



# N2S3, an Open-Source Scalable Spiking Neuromorphic Hardware Simulator

Pierre Boulet, Philippe Devienne, Pierre Falez, Guillermo Polito, Mahyar Shamsavari, Pierre Tirilly

## ► To cite this version:

Pierre Boulet, Philippe Devienne, Pierre Falez, Guillermo Polito, Mahyar Shamsavari, et al.. N2S3, an Open-Source Scalable Spiking Neuromorphic Hardware Simulator. [Research Report] Université de Lille 1, Sciences et Technologies; CRISTAL UMR 9189. 2017. hal-01432133v2

**HAL Id: hal-01432133**

**<https://hal.science/hal-01432133v2>**

Submitted on 12 Jan 2017

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution - ShareAlike| 4.0 International License

# N2S3, an Open-Source Scalable Spiking Neuromorphic Hardware Simulator

Pierre Boulet\*, Philippe Devienne\*, Pierre Falez\*, Guillermo Polito\*, Mahyar Shahsavari\* and Pierre Tirilly\*

\*Univ. Lille, CNRS, Centrale Lille,

UMR 9189 – CRISTAL – Centre de Recherche en Informatique, Signal et Automatique de Lille,

F-59000, Lille, France

Email: Firstname.Lastname@univ-lille.fr

**Abstract**—One of the most promising approaches to overcome the end of Moore’s law is neuromorphic computing. Indeed, neural networks already have a great impact on machine learning applications and offer very nice properties to cope with the problems of nanoelectronics manufacturing, such as a good tolerance to device variability and circuit defects, and a low activity, leading to low energy consumption. We present here N2S3 (for Neural Network Scalable Spiking Simulator), an open-source simulator that is built to help design spiking neuromorphic circuits based on nanoelectronics. N2S3 is an event-based simulator and its main properties are flexibility, extensibility, and scalability. One of our goals with the release of N2S3 as open-source software is to promote the reproducibility of research on neuromorphic hardware. We designed N2S3 to be used as a library, to be easily extended with new models and to provide a user-friendly special purpose language to describe the simulations.

## I. INTRODUCTION

Neuromorphic computing has the potential to bring very low power computation to future computer architectures and embedded systems [1]. Indeed, parallel neuromorphic computing, by performing computation and storage on the same devices, can overcome the von Neumann bottleneck. Several large projects are based on neuromorphic systems, such as the EU Human Brain Project [2], the DARPA/IBM SYNAPSE project [3], and deep learning research led by Google and Facebook, among others.

Recently, emerging nano-scale devices have demonstrated novel properties for producing new memories and unconventional processing units. One of those is the memristor, that was hypothetically presented by Leon Chua in 1971 [4]; after a few decades, HP was the first to announce a successful memristor fabrication [5]. The unique properties of memristors, such as extreme scalability, flexibility thanks to their analog behavior, and their ability to remember their last state, make memristors very promising candidates to be used as synapses in Spiking Neural Networks (SNN) [6].

Given their potential of very low power execution and their capability to handle natural signals, we focus on SNN. A comprehensive introduction and literature review about SNN was published by Paugam-Moisy and Bohte in 2012 [7]. The authors explore the computational capabilities of SNN, their learning capabilities, and their simulation.

Brette *et al.* [8] surveyed and discussed the existing work on SNN simulation in 2007. All the simulators discussed in

this report as well as the more recent Brian [9] target the simulation of biological SNN. More recently, Bichler *et al.* [10] proposed Xnet, a C++ event-driven simulator dedicated to the simulation of hardware SNN. In our work, we share the goals of Xnet: “intermediate modeling level, between low-level hardware description languages and high-level neural networks simulators used primarily in neurosciences”, and “the integration of synaptic memristive device modeling, hardware constraints and any custom features required for the targeted application”. In addition to these goals, we put an emphasis on *flexibility and usability* to allow the study of various kinds of hardware designs (possibly by other researchers than us), *scalability* to simulate large hardware SNNs, and *software engineering best practices* (robust and extensible software architecture for maintainability, extensive test suite, continuous integration, open-source distribution).

