Neural Processing Letters (2020) 52:1675–1701
https://doi.org/10.1007/s11063-020-10322-8

Spiking Neural Networks: Background, Recent Development and the NeuCube Architecture

Clarence Tan[1] · Marko Šarlija[2] · Nikola Kasabov[1]

Published online: 13 August 2020

Abstract

This paper reviews recent developments in the still-off-the-mainstream information and data processing area of spiking neural networks (SNN)—the third generation of artificial neural networks. We provide background information about the functioning of biological neurons, discussing the most important and commonly used mathematical neural models. Most relevant information processing techniques, learning algorithms, and applications of spiking neurons are described and discussed, focusing on feasibility and biological plausibility of the methods. Specifically, we describe in detail the functioning and organization of the latest version of a 3D spatio-temporal SNN-based data machine framework called NeuCube, as well as it's SNN-related submodules. All described submodules are accompanied with formal algorithmic formulations. The architecture is highly relevant for the analysis and interpretation of various types of spatio-temporal brain data (STBD), like EEG, NIRS, fMRI, but we highlight some of the recent both STBD- and non-STBD-based applications. Finally, we summarise and discuss some open research problems that can be addressed in the future. These include, but are not limited to: application in the area of EEG-based BCI through transfer learning; application in the area of affective computing through the extension of the NeuCube framework which would allow for a biologically plausible SNN-based integration of central and peripheral nervous system measures. Matlab implementation of the NeuCube's SNN-related module is available for research and teaching purposes.

Keywords Artificial neural networks · Spiking neural networks · Spike encoding · Spike-timing dependent plasticity · Spatio-temporal brain data · NeuCube

✉ Marko Šarlija
marko.sarlija@fer.hr

Clarence Tan
cltan@aut.ac.nz

Nikola Kasabov
nkasabov@aut.ac.nz

[1] Knowledge Engineering and Discovery Research Institute, Auckland University of Technology, Private Bag 92006, Auckland 1010, New Zealand

[2] Faculty of Electrical Engineering and Computing, University of Zagreb, Unska 3, 10000 Zagreb, Croatia

1 Introduction

In the parlance of machine learning and artificial intelligence (AI), a neural network can be defined as a network of neurons that are able to perform computations and solve problems. Neural networks for learning as seen today, have come a long way since their discovery in the late 50s by Hubel and Wiesel [1] followed by the development of Neocognitron, a neural network composed of multiple layers, by Fukushima in the early 80s [2]. Depending on the type of neurons used, artificial neural networks (ANNs), as they are referred to, can be thought to belong to three generations.

First-generation ANNs are composed of neurons that compute a weighted sum of binary inputs and produce an output of 1 if the sum crosses a pre-defined threshold, else the output is zero (see Fig. 1a). These neurons are also known as perceptrons [3] or threshold gates. Mathematically, a perceptron can be expressed as:

$$y = \begin{cases} 0, \sum_j w_j x_j \leq \theta \\ 1, otherwise \end{cases} \tag{1}$$

where θ is a threshold parameter.

The second-generation ANNs are composed of artificial neurons, also known as Sigmoid neurons, which apply a nonlinear activation function f to the sum of weighted neuron inputs, as shown in Fig. 1b. The function f is a sigmoid function, which is smooth and differentiable. The primary motivation behind using such functions is to enable the application of backpropagation algorithms, that are based on error function gradient computation, to train the ANNs.

By stacking more than one layer of neurons, as shown in Fig. 1c, and applying backpropagation to learn multiple layers of representation, deep learning neural networks can be constructed. Deep-learning neural networks are today capable of solving problems in diverse areas ranging from speech recognition [4], visual recognition [5], pedestrian detection [6], recognition of traffic signs [7] to playing GO [8] and biomedical signal processing [9,10], and can in some cases outperform humans.

