International Journal of Computer Science and Engineering Research and Development (IJCSERD)
ISSN Print: 2248-9363, ISSN Online: 2248-9371
Volume 15 Issue 2, March-April (2025), pp. 106-113.
DOI**:** https://doi.org/10.63519/IJCSERD_15_02_008

© PRJ Publication

NATURAL NEURON NETWORKS MODELING

Dmytro Rakovskyi *,

Jneopallium, Kharkiv, Ukraine.

*** Corresponding author: Dmytro Rakovskyi**

Abstract

This article presents Jneopallium, a robust framework designed for modeling natural neuron networks with varying levels of detail. Drawing inspiration from historical advancements in neuropsychology and artificial neural networks, Jneopallium offers a modular and flexible approach to simulate neural structures. It allows for the definition of multiple signal types, neuron types, and processing logic, enabling detailed replication of natural cognitive processes. Utilizing Java for implementation, Jneopallium provides an intuitive interface for researchers to define neural architectures, processing rules, and inputoutput logic. This framework aims to bridge the gap between neurobiology and computer science, supporting applications in robotics, AI development, and neuroscience research. The paper details the functional, structural, and IO logic definition processes, showcasing the frameworks versatility and potential for advancing neural network modeling.

Key words: Natural neuron networks, Neural modeling, Jneopallium, Cognitive simulation, Java framework.

Cite this Article: Dmytro Rakovskyi. (2025). Natural neuron networks modeling. *International Journal of Computer Science and Engineering Research and Development (IJCSERD)*, *15*(2), 106-113.
DOI**:** https://doi.org/10.63519/IJCSERD_15_02_008

1. Introduction

First theoretical attempt to describe learning algorithm based on natural neuron nets has been performed by psychologist Donald Olding Hebb in 1940s [1]. Hebbian network has been implemented in code in 1954 at MIT by Farley and Wesley Allison Clark [2]. Psychologist Frank Rosenblatt published idea of perceptron in 1958 [3]. In 1982 neurophysiologically inspired self-organizing maps have been described by Teuvo Kohonen [4][5]. Neocognitron has been designed by Kunihiko Fukushima in 1980 [6]. This invention has been inspired by visual cortex research of neuropsychologists David Hunter Hubel and Torsten Nils Wiesel work [7].

It is safe to say that a lot of core artificial neuron network algorithms are low detailed models of natural neuron networks and/or it parts.

2. Problem formalization

Accordingly to previous section it seems logically to have some unified framework for building custom depth detalization natural neuron networks modelling framework. After high level research of neurobiology and comparison with current artificial neuron network algorithms I have formed next statements:

1. Neurons can process 2 classes of signals: biochemical and bioelectrical. Difference in bioelectrical and biochemical signals propagation is significant.
2. Different signals has different propagation time.
3. The set of neuron receptors defines signals it can process and structure. The set of receptors in different neuron types are different.
4. Cognitive processes is time related.

The modelling framework should be able:

1. Define different types of signals.
2. Define neuron that able to process multiple signal types with different processing logic for each signal type.
3. Define different types of neurons.
4. Define relative processing rates for 2 classes of signals.
5. Define relative processing rate for each type of signal.

These requirements have been used for jneopallium implementation.

3. Natural neuron net modelling process

High level architecture

Jneopallium is set of interfaces and implementations that separate neuron network processing logic from actual neuron and signal types in similar way collections separates storage logic from actual object types that it stores with the help of generics. I have chosen java for implementation because it is suitable for interfaces and generic usage and provides some sort of type safety. All jneopallium code placed in github[8] and gitlab[9] repositories are distributing by BSD 3 – Clause License.

In order to build model user should define signal types, neuron types, input sources and output collector classes. Then describe neuron network structure, specify technical information in configuration file and launch jneupallium with specified path to user defined code jar, neuron network structure and configuration file. The second reason why I have chosen java for implementation is because it can load user defined code in runtime. Jneopallium can work in 3 modes: local, cluster http and cluster grpc. Grpc allows to run jneopallium on FPGAs. For this article I have split modelling process on 3 parts: functional logic definition, structural logic definition and io logic definition. Following 3 sub sections describes modelling process.