In this report we present N2S3 (Neural Network Scalable Spiking Simulator, pronounced “Nessie”, hence its logo), an event-driven simulator dedicated to the exploration of hardware SNN architectures. The internals of N2S3 are based on the exchanges of messages between concurrent actors [11], mimicking the exchange of spikes between neurons. N2S3 has been developed from the ground up for extensibility, allowing to model various kinds of neuron and synapse models, various network topologies (especially, it is not restricted to feed-forward networks), various learning procedures, various reporting facilities, and to be user-friendly, with a domain specific language to easily express and run new experiments. It is available as open-source software at <https://sourcesup.renater.fr/wiki/n2s3> so that its users can share their models and experimental settings to enable others to reproduce their results. In this spirit, N2S3 is distributed with the implementation of two “classical” experiments: handwritten digit recognition on the MNIST dataset [12], [13] and the highway vehicle counting experiment [14].

In the remainder of this report, we will first detail the fundamental architectural choices of N2S3, then the models that are already implemented in N2S3, and finally the features that N2S3 provides to implement and run experiments.



Fig. 1. N2S3 Logo

## II. EVENT-DRIVEN SIMULATION ARCHITECTURE

### A. Event-Driven vs Clock-Driven Simulation

SNN are essentially defined by standard differential equations, but because of the discontinuities caused by the spikes, designing an efficient simulation of spiking neural networks is a non-trivial problem. There are two families of simulation algorithms: event-based simulators and clock-based ones. Synchronous or “clock-driven” simulation simultaneously updates all the neurons at every tick of a clock, and is easier to code, especially on GPUs, for getting an efficient execution of data-parallel learning algorithms. Event-driven simulation behaves more like hardware, in which conceptually concurrent components are activated by incoming signals (or events).

Event-driven execution is particularly suitable for untethered devices such as neurons and synapses, since the nodes can be put into a sleep mode to preserve energy when no interesting event is happening. Energy-aware simulation needs information about active hardware units and event counters to establish the energy usage of each spike and each component of the neural network. Furthermore, as the learning mechanisms of spiking neural networks are based on the timings of spikes, the choice of the clock period for a clock-based simulation may lead either to imprecision or to a higher computational cost.

There is also a fundamental gap between this event-driven execution model and the clock-based one: the first one is independent on the hardware architecture of computers on which it is running. So, event-driven simulators can naturally run on a grid of computers, with the caveat of synchronization issues in the management of event timings.

### B. Technological Choices: Scala and Akka

To address our concurrency and distributability requirements (i.e., ability to scale out a simulation on several computers to handle large networks) we have chosen to use the Scala programming language [15] along with the Akka actor library [11].

### C. Software Architecture

The architecture of N2S3 is organized in Akka actors. Our choice for an actor model aims mainly for the distribution and scalability of large neural networks. Each entity of the simulation is deployed in a concrete actor that is assigned to a concrete machine at runtime. The core of a typical neural network simulation in N2S3 is composed mainly by the following network entities:

a) *Containers*: Containers are network entities in charge of organizing the network structure. Their main responsibility is to contain network entities and dispatch messages to their children.

b) *Neurons*: The basic building blocks of a neural network. In N2S3, a neuron is composed of both its nucleus (the soma) and its incoming connections (dendrites and associated synapses).

Each N2S3 actor has a network container capable of containing one or multiple entities. This particularity allows us to control how distribution and parallelism is managed in each simulation. Figure 2 illustrates the architecture with an example