As impressive as this feat may be, the fact remains that biological neurons in humans and other animals still outperform ANNs in terms of energy and efficiency. Many deep learning algorithms rely on hundreds of graphical processing units (GPUs) and central processing (CPUs) to solve problems which a human brain can solve in a fraction of these resources in terms of energy. To give an example, it required 1 megawatt (MW) of energy to solve GO

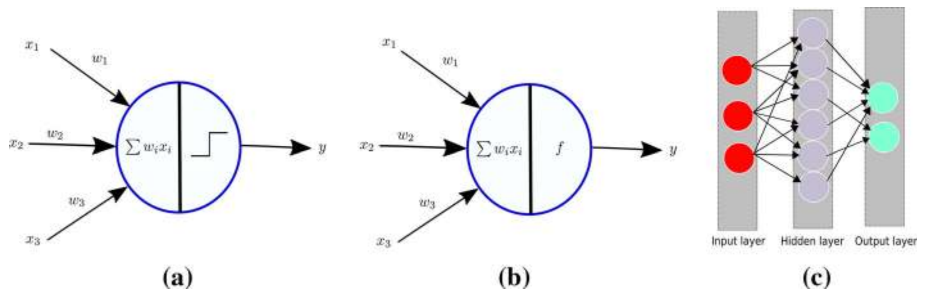


Fig. 1 Model of a perceptron that uses, **a** a step-function to give an output of 1 if the weighted sum of inputs cross a pre-defined threshold and **b** a sigmoid function to give a continuous output based on the weighted sum of continuous inputs. A neural network model with one hidden layer is shown in **c**

challenge using deep learning, whereas the best human players could achieve similar results in about 20 W [11], which is the power rating of a human brain.

Furthermore, deep learning networks are not biologically plausible. Although initially inspired by biological neurons, the neurons in deep-learning networks solve the learning problem in a fundamentally different manner. Bengio et al. in their recent work discuss problems that arise regarding the biological plausibility of backpropagation [12]. For instance, backpropagation algorithms are not biologically plausible as biological neurons also exhibit nonlinearity, whereas backpropagation algorithm is purely a linear operation. Another crucial issue is the biological implausibility of ANNs themselves, as biological neurons communicate with each other using discrete spikes known as action potentials and not by continuous values as seen in the implementation of deep learning algorithms with second-generation ANNs. These issues gave rise to the present or third generation of ANNs known as the spiking neural networks (SNNs) [13], which use spiking neurons as their computational unit. In the remainder of the paper, ANNs will be used to refer to second-generation ANNs. Accordantly with what is previously said, off-the-mainstream approaches in neural networks and machine learning for pattern recognition have been recently encouraged [14] in Neural Processing Letters, with an entire special issue being devoted to various off-the-mainstream fields in pattern recognition (associative memories, density-related algorithms, etc.). We believe that the SNNs, being the third-generation of ANNs, as an alternative or a complement to traditional second-generation ANNs, definitely deserve attention when considering off-the-mainstream approaches in neural processing.

The paper is organized as follows. Section 2 formally introduces SNNs and Sect. 3 briefly reviews some of the popular computational models used in SNNs. Sections 4 and 5 describe information processing and learning algorithms used in SNNs. Finally, Sect. 6 presents a recently developed 3D SNN architecture for mapping, learning and understanding of spatio-temporal brain data called NeuCube [15]. In the conclusion (Sect. 7) we discuss potential applications and challenges in SNNs/NeuCube, and encourage it's further use and development while not staying limited to just brain-data-specific applications.

2 Spiking Neural Networks

The human brain is composed of 10^{12} neurons and each neuron makes about 10,000 connections, known as synapses. The structure of a neuron can be divided into three basic parts: (1) dendrites, (2) cell body or soma, and (3) axons. The dendrites are nerve cell extensions that process input signals to a neuron via synapses and axon, the thin projection of a neuron, can be thought of as a long process that carries the output signal away from the neuron. The cell body or soma of a neuron contains the nucleus and cytoplasm, as with any other cell in the body.

In Fig. 2, what is known as a pre-synaptic neuron is carrying an action potential (AP) or spike through its axon, which causes the release of neurotransmitters at the synapse. The neurotransmitters come in two flavors—excitatory and inhibitory. If an excitatory neurotransmitter is released, positive ions are released into the dendrite of the post-synaptic neuron, causing an excitatory post-synaptic potential (EPSP). If an inhibitory neurotransmitter is released, negative ions flow into the dendrite of post-synaptic neuron causing an inhibitory post-synaptic potential (IPSP). At the soma of the post-synaptic neuron the IPSPs and EPSPs from all the pre-synaptic neurons are spatially and/or temporally summed. When this sum exceeds a threshold, the post-synaptic neuron fires an AP. Thus, the AP can be thought of as a "currency" with which neurons exchange information, and using discrete spikes like APs rather than continuous signals is what makes the brain energy-efficient. Spiking neural

