NeuCube Neuromorphic Framework for Spatio-Temporal Brain Data and Its Python Implementation

Nathan Scott[1]*, Nikola Kasabov[1], and Giacomo Indiveri[2]

[1] Knowledge Engineering and Discovery Research Institute, Auckland University of Technology, New Zealand
{nascott,nkasabov}@aut.ac.nz

[2] Neuromophic Cognitive Systems, Institute of Neuroinformatics, University of Zurich and ETH Zurich, Switzerland
giacomo@ethz.ch

Abstract. Classification and knowledge extraction from complex spatio-temporal brain data such as EEG or fMRI is a complex challenge. A novel architecture named the NeuCube has been established in prior literature to address this. A number of key points in the implementation of this framework, including modular design, extensibility, scalability, the source of the biologically inspired spatial structure, encoding, classification, and visualisation tools must be considered. A Python version of this framework that conforms to these guidelines has been implemented.

Keywords: NeuCube, Neurogenetic, Neuromorphic, Neuroinformatic, Spiking Neural Network, Pattern Recognition

1 Introduction

The classification and interpretation of spatio-temporal brain data is one of the most complex challenges in modern machine learning. The sheer scale and complexity of the processing units and connections within the human brain lead to highly non-linearly-separable patterns, which make classification of these patterns difficult at best. To this end, a novel computational evironment, the NeuCube, has been proposed in prior literature [1]. This paper introduces a general implementation framework, and the specific version written in the Python programming language.

1.1 Overview of the NeuCube Framework

A detailed discussion of the NeuCube Framework is beyond the scope of this paper. For further details see the paper in which it was first proposed [1].

It consists of the following modules: input information encoding module; NeuCube Reservoir module; output module; gene regulatory network (GRN)

* Corresponding Author

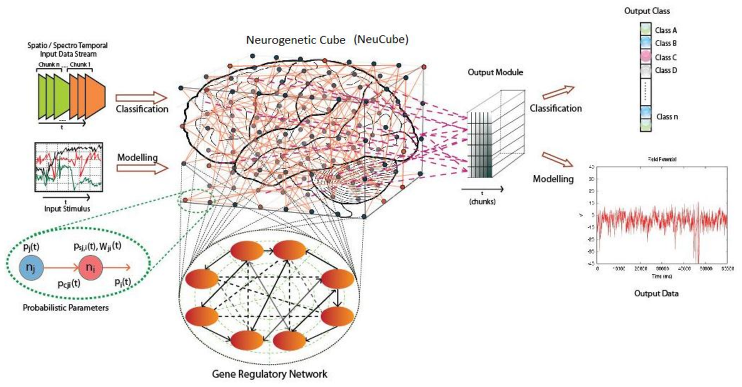

Fig. 1. A block diagram of the NeuCube framework [1]

module. The input module encodes input data (EEG, fMRI and other brain data) into trains of spikes and is then directly entered into the main module – the NeuCube reservoir (NCR). The framework as a whole (incl. encoding and classification tools) is the 'NeuCube Environment'.

Example Application on Neuroinformatic data

Training of a NeuCube Architecture:

1. The NC Reservoir is structured to match the spatial distribution of the EEG or fMRI data (or both). This retains the complex spatio-temporal relationships within the data.
2. The available data is converted into spike trains by the encoding module and entered into the corresponding neuron (neurons) in the NC Reservoir. STDP learning is applied to establish the connection weights of spatial-temporal patterns of pathway connectivity.
3. The output classification module (control module) is trained to recognize the states of the NC Reservoir into predefined classes (activate desired control devices). The polychronisation patterns formed in the NC Reservoir corresponding to each input data pattern are analysed for the purpose of understanding the data.

Recall of the Trained NeuCube Architecture on New Data:

1. New input data is propagated through the NC Reservoir in the same manner as it is initially trained.
2. Output classification (control) results are recorded and the activity of the NeuCube in terms of polychronisation trajectories are analysed and conclusions are made regarding the new input data and the spatio-temporal connectivity and pathways.

Further Adaptation of the NeuCube Architecture: If new data is available that belongs to either existing or new classes, further training of the NeuCube architecture is performed and new output classification (control) neurons are added/evolved and trained in the same way.