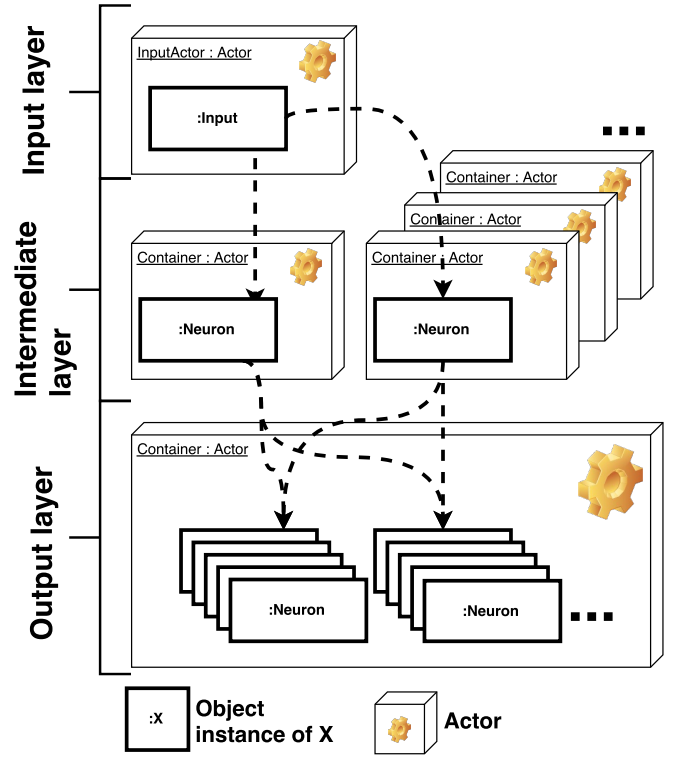


Fig. 2. **N2S3 Architecture.** A network is organized in actors that may contain one or more network entities. Such entities could be for example, inputs or any other.

network. In this figure, the simulation is made of one input and a neural network organized in two layers. The input of the simulation resides in one actor, the hidden layer is split in one actor per neuron, and all neurons of the output layer reside in a single actor. The reason why one would put several neurons in the same or a different actor is based on the scalability and the synchronization choices for the experiment. The topology of the network (discussed in more details in Section III-C) is one of the major influence on these questions.

The communication between simulation entities requires to identify each object by a URI made of a pair (actor, local\_identifier). The component actor is the Akka actor that contains the entity and local\_identifier is a path that identifies each object uniquely within its actor.

Since actors are inherently concurrent, one concern is how the temporal order of messages is guaranteed during the simulation. To do so, N2S3 allows to configure several levels of synchronization to be used by the simulation designer. On one end of the spectrum, N2S3 may make use of a unique synchronizer for the simulation, which will ensure that no causality issues happen but can create a bottleneck that will affect the performance of the simulation. On the other end of the spectrum, we can also configure N2S3 to use a synchronization mechanism which is local to each neuron. The latter policy enables better parallelism, but may cause some temporal consistency problems.

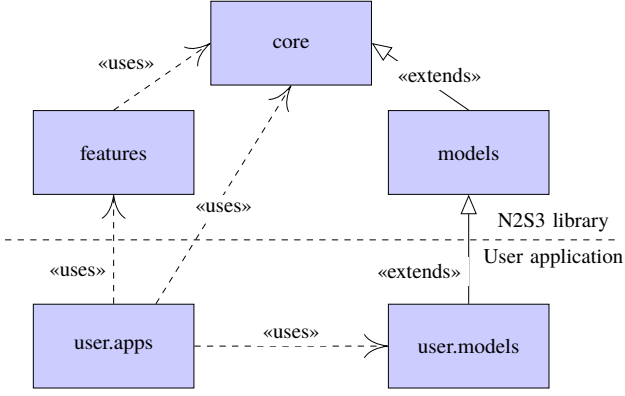


Fig. 3. N2S3 Packages

### III. NEURON, SYNAPSE, NETWORK MODELING

#### A. Neuron Modeling

The neuron is a dynamic element and processing unit that emits output pulses whenever the excitation exceeds some threshold. The resulting sequence of pulses or spikes contains all the information that is transmitted from one neuron to another one. As we have chosen event-driven simulation, the state of a neuron is updated only when an event representing a spike is received.

We provide by default a *Leaky-Integrate-and-Fire (LIF)* neuron model [16]. There are several reasons for using a LIF model:

- CMOS technology fabrication of this model is available [17], [18];
- it is fast enough to simulate, and in particular it is effective for large-scale network simulations [8].