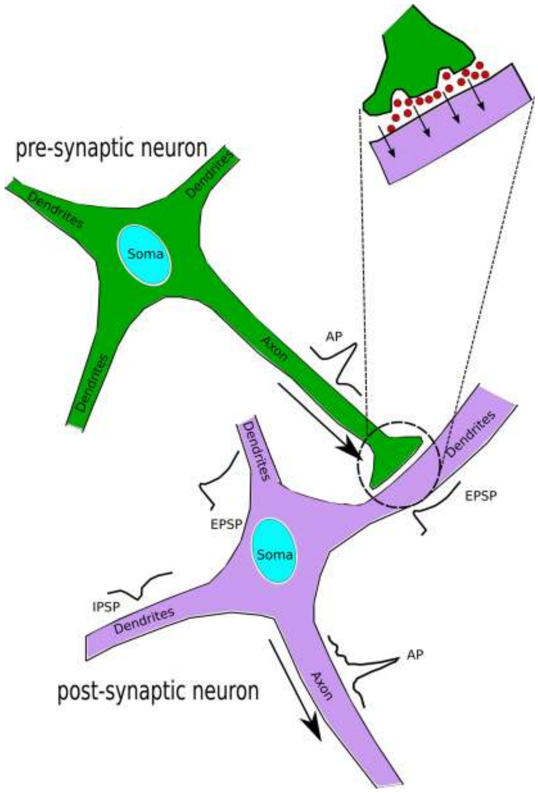


Fig. 2 Illustration of a pre-synaptic neuron (green) and a post-synaptic neuron (purple) connected through a synapse. Neurotransmitters (red circles) are released at the synapses of many dendrites of a post-synaptic neuron, giving rise to post-synaptic potentials, which are finally summed and a decision to send an action potential via the axon of post-synaptic neuron is made

networks (SNNs) are a special class of artificial neural networks (ANNs), also commonly referred to as the third generation of ANNs, where the neuronal units communicate using discrete spike sequences, as exemplified in Fig. 3. Analogous to a biological neuron, the inputs to a spiking neuron are discrete spikes, which are then combined to produce an output spike if a certain threshold is exceeded. Otherwise, the output is zero. Therefore, SNNs also contain temporal dynamics, which makes them suitable for real-time operation, making updates that are purely event- and data-driven, unlike the repeated and often redundant process of updating the weights in ANNs, which is the computational bottleneck for tasks that require real-time interaction with the environment.

3 Computational Models of SNNs

As mentioned previously, SNNs use spiking neurons as their main computational unit. Numerous mathematical models of a spiking neuron have been proposed in the literature [16]. Below we briefly review the three most widely used models—(1) Leaky integrate-and-fire (LIF), (2) Izhikevich model and (3) Spike response model (SRM).

3.1 Leaky Integrate-and-Fire Model

In this type of model, illustrated by Fig. 3 and first introduced by Lapicque [17,18] over a hundred years ago, the membrane is characterised by a resistance R and capacitance C. Let

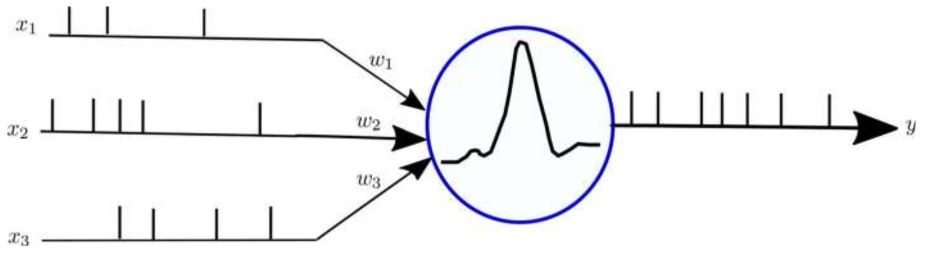


Fig. 3 Model of a LIF spiking neuron

$u(t)$ be the voltage across the membrane at time t and u_{rest} be the resting-state potential. Then, we have the following equation describing an RC-circuit:

$$i(t) = \frac{u(t) - u_{rest}}{R} + C\frac{du(t)}{dt} \tag{2}$$

where $i(t)$ is the membrane current. Rearranging the terms in the above equation, we can write:

$$\tau\frac{du(t)}{dt} = u(t) - u_{rest} + Ri(t) \tag{3}$$

where τ is the membrane time constant. A spike is generated at time t_f, when the membrane potential reaches the threshold, i.e., $u(t_f) \geq u_{thresh}$. After t_f, the membrane potential is reset to the resting state value u_{rest} and for $t > t_f$ the dynamics are again given by the equation above (3). To make this model biologically more plausible, an absolute refractory time δ_{ref} may be included such that the dynamics are restarted as per the Eq. 3 after $t_f + \delta_{ref}$ rather than immediately after t_f [19].