Functional logic definition

Modelling process starts from signal definition. User should define all signals in system and weight object that will be used for learning. Next step is neuron interfaces definitions. Each processing mechanism should have separate neuron interface that extends basic INeuron interface. The third step is signal processors implementation. Signal processor should implement ISignalProcessor interface parametrized by signal it process and neuron interface that has suitable mechanisms for processing. Then user should implement neurons by extending Neuron.class and implementing interface or interfaces defined on step 2. Multiple inheritance via interfaces implementation allows user to implement neurons with multiple processing mechanism that can process different signal types. Also, neuron has Axon.class and Dendrites.class. Dendreties incapsulate input addresses (input source name or layer id and neuron id), signal types and weights. These weights apply to input signals and should be used in the learning process. Axon incapsulate output addresses (layer id and neuron id) signal type and weight. These weights apply to output signals and also should be used in the learning process.

To show example of modelling process I have defined 4 signals and 3 neurons in the separate test branch [10]. IntSignal.class represents signal described with integer value. DoubleSignal.class represents signal described with double value. IntProcessor and DoubleProcessor classes describe processing logic for these signals. NeuronIntField and NeuronWithDoubleField interfaces describe neuron with internal structure that allows process IntSignal and DoubleSignal respectively. NeuronC and NeuronB are neuron implementations that process just one type of signal. NeuronA is neuron that can process both signals i.e. it has 2 receptors.

Structural logic definition

After all functional model parts have been defined user should define structure of neural network. I recommend to use statistical approach i.e. find probability appearance of each neuron on each layer. It allows to model horizontal structure. In order to define what neuron order on layers can be user should implement NeighboringRules interface. This feature allows to model vertical neuron structure.

Structure modelling examples are placed here [11]. Structure modelling performed with the help of NeuronNetStructureGenerator. It requires hash map with layer sizes, hash map with statistical properties for each neuron type, list of NeighboringRules and class that implements IConnectionGenerator. IConnectionGenerator describes how to connect neurons.

I/O logic definition

I/O logic describes input sources and output destination. The neuron net can have multiple inputs. To define input source user should implement interface IInitInput. Each input has default processing frequence that shows how often signals from input will be propagated to neurons. Processing frequency can be modified with the help of signal sending to CycleNeuron (more details about it will be in the next sub section). The way how input signals propagate to neurons should be described with the help of implementation of InputInitStrategy inteface. Each input source can have separate InputInitStrategy. If the input is other neuron network output, signals can be send to the neuron network. In this case input should be implement INeuronNetInput interface. This feature can be useful to build modular models in order to simplify learning. To define output destination user should implement IOutputAggregator interface. The example of i/o logic definition placed in this package [12].

Signals processing frequency

The signals processing frequency defined by 2 processing loops. Fast loop process every processing iteration and slow loop process once in n iterations of fast loop. The n defined in CycleNeuron and can be changed with the help of sending signal to layer with id –2147483648 and neuron with id 0. Each signal type and input source has ProcessingFrequency that described with integer field loop and long field epoch. Signal with ProcessingFrequency loop 1 will be processed each time of fast loop processing, with value 2 once in 2 processing, with value 3 once in 3 processing etc. ProcessingFrequency epoch use the same logic but for slow loop. The following code describes all possible signal to CycleNeuron and processing logic [13].

Layer sizing

Layer can be sized with the help of signal sending to LayerManipulatingNeuron. It situated on each layer with id –9 223 372 036 854 775 808 and can create and delete neurons. Here You can find the list of signal and processing logic [14].

Additional features

There exists ability to define any number of discriminators for neuron network. It can be used to implement GAN. Also, user can store and extract parameters in layer.

Configuration files

Examples of configuration files You can find here [15].

4. Application, monetization, competitors

Application

Models built with the help of jneopallium can be used for robotics. The output and input are defined by user so it can directly communicate with controllers. I expect that general AI can be implemented with such approach.

Also, such models can be used for company management in environment with different volatility signals and metrics.

It can be used for natural neuron network modelling especially when should be modeled control structure and structure with different deviations.

It can be used for autonomous mission control when the connection latency to high and exists high conditions and mission flow uncertainty.