In LIF, neurons integrate the spikes coming from other neurons. These spikes can change the internal potential of the neuron, which is known as the *membrane potential* or *state variable* of the neuron. When the membrane potential reaches a given threshold voltage after integrating enough input spikes, an action potential occurs; in other words, the neuron fires, generating an output spike. This is done by updating the membrane potential using the value computed when receiving the last spike and the analytical solution to the differential equations defining the behavior of the LIF neuron.

In order to improve the learning capabilities of neural networks, we provide two refinements of the neuron model: homeostasis and a refractory period. Homeostasis [19] dynamically adapts the threshold of each neuron to the activity of this neuron to prevent a neuron from being over-active or inactive. To allow more specialization of the neurons, we include a refractory period after the firing of a neuron during which it ignores incoming spikes.

#### B. Synapse modeling and learning

A synapse operates as a plastic controller between two neurons. This plasticity is believed to be the origin of the learning and memorization capabilities of the brain [20]. Hence,

hardware spiking neural networks usually use some kind of synaptic plasticity to enable learning. In N2S3, we have modeled and simulated hardware synapses and one standard learning behavior, namely Spike Timing-Dependent Plasticity (STDP).

STDP is a local weight modification based on the timing difference between presynaptic spikes and postsynaptic spikes. It increases the connectivity between the neurons when there is a temporal causality between the spikes they emitted. This is implemented by locally remembering the past spikes and updating the synaptic weight according to the chosen STDP rule.

Using a memristor as a nonvolatile synaptic memory has been proposed in previous work [6], [21]–[23]. The basic artificial synapse model that we provide is taken from [24], [25], which describes a nonvolatile memristor suitable for event-driven and STDP computation. In addition, we designed a new model of synapse which is able to forget unimportant events and remember significant events by combining a nonvolatile memristor and a volatile one [26].

#### C. Network Topologies

Another purpose of N2S3 is to allow an easy exploration of different neural network configurations. In order to facilitate the network construction, neurons are gathered into groups. Those groups, which usually represent neuron layers, are organized into a specified shape. This information allows the automation of the builder and of the visualization creation.

Inside a group one can specify the use of lateral inhibition to implement the winner-take-all rule [27] that states that a spiking neuron inhibits its neighbors during a short time period to allow the specialization of the neurons and hence to enhance unsupervised learning.

Neuron groups can be connected to each other. Connection patterns are managed by different connection policies. By default, N2S3 uses unidirectional full connections (each neuron of the input group is connected to every neuron of the output group). Other policies are also bundled with N2S3, such as the *one-to-one* policy, which connects each neuron of the input group to only the corresponding neuron of the output group, and a random policy, which creates connections according to a given probability distribution, e.g. to enable Reservoir Computing [28] modeling.

Several builders have the responsibility to create different topologies. They can use the information of an existing layer (e.g., the shape of the neuron group), if there is one, to construct a new one. Builders bundled with N2S3 include, for instance, builders for the typical layers of Convolutional Neural Networks [13].

### IV. N2S3 FEATURES

#### A. Input Processing

The simulator uses a piped stream system to provide stimuli to the network entities. The input process typically starts by an input reader, which reads data from files or any external

source, followed by a number of streams that filter the input data before feeding it to the network.

Input readers provided with N2S3 allow to read data in a variety of format, including standard formats such as Address Event Representation (AER), a data format used by spike-based cameras, or MNIST, used in a standard dataset for handwritten digit recognition (see Section IV-E for details about the data).

Subsequent filter streams available in N2S3 include coding streams, which convert raw numerical data into sequences of spike timings, spike presentation streams (e.g., repeating input spike over a given period, or shuffling spikes), and modifier streams, that alter the input spikes (e.g., by adding noise). Users are free to use one or multiple input readers and to combine any number of filter streams in any order; they may also easily create their own readers and filters.