3.2 Izhikevich Model

The Izhikevich model [20] basically has two variables, u and v that describe membrane voltage and the recovery rate of a neuron. The evolution of the membrane voltage u is described by a pair of ordinary differential equations (ODEs):

$$\frac{du}{dt} = 0.04u^2 + 5u + 140 - v - i \tag{4}$$

$$\tau\frac{dv}{dt} = a(bu - v) \tag{5}$$

After initiating the action potential, the following resetting scheme is used:

$$u = cv = v + d \tag{6}$$

when $u \geq 30mV$, which is the peak of the spike [21]. The model parameters, a, b, c and d can be tuned to produce various neural dynamics [20,21]. Briefly,

1. The parameter a describes the time-scale of v.
2. The parameter b describes sensitivity of v to the (subthreshold) fluctuations in u.
3. The parameter c describes the resetting of u after the initiation of a spike, which is caused by fast high-threshold K^+ conductance.
4. The parameter c describes the resetting of v after the initiation of a spike, which is caused by slow high-threshold K^+ and Na^+ conductance.

Table 1 Comparison of different models of spiking neurons

Models	No. of variables	Complexity	Biologically plausible
Leaky integrate-and-fire	1	Very low	No
Izhikevich	2	Very low	No
SRM	1	Low	No
Hodgkin–Huxley	4	Very high	Yes
FitzHugh–Nagumo	2	Medium	No
Wilson	2	Medium	No
Moris-Lecar	3	High	Yes

3.3 SRM

The spike-response model (SRM) is based on kernel function $K(\cdot)$ and described by a single variable u_i, which is the membrane voltage of the ith neuron. To describe this model, let us assume that the neuron i is at a resting-state potential of 0 Volts. Incoming spikes from pre-synaptic neurons will affect $u_i(t)$ and after some time, $u_i(t)$ will return to its resting state value. If the net effect of all the incoming spikes is such that $u_i(t)$ crosses the threshold θ, then an output spike is triggered. Assuming that the neuron i has last fired its spike at time t_0, the dynamics of $u_i(t)$ is given by:

$$u_i(t) = \eta(t - t_i') + \sum_j w_{ij} \sum_f \epsilon_{ij} \left(t - t_i',\ t - t(f)_j\right) \int_0^{\propto} \kappa\left(t - t', s\right) I(t - s) ds \quad (7)$$

where the function $\eta(\cdot)$ describes the form of the action potential, i.e., the spike and I is the external driving current. The term $t_j^{(f)}$ describes the time of the input spike of neuron j and w_{ij} represents the synaptic weight between neurons i and j.

3.4 Other Models

In addition to the three widely used deterministic models in SNNs described above, several other models of spiking neurons have been proposed and these include Hodgkin–Huxley model (HH) [22], Wilson model [23], FitzHugh–Nagumo model [24], Hindmarsh–Rose model [25] and Morris–Lecar model [26] (see [21] for description). The Table 1 shows a comparison of various spiking models in terms of the number of variables, complexity and biophysical plausibility, which is adapted and modified from [21].

It is important to note that all widely used models discussed in this paper are deterministic, and as such might have limited applicability for solving more complex problems in the future. Spiking processes in biological neurons could also be viewed on as stochastic by nature. For example, spike time could also depend, besides on the deterministic input signals, on gene and protein expression [27], on physical connection properties [28], on probabilities of spikes being received at the synapses, etc. These facts motivated for new ways to enhance the current SNN models with probabilistic parameters, yielding a recent development of a probabilistic spiking neuron model (pSNM) [29], that may allow for a broader range of applications and understanding in the future, but exceeds the scope of this paper.

4 Information Processing in SNNs

The real world data and signals are analogue, continuous. Efficient and successful encoding of such inputs into discretely timed spike trains is the crucial and initial step of information processing in SNNs [30]. For this cause, we turn to prevailing neural coding theories in neuroscience.

Almost a century ago, Adrian and Zoltermann demonstrated that the firing rate of the stretch receptor in a muscle spindle of a frog increased with the strength of the stimulus [31], leading to the conclusion that firing rate is the primary "currency" of information exchange. This coding scheme is known as rate coding. Since this seminal study, there have been many other studies that have challenged this simplistic view of neural coding [32–34]. In particular, Thorpe [35] argued that since neural processing is so fast (for example visual processing is completed under 100 ms), it must rely on temporal coordination of spikes rather than the firing rate, which requires more time.