Monetization

Jneopallium has few scenarios of monetization. First one is through building models for different products. The other way of monetization is providing hosting services the same way cloud providers do with Spark. The third way is FPGAs optimization for model.

Competitors

The closest competitor for jneopallium is NEURON Simulator [16][17] and CoreNeuron[18]. It allows to build highly detailed models of natural neuron networks. The main difference that jneopallium allows user to choose level of detalization. Jneopallium's main purpose is to be a bridge between neurobiology and computer science. NEURON Simulator and CoreNeuron main purpose is to build exact copy of natural neuron network.

5. Conclusion

Jneopallium represents a significant step forward in the modeling of natural neuron networks, offering a versatile and scalable framework that integrates the complexities of neurobiology with the precision of computer science. By allowing users to define various neuron and signal types, processing logic, and neural structures, Jneopallium facilitates the creation of detailed and functional neural models. Its potential applications span across robotics, artificial intelligence, and neuroscience research, providing a valuable tool for exploring and understanding cognitive processes. As a bridge between the fields of neurobiology and computer science, Jneopallium stands out for its ability to simulate natural neural networks with customizable levels of detail, promising advancements in both theoretical and practical domains.

Authors' contributions:

Architecture design, code implementation, testing – Dmytro Rakovskyi

References

[1] Hebb D (1949). The Organization of Behavior. New York: Wiley. ISBN 978-1-135-63190-1.

[2] Farley B, W.A. Clark (1954). "Simulation of Self-Organizing Systems by Digital Computer". IRE Transactions on Information Theory. 4 (4): 76–84. doi:10.1109/TIT.1954.1057468.

[3] Rosenblatt F (1957). "The Perceptron—a perceiving and recognizing automaton". Report 85-460-1. Cornell Aeronautical Laboratory.

[4] Kohonen T (1982). "Self-Organized Formation of Topologically Correct Feature Maps". Biological Cybernetics. 43 (1): 59–69. doi:10.1007/bf00337288. S2CID 206775459.

[5] Von der Malsburg C (1973). "Self-organization of orientation sensitive cells in the striate cortex". Kybernetik. 14 (2): 85–100. doi:10.1007/bf00288907. PMID 4786750. S2CID 3351573.

[6] Fukushima, Kunihiko (1980). "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position" (PDF). Biological Cybernetics. 36 (4): 193–202. doi:10.1007/BF00344251. PMID 7370364. S2CID 206775608. Archived (PDF) from the original on 3 June 2014. Retrieved 16 November 2013.

[7] Hubel, D. H.; Wiesel, T. N. (1968-03-01). "Receptive fields and functional architecture of monkey striate cortex". The Journal of Physiology. 195 (1): 215–243. doi:10.1113/jphysiol.1968.sp008455. ISSN 0022-3751. PMC 1557912. PMID 4966457.

[8] https://github.com/rakovpublic/jneopallium

[9] https://gitlab.com/rakovpublic/jneopallium

[10] https://github.com/rakovpublic/jneopallium/tree/test/alfaTestAndGettingStarted

[11] https://github.com/rakovpublic/jneopallium/tree/test/alfaTestAndGettingStarted/worker/src/main/java/com/rakovpublic/jneuropallium/worker/test/definitions/structurallogic

[12] https://github.com/rakovpublic/jneopallium/tree/test/alfaTestAndGettingStarted/worker/src/main/java/com/rakovpublic/jneuropallium/worker/test/definitions/ioutils

[13] https://github.com/rakovpublic/jneopallium/tree/master/worker/src/main/java/com/rakovpublic/jneuropallium/worker/net/neuron/impl/cycleprocessing

[14] https://github.com/rakovpublic/jneopallium/tree/master/worker/src/main/java/com/rakovpublic/jneuropallium/worker/net/neuron/impl/layersizing

[15] https://github.com/rakovpublic/jneopallium/tree/test/alfaTestAndGettingStarted/worker/src/main/resources

[16] https://github.com/neuronsimulator/nrn

[17] https://nrn.readthedocs.io/en/8.2.4/

[18] https://github.com/BlueBrain/CoreNeuron