### B. Visualization tools

Users may observe simulation outputs (spikes, weight values...) through network observers. Network observers follow the observer pattern by subscribing to events in the simulation (e.g., when a spike happens), perform some calculations on such events, and make them visible to the user. Examples of such observers range from textual loggers to dynamic visualizations of the spikes of each neuron. Concretely, N2S3 provides a spike activity map of the network, a synaptic weight evolution visualizer, and the calculation of evaluation metrics (e.g. recognition rates, confusion matrices...).

### C. Experiment Specification Language

N2S3 includes a dedicated internal Domain Specific Language (DSL) that aims to simplify the creation of simulations. At a higher level of abstraction, users can design experiments (network topology, neuron and synapse properties, observation units...) without having to deal with core features such as synchronization or actor policies. The snippet of code below illustrates (1) the creation of a layer with 18 standard LIF neurons with a 20 mV threshold, (2) its full connection to an existing layer (named `outputLayer`), and the addition of two observers: (3) a standard weight observer and (4) a custom (i.e., defined by the user) spike observer. The DSL also allows the definition of different stages for the simulation (e.g., splitting the simulation into a training phase and a test phase).

```
val hiddenLayer =
  n2s3.createNeuronGroup() // (1)
    .setNumberOfNeurons(18)
    .setDefaultNeuronConstructor( () => {
      new QBGNeuron()
    })
    .setProperty(NeuronThreshold, 20 millivolts)
hiddenLayer.connectTo(outputLayer) // (2)
n2s3.createSynapseWeightGraphOn(hiddenLayer,
  outputLayer) // (3)
n2s3.addNetworkObserver(new SpikeLogger) // (4)
```

At a lower level, the DSL can specify how neurons are organized within actors (e.g., deploying all neurons within the same actor or each neuron in a specific actor). Users may use pre-existing policies or define their own.

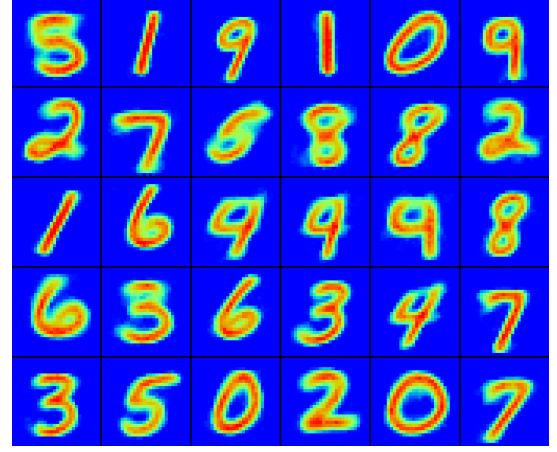


Fig. 4. Heat map of the synaptic weights after learning the MNIST data base with 30 neurons on the output layer.

### D. Software Engineering Practices

To ensure the quality of the simulator's core and its surrounding features, we follow several good practices of software engineering. We use automated tests (unit tests and integration tests) to validate the basic behavior of the simulator. These unit tests are run both locally and within a continuous integration environment to detect regressions in the project as soon as possible. N2S3's codebase is open-source, and its developers perform periodic code reviews on it.

### E. Standard experiments

N2S3 comes with a set of pre-implemented experiments from the literature. These implementations both demonstrate the features and simulation accuracy of N2S3 and provide code snippets to allow users to understand the ESL and help them designing their own simulations. Specifically, experiments for two standard tasks are provided with N2S3: handwritten digit recognition on the MNIST dataset and car counting on the Freeway dataset. In both cases N2S3 provide simulation results comparable to those reported in the literature.

MNIST [12] is a standard dataset for automatic handwritten digit recognition. Since its creation, it has been extensively used to evaluate algorithms for machine learning, computer vision and document recognition. The task is to learn to identify which digit is represented on each image. The dataset includes 60,000 greyscale images of size  $128 \times 128$  pixels, divided into a training set (50,000 images) and a test set (10,000). The implementation of SNN learning on MNIST provided with N2S3 follows the experimental settings from [29]:  $128 \times 128$  inputs (1/pixel) and one output layer (10 to 300 neurons). Figure 4 shows the result of learning the MNIST database with 30 neurons on the output layer and winner-take-all activated to enhance learning. The initial synaptic weights are random.

Freeway [30] is an AER (Address Event Representation) video of a Freeway in Pasadena. It is a standard video demonstrating the capabilities of spike-based cameras. The task here is to count the number of vehicles passing over each



Fig. 5. Input data for the freeway experiment coming from a spike-based camera. The spikes represent a variation of intensity for a given pixel and are generated asynchronously.

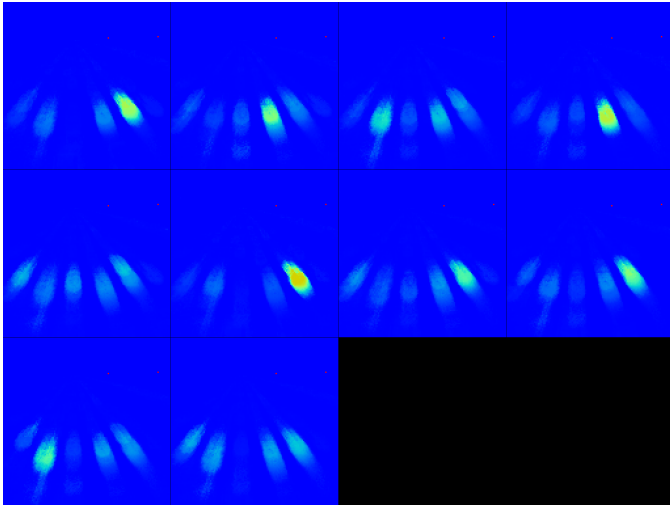


Fig. 6. Heatmap showing the reconstruction of the contribution of each input pixel to the activity of the 10 output neurons for the freeway experiment. One can see that some neurons have clearly specialized to detect vehicles on a particular lane.

of the six lanes of the freeway. The input data is the spikes recorded by a spiking camera of resolution  $128 \times 128$  pixels over a sequence of 78.5 seconds (5.2 millions spikes). The implementation of the Freeway experiment provided with N2S3 reproduces the experimental architecture and settings described in [14]:  $2 \times 128 \times 128$  inputs, one hidden layer (60 neurones), one output layer (10 neurones), with lateral inhibition on every layer. Figure 5 represents the input of the neural network and figure 6 shows the result of the unsupervised learning process.

## V. CONCLUSION

We have presented in this article a spiking neural network simulator, N2S3. This simulator is dedicated to nanoelectronics-based hardware neural networks. Its simulation strategy is event-based, for precision and flexibility sakes. As one of the main goals of N2S3 is to promote open research data, i.e. the open distribution of models and experiments for research result reproducibility, it is distributed as open-source software (at <https://sourcesup.renater.fr/wiki/n2s3>), and a lot of effort has been put in the software architecture, maintainability and enabling features (I/O, graphical outputs, logging, domain specific language). As of version 1.0 (January 2017), N2S3 is distributed with a few neuron and synapse models. It can simulate not only feed-forward networks but also convolutional or even recurrent networks. Finally, it is extensively extensible.

We are planning to periodically release new versions in a timeboxed fashion and to perform more quality measurements in the near future, for example: continuous benchmarking and performance testing to detect non-functional regressions, adding automated code quality and architectural rule enforcement to detect potential bugs, ensuring the respect for architectural and programming idioms, and agilizing the code reviews.

On the functional side, future work includes new models of neurons and synapses, new learning algorithms and procedures, noise modeling, energy modeling, automatic design space exploration, and new applications.

## ACKNOWLEDGMENT

This work has been partly funded by IRCICA (Univ. Lille, CNRS, USR 3380 – IRCICA, F-59000 Lille, France).

The authors would like to thank several colleagues from IEMN for many fruitful discussions about memristor and under the threshold CMOS device and circuit modeling: Fabien Alibart, Alain Cappy, François Danneville, and their teams.

## REFERENCES

- [1] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, Aug. 2014. [Online]. Available: <http://science.sciencemag.org/content/345/6197/668>
- [2] H. Markram, "The human brain project," *Sci. Am.*, vol. 306, no. 6, pp. 50–55, Jun. 2012.
- [3] R. Ananthanarayanan, S. K. Esser, H. D. Simon, and D. S. Modha, "The cat is out of the bag: cortical simulations with 109 neurons, 1013 synapses," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, Nov. 2009, pp. 1–12.

- [4] L. Chua, "Memristor-The missing circuit element," *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507–519, Sep. 1971.
- [5] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, May 2008. [Online]. Available: <http://www.nature.com/nature/journal/v453/n7191/full/nature06932.html>
- [6] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale Memristor Device as Synapse in Neuromorphic Systems," *Nano Lett.*, vol. 10, no. 4, pp. 1297–1301, Apr. 2010. [Online]. Available: <http://dx.doi.org/10.1021/nl904092h>
- [7] H. Paugam-Moisy and S. Bohte, "Computing with Spiking Neuron Networks," in *Handbook of Natural Computing*, G. Rozenberg, T. Bäck, and J. N. Kok, Eds. Springer Berlin Heidelberg, Jan. 2012, pp. 335–376. [Online]. Available: [http://link.springer.com/referenceworkentry/10.1007/978-3-540-92910-9\\_10](http://link.springer.com/referenceworkentry/10.1007/978-3-540-92910-9_10)
- [8] R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. M. Bower, M. Diesmann, A. Morrison, P. H. Goodman, F. C. Harris, M. Zirpe, T. Natschlager, D. Pecevski, B. Ermentrout, M. Djurfeldt, A. Lansner, O. Rochel, T. Vieville, E. Muller, A. P. Davison, S. E. Boustani, and A. Destexhe, "Simulation of networks of spiking neurons: A review of tools and strategies," *J Comput Neurosci*, vol. 23, no. 3, pp. 349–398, Jul. 2007. [Online]. Available: <http://link.springer.com/article/10.1007/s10827-007-0038-6>
- [9] D. F. M. Goodman, R. Brette, D. Goodman, and R. Brette, "Brian: a simulator for spiking neural networks in Python," *Front. Neuroinform.*, vol. 2, p. 5, 2008. [Online]. Available: <http://journal.frontiersin.org/article/10.3389/neuro.11.005.2008/full>
- [10] O. Bichler, D. Roclin, C. Gamrat, and D. Querlioz, "Design exploration methodology for memristor-based spiking neuromorphic architectures with the Xnet event-driven simulator," in *2013 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, Jul. 2013, pp. 7–12.
- [11] D. Wyatt, *Akka Concurrency*. USA: Artima Incorporation, 2013.
- [12] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
- [13] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995. [Online]. Available: [https://www.researchgate.net/profile/Yann\\_Lecun/publication/2453996\\_Convolutional\\_Networks\\_for\\_Images\\_Speech\\_and\\_Time-Series/links/0deec519dfa2325502000000.pdf](https://www.researchgate.net/profile/Yann_Lecun/publication/2453996_Convolutional_Networks_for_Images_Speech_and_Time-Series/links/0deec519dfa2325502000000.pdf)
- [14] O. Bichler, D. Querlioz, S. J. Thorpe, J.-P. Bourgoin, and C. Gamrat, "Extraction of temporally correlated features from dynamic vision sensors with spike-timing-dependent plasticity," *Neural Networks*, vol. 32, pp. 339–348, Aug. 2012.
- [15] M. Odersky, P. Altherr, V. Cremet, I. Dragos, G. Dubochet, B. Emir, S. McDermid, S. Micheloud, N. Mihaylov, M. Schinz, E. Stenman, L. Spoon, and M. Zenger, "An overview of the Scala programming language (2nd Edition)," École Polytechnique Fédérale de Lausanne (EPFL), Technical Report LAMP-REPORT-2006-001, 2006.