Still, one of the fundamental questions in neuroscience is how neurons encode and decode information through spikes [36]? Is it through rate coding or spike coding? This remains a matter of intense debate and below we briefly review some of the popular theories.

4.1 Rate Coding

The rate code model is one of the most commonly used models to describe information processing and can use different definitions for the firing rate. The simplest definition of the firing rate is the temporal average, i.e., the number of spikes divided by the corresponding time interval duration. One can also define the firing rate in the context of spatial average, where spikes from a population of neurons within a certain time-interval are averaged. For the third definition of firing rate, one can consider spikes as random events and the firing rate is given as the average over several trials, also known as spike density, of the same stimulus for a single neuron. Although it has been argued that the rate coding scheme based on spike density is biologically not very accurate [32,35], it is one of the most widely used/considered coding schemes.

4.2 Temporal Spike Coding

As opposed to rate coding, temporal coding encodes the information by employing the exact timing of individual spikes. Such approach to information processing in SNNs is biologically supported by evidence based on observation of different types of biological neurons [34], with first successful application in SNN-based supervised learning demonstrated in [37]. In the following paragraphs we discuss some of the commonly used coding schemes based on temporal spike coding.

4.2.1 Time-to-First-Spike

In this neural coding scheme, information is encoded in the time between the beginning of the stimulus and the appearance of the first spike (see Fig. 4). Such a scheme is ideal for fast processing of information as it requires only a few milliseconds after the appearance of a stimulus to convey a decision on a stimulus.

4.2.2 Rank Order Coding

In the rank order (RO) coding scheme [38], a population of neurons is considered and the information is encoded in the order in which the neurons spike, as shown in Fig. 5a. This

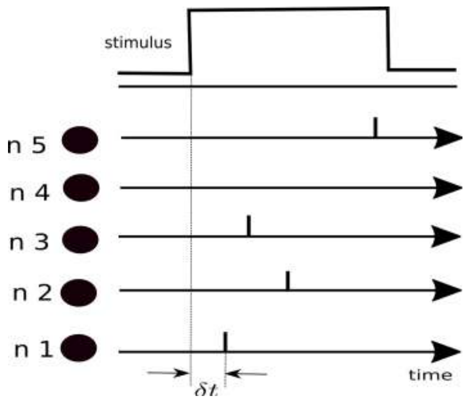


Fig. 4 Time-to-first-spike: neuron $n1$ is the first to spike at δt after the stimulus onset

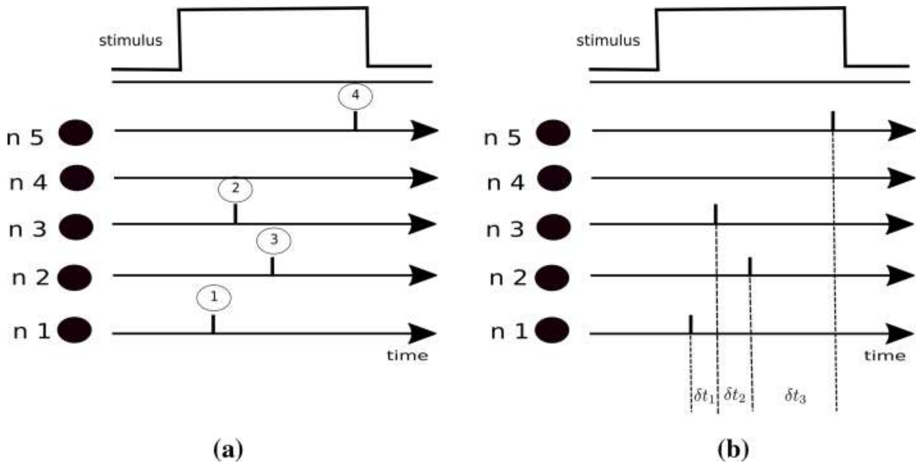


Fig. 5 Rank order coding (**a**): information is encoded in the order in which neurons spike. In this example the order is $n1$-$n3$-$n2$-$n5$. Latency coding (**b**): information is encoded in the spike timing δt_1, δt_2, δt_3 (neurons $n3$, $n2$ and $n5$) relative to $n1$ which spikes first

scheme assumