Dudek, Philipp Häfliger, Sylvie Renaud, Johannes Schemmel, Gert Cauwenberghs, John Arthur, Kai Hynna, Fopefolu Folowosele, Sylvain Saighi, Teresa Serrano-Gotarredona, Jayawan Wijekoon, Yingxue Wang, and Kwabena Boahen. Neuromorphic silicon neuron circuits. *Frontiers in Neuroscience*, 5:73, January 2011.
29. Giacomo Indiveri and Shih-Chii Liu. Memory and information processing in neuromorphic systems. *Proceedings of the IEEE*, 103(8):1379–1397, 2015.
30. Nikola Kasabov. *Evolving connectionist systems: the knowledge engineering approach*. Springer Science & Business Media, 2007.
31. Nikola Kasabov, Kshitij Dhoble, Nuttapod Nuntalid, and Giacomo Indiveri. Dynamic evolving spiking neural networks for on-line spatio-and spectro-temporal pattern recognition. *Neural Networks*, 41:188–201, 2013.
32. Nikola Kasabov and Yingjie Hu. 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, 2010.
33. Nikola Kasabov, Nathan Matthew Scott, Enmei Tu, Stefan Marks, Neelava Sengupta, Elisa Capecci, Muhaini Othman, Maryam Gholami Doborjeh, Norhanifah Murli, Reggio Hartono, et al. Evolving spatio-temporal data machines based on the neucube neuromorphic framework: design methodology and selected applications. *Neural Networks*, 78:1–14, 2016.
34. Nikola Kasabov, Neelava Sengupta, and Nathan Scott. From von neumann, john atanasoff and abc to neuromorphic computation and the neucube spatio-temporal data machine. In *Intelligent Systems (IS), 2016 IEEE 8th International Conference on*, pages 15–21. IEEE, 2016.
35. Nikola K Kasabov. Neucube: A spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data. *Neural Networks*, 52:62–76, 2014.
36. Teuvo Kohonen. The self-organizing map. *Neurocomputing*, 21(1):1–6, 1998.
37. Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
38. C.-Y. Lin, K.-L. Tsai, S.-C. Wang, C.-H. Hsieh, H.-M. Chang, and A.-S. Chiang. The Neuron Navigator: Exploring the information pathway through the neural maze. In *2011 IEEE Pacific Visualization Symposium (PacificVis)*, pages 35–42, 2011.
39. Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997.
40. Wolfgang Maass and Christopher M Bishop. *Pulsed neural networks*. MIT press, 2001.
41. S Marks, Javier Estevez, and Nathan Scott. Immersive visualisation of 3-dimensional neural network structures. 2015.

42. Stefan Marks. Immersive visualisation of 3-dimensional spiking neural networks. *Evolving Systems*, pages 1–9, 2016.
43. Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.
44. Carver Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636, 1990.
45. Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.
46. Ammar Mohemmed, Stefan Schliebs, Satoshi Matsuda, and Nikola Kasabov. Span: Spike pattern association neuron for learning spatio-temporal spike patterns. *International Journal of Neural Systems*, 22(04):1250012, 2012.
47. Thomas Natschläger, Henry Markram, and Wolfgang Maass. Computer models and analysis tools for neural microcircuits. In *Neuroscience Databases*, pages 123–138. Springer, 2003.
48. Steffen Nissen and Evan Nemerson. Fast artificial neural network library. *Available at leenissen. dk/fann/html/files/fann-h. html*, 2000.
49. Jooyoung Park and Irwin W Sandberg. Universal approximation using radial-basis-function networks. *Neural computation*, 3(2):246–257, 1991.
50. Dejan Pecevski. Oger: Modular learning architectures for large-scale sequential processing.
51. Dejan Pecevski, Thomas Natschläger, and Klaus Schuch. Pcsim: a parallel simulation environment for neural circuits fully integrated with python. *Frontiers in neuroinformatics*, 3, 2009.
52. Dimitri Perrin. Complexity and high-end computing in biology and medicine. *Advances in Experimental Medicine and Biology*, 696:377–84, January 2011.
53. Brian Randell. *The origins of digital computers: selected papers*. Springer, 2013.
54. Robert R Schaller. Moore's law: past, present and future. *IEEE spectrum*, 34(6):52–59, 1997.
55. Tom Schaul, Justin Bayer, Daan Wierstra, Yi Sun, Martin Felder, Frank Sehnke, Thomas Rückstieß, and Jürgen Schmidhuber. Pybrain. *The Journal of Machine Learning Research*, 11:743–746, 2010.
56. Ivan Schuler. Neuromorphic computing: From materials to systems architecture. Accessed: 2016-07-16.
57. Nathan Scott, Nikola Kasabov, and Giacomo Indiveri. Neucube neuromorphic framework for spatio-temporal brain data and its python implementation. In *Neural Information Processing*, pages 78–84. Springer, 2013.
58. Neelava Sengupta, Nathan Scott, and Nikola Kasabov. Framework for knowledge driven optimisation based data encoding for brain data modelling using spiking neural network architecture. In *Proceedings of the Fifth International Conference on Fuzzy and Neuro Computing (FANCCO-2015)*, pages 109–118. Springer, 2015.
59. Sen Song, Kenneth D Miller, and Larry F Abbott. Competitive hebbian learning through spike-timing-dependent synaptic plasticity. *Nature neuroscience*, 3(9):919–926, 2000.
60. Donald F Specht. Probabilistic neural networks. *Neural networks*, 3(1):109–118, 1990.
61. Chris Toumey. Less is moore. *Nature Nanotechnology*, 11:2–3, 2016.
62. A. von Kapri, T. Rick, T. C. Potjans, M. Diesmann, and T. Kuhlen. Towards the visualization of spiking neurons in virtual reality. *Studies in Health Technology and Informatics*, 163:685–87, 2011.
63. E Courtenay Wilson. *Parallel implementation of a large scale biologically realistic neocortical neural network simulator*. PhD thesis, University of Nevada Reno, 2001.