- [16] W. Maass and C. M. Bishop, Eds., *Pulsed Neural Networks*. Cambridge, MA, USA: MIT Press, 1999.
- [17] E. Chicca, D. Badoni, V. Dante, M. D'Andreagiovanni, G. Salina, L. Carota, S. Fusi, and P. D. Giudice, "A VLSI recurrent network of integrate-and-fire neurons connected by plastic synapses with long-term memory," *IEEE Transactions on Neural Networks*, vol. 14, no. 5, pp. 1297–1307, Sep. 2003.
- [18] S.-C. Liu and R. Douglas, "Temporal coding in a silicon network of integrate-and-fire neurons," *IEEE Transactions on Neural Networks*, vol. 15, no. 5, pp. 1305–1314, Sep. 2004.
- [19] E. Marder and J.-M. Goaillard, "Variability, compensation and homeostasis in neuron and network function," *Nat Rev Neurosci*, vol. 7, no. 7, pp. 563–574, Jul. 2006. [Online]. Available: <http://www.nature.com/nrn/journal/v7/n7/full/nrn1949.html>
- [20] S. J. Martin, P. D. Grimwood, and R. G. Morris, "Synaptic plasticity and memory: an evaluation of the hypothesis," *Annu. Rev. Neurosci.*, vol. 23, pp. 649–711, 2000.
- [21] F. Alibart, S. Pleutin, D. Guérin, C. Novembre, S. Lenfant, K. Lmimouni, C. Gamrat, and D. Vuillaume, "An Organic Nanoparticle Transistor Behaving as a Biological Spiking Synapse," *Adv. Funct. Mater.*, vol. 20, no. 2, pp. 330–337, Jan. 2010. [Online]. Available: <http://onlinelibrary.wiley.com/doi/10.1002/adfm.200901335/abstract>
- [22] H. Kim, M. P. Sah, C. Yang, T. Roska, and L. O. Chua, "Memristor Bridge Synapses," *Proceedings of the IEEE*, vol. 100, no. 6, pp. 2061–2070, Jun. 2012.
- [23] G. Indiveri, B. Linares-Barranco, R. Legenstein, G. Deligeorgis, and T. Prodromakis, "Integration of nanoscale memristor synapses in neuromorphic computing architectures," *Nanotechnology*, vol. 24, no. 38, p. 384010, Sep. 2013, arXiv: 1302.7007. [Online]. Available: <http://arxiv.org/abs/1302.7007>
- [24] D. Querlioz, O. Bichler, P. Dollfus, and C. Gamrat, "Immunity to Device Variations in a Spiking Neural Network With Memristive Nanodevices," *IEEE Transactions on Nanotechnology*, vol. 12, no. 3, pp. 288–295, May 2013.
- [25] D. Querlioz, P. Dollfus, O. Bichler, and C. Gamrat, "Learning with memristive devices: How should we model their behavior?" in *2011 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, Jun. 2011, pp. 150–156.
- [26] M. Shahsavari, P. Falez, and P. Boulet, "Combining a Volatile and Nonvolatile Memristor in Artificial Synapse to Improve Learning in Spiking Neural Networks," in *12th ACM/IEEE International Symposium on Nanoscale Architectures (Nanoarch 2016)*, Beijing, China, Jul. 2016.
- [27] W. Maass, "On the Computational Power of Winner-Take-All," *Neural Computation*, vol. 12, no. 11, pp. 2519–2535, Nov. 2000. [Online]. Available: <http://dx.doi.org/10.1162/089976600300014827>
- [28] D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt, "An experimental unification of reservoir computing methods," *Neural Networks*, vol. 20, no. 3, pp. 391–403, Apr. 2007. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S089360800700038X>
- [29] D. Querlioz, W. S. Zhao, P. Dollfus, J.-O. Klein, O. Bichler, and C. Gamrat, "Bioinspired Networks with Nanoscale Memristive Devices that Combine the Unsupervised and Supervised Learning Approaches," 2012, pp. 203–210.
- [30] "DVS128 Dynamic Vision Sensor Silicon Retina Data." [Online]. Available: <https://sourceforge.net/p/jaer/wiki/AER%20data/>