2 General Framework for Implementation

The following section describes the general framework for an implementation of the NeuCube system. The italicised headings are aspects that should be considered over and above the standard implementation of the previously discussed theory. The Python implementation of this framework conforms to all of these suggestions, particularly the modular use of framework elements.

Spatial structure: The spatial positioning for the neurons within this reservoir is drawn directly from the Talairach atlas provided by [2]. Each position in 3D space represents a physical location within the brain, and has associated Talairach data, including Broadmann areas and the gyri it is conceptually located in.

At present, in this model, the 1471 neurons used each represent $1\,\mathrm{cm}^3$ of brain tissue. This spatial resolution (number of neurons) should be easily scaled through the software environment, and should also be capable of mapping an aribitrarily large number of neurons within the volume. It is noted here that this arbitrarily large number is reasonably constrained by the computational power available to the environment.

As these locations within the model retain their biological metadata including functional gyri and Broadmann areas, it is easily possible for us to incorporate genetic data, or location specific connection structures based on observed biological phemonmena in these areas. For example, neurons in the area representing the occipital lobe could easily be connected in columns and layers, mimicking more accurately the connections in the human brain's occipital lobe.

Connections: These connections are generated homogenously over the whole model, in a small-world manner. The degree and probability of the small-world connections is configurable.

These small-world connections are also capable of performing Spike Time Dependent Plasticity learning [3]. This learning can be adjusted or removed depending on the specific experiment.

Input Mapping: Due to this realistic spatial structure, we can inject brain data directly into the model location representing the physical location that it was recorded. For example, with electroencephalographic data, we use the excellent mapping presented in [4] to associate a standard International 10-10 or 10-20 electrode location with a specific voxel of brain tissue.

We then use this physical brain location, and map it to the nearest location represented in our NC Reservoir, preserving the complex spatio-temporal relationships within the data. See Table 1 for an example of this electrode – location

Table 1. Example locations of EEG data input positions in Talairach space [4], used to select input neurons in the NeuCube Reservoir, retaining the spatio-temporal relationships within the data.

Labels	Talairach Coordinates			Gyri	BA
	x	y	z		
F7	-52.1 ± 3.0	28.6 ± 6.4	3.8 ± 5.6	L FL Inferior Frontal	45
FC5	-59.1 ± 3.7	3.0 ± 6.1	26.1 ± 5.8	L FL Precentral	6
T7	-65.8 ± 3.3	17.8 ± 6.8	-2.9 ± 6.1	L TL Middle Temporal	21
O1	-25.8 ± 6.3	-93.3 ± 4.6	7.7 ± 12.3	L OL Middle Occipital	18

mapping as used in the software environment. A list of these locations in the same order as their data is presented in the input file is added to the experiment configuration file, which allows this system to adapt to multiple different data structures.

Experiment Configuration: A simple configuration script is generated for each experiment series. This script is written in plain text and contains a number of necessary configuration parameters for the modules used. These parameters are grouped under several headings, defining neuron behaviours, connection parameters, simulation times, classification tools, and so on. These files are easily generated in an automated fashion with an included tool, which is of particular use when performing parameter space searches.

Experiments can be run singly or in a batch job form depending on the number of experiment files provided. Multiple experiments can be run on the same data set, including the comparison of different classification tools or encoding techniques. These experiments can be run remotely via a command line.

Spike Encoding: A number of different spike encoding schemes should be provided, as different forms of input data require different types of preprocessing. These take input and provide an output in a standard form across all encoding schemes. Examples can be found in [1].

Classification: The capacity for multiple classification tools to be implemented is necessary, as these serve different end goals for the system. Particular examples include SPAN [5] for motor signal control and deSNN variants [6] for simple classification.

Statistics: A comprehensive statistics and monitoring package is necessary. This package is responsible primarily for the calculation of statistics related to classification error rates, and connection parameters.

Parallelisation and Clustering: Scalability is handled through the use of local machine parallelisation and multi-machine clustering. A standard protocol such as MPI is encouraged.

Extensibility: The environment should be written in such a way that it is easily extended with a common language. Functionality that may be extended includes neural models (for example, implementing a probabilistic neural model as [7]), connection models, or the addition of tools such as a neurogenetic optimisation module [1] which directly affects neuronal paramters in real time.

2.1 Use of the Tools Independently

As this software environment has been developed in a modular fashion, each of these modules can be used independently or in concert depending on the user's requirements, greatly increasing its flexibility. For example, a data sample can be encoded into spikes using the Population Encoding module, and then presented directly to the deSNN module without using the reservoir. Similarly, a pre-encoded spike train can be presented to the NeuCube Reservoir and the outputs recorded and used elsewhere. See Table 2 for a list of the independent modules currently included in the environment. The main categories of these are briefly discussed below.

Encoding Module: The included spike train encoding tools can be used to convert a number of types of spatio-temporal data into spike trains suitable for general use in either the PyNEST or Brian simulators.

NeuCube Reservoir Module: The NC Reservoir module can be used independently as a standard Liquid State Machine type SNN reservoir. An option is provided to map input channels randomly into the model (as opposed to the fixed spatial locations intended for NI data) and to disable the capacity for STDP learning.

In addition, it can be noted here that this implementation has the capacity to include a standard LSM in place of the NCR, to allow for non-NI data to be modelled through the associated toolchain.

Classification Module: The classification techniques implemented can be applied to a wide range of spiking neural network pattern recognition tasks, independent of the NC module.

3 Python Version

An implementation of this framework has been developed in the Python programming language, using the NEST spiking neural network simulator's PyNEST interface. This implementation has been created following the guidelines previously established in this paper, with particular emphasis on modularity and extensibility.

Table 2. Software tools within the Python NeuCube Environment capable of being used in a modular fashion either within or external to the environment.

Category	Type
Encoding	AER
	BSA
	Population
Reservoir	NeuCube (STDP)
	NeuCube (static synapses)
	Standard LSM
Classification	MultiSPAN
	sSPAN
	deSNNs
	deSNNr
Utilities	SNN Visualisation
	Experiment file generator
	Statistics

Extensibility: This implementation (including the PyNEST simulator) is easily extended through standard Python and C++. Computationally intensive tasks (probabilistic creation of connections, *etc.*) are vectorised, moved down to C++ and wrapped with Python, or compiled just-in-time through the Numba[3] Python library, giving performace comparable to standard C++ code.

Visualisation Tools: A number of tools to visualise various aspects of the system have been developed. Of specific interest is the capacity to visualise both spiking activity and the evolution of neural connectivity over the life cycle of the simulation, in three dimensions. This is particularly useful for knowledge extraction from the model, and in itself represents a novel contribution to the field.

Standard SNN visualisation tools including raster plots of spiking activity and so on are included, and can be extended easily through the matplotlib[4] library used

Hardware Implementations: The Python software implementation will allow direct neuromorphic implementations on SNN chips and systems as follows:

1. Implementation of the NeuCube framework on a SNN SRAM chip. The following features characterise the SRAM SNN chip [8, 9]:
   - It consists of 32 x32 SRAM matrix of weights, each 5 bits (values between 0 and 31); 32 neurons of the adaptive, exponential IF type – each neuron has 2 excitatory and 2 inhibitory inputs to which any of the 32 input dendrites (rows of weights) can be connected;

[3] `http://numba.pydata.org/`
[4] `http://matplotlib.org/`

   - It use AER for both input data, changing the connection weights and for output data streams;
   - It does not have any learning rule implemented in hardware, so it is possible to experiment with different supervised and unsupervised learning rules; Learning (changing the synaptic weights) is calculated outside the chip (in a connected computer) in an asynchronous manner (only synaptic weights that need to change at the current time moment are changed (calculated) and then loaded into the SRAM), allowing us to apply the learning rules of the NeuCube framework.
2. Implementation on the SpiNNaker SNN supercomputer [10]. A Python version of the NC is desirable as the SpiNNaker project is compatible with PyNN scripts [11], providing the NC environment with the capacity to be run on this large-scale, dedicated hardware system.
3. Implementation on a memristor based highly-parallel computation system currently in development [12] is also theoretically possible, and warrants exploration.

3.1 Future Additions

A number of extensions are planned for future versions of this environment. The primary two are mentioned briefly below.

Neurogenetic Optimisation Module: The neurogenetic optimisation module described in [1] will be implemented in the near future. This module takes real genetic data acquired from the Allen Brain Atlas [13] associated with the spatial location of the neurons in the NC Reservoir and uses this to modulate the behaviour of these neurons and their connectome. It is also possible to use this gene and brain composition data to structure the simulation's connections in such a way that they are more biologically plausible.

Multi-Simulator Support: As aspects of the model are already compatible with both the PyNEST and Brian SNN simulators, it is a trivial matter to add the necessary functionality to make this environment fully compatible with Brian.

The feasibility of implementing this environment in PyNN will be explored in the near future. This would provide implicit multi-simulator support [14].

4 Conclusion

This paper has presented an introduction to a new spiking neural network brain data classification framework, and a specific Python implementation of the same.

The environment is also capable of application on general spatio- or spectro-temporal data. As it is implemented in a modular fashion, selected components can be utilised separately or as a whole depending on specific user needs. Simple configuration of experiments through a standard text file format allows for automated generation and execution of large numbers of experiments. Large experiments completed at high speeds are possible through the use of local multithreading, and clustering via MPI. The code is easily extensible, and visualisation tools allow for novel knowledge extraction from the system's dynamic behaviour.

Acknowledgments

The authors would like to thank A. Mohemmed, S. Schliebs, S. Marks, J. Hu, and Y. Chen for valuable discussions on this topic.

References

1. Kasabov, N.,: NeuCube EvoSpike Architecture for Spatio-Temporal Modelling and Pattern Recognition of Brain Signals. in: Mana, Schwenker and Trentin (Eds) ANNPR, Springer LNAI 7477; 225-243 (2012).
2. Lancaster, J.L., Woldorff, M.G., Parsons, L.M., Liotti, M., Freitas, C.S., Rainey, L., Kochunov, P.V., Nickerson, D., Mikiten, S.A., Fox, P.T.: Automated Talairach Atlas labels for functional brain mapping. Human Brain Mapping **10** 120–131 (2000)
3. Kepecs, A., van Rossum, M.C.W., Song, S., Tegner, J.: Spike-timing-dependent plasticity: common themes and divergent vistas. Biological Cybernetics **87** 446–458 (2002)
4. Koessler, L., Maillard, L., Benhadid, A., Vignal, J.P., Felblinger, J., Vespignani, H., Braun, M.: Automated cortical projection of EEG sensors: Anatomical correlation via the international 10-10 system. NeuroImage **46** 64–72 (2009)
5. Mohemmed, A., S.Schliebs, S.Matsuda, N. Kasabov: SPAN: Spike Pattern Association Neuron for Learning Spatio-Temporal Sequences. Int. J. of Neural Systems **22**(4) 1–16 (2012)
6. Kasabov, N., Dhoble, K., Nuntalid, N., G. Indiveri: Dynamic Evolving Spiking Neural Networks for On-line Spatio- and Spectro-Temporal Pattern Recognition. Neural Networks **41** 188–201 (2013)
7. Kasabov, N.: To spike or not to spike: A probabilistic spiking neuron model. Neur. Netw. **23**(1) 16–19 (2010)
8. Indiveri, G., Stefanini, F., Chicca, E.: Spike-based learning with a generalized integrate and fire silicon neuron. In: 2010 IEEE Int. Symp. Circuits and Syst., Paris, 1951–1954 (2010)
9. Indiveri, G., Horiuchi, T.K.: Frontiers in neuromorphic engineering. Frontiers in Neuroscience, **5** 1–2 (2011).
10. Furber, S.: To Build a Brain. IEEE Spectrum, **49**(8) 39–41 (2012)
11. Galluppi, F., Rast, A., Davies, S., Furber, S.: A general-purpose model translation system for a universal neural chip. In: Proc. 17th International Conference on Neural Information Processing. Sydney, Australia. 58–65 (2010)
12. Serrano-Gotarredona, T., Prodromakis, T., Indiveri, G., Linares-Barranco, B., Masquelier, T.: STDP and STDP variations with memristors for spiking neuromorphic learning systems. Frontiers in Neuroscience, **7** (2013)
13. Hawrylycz, M.J. et al.: An anatomically comprehensive atlas of the adult human brain transcriptome. Nature **489** 391–399 (2012)
14. Davison, A.P., Brüderle, D., Eppler, J.M., Kremkow, J., Muller, E., Pecevski, D.A., Perrinet, L., Yger, P.: PyNN: a common interface for neuronal network simulators. Frontiers in Neuroinformatics **2**:11 1–10 (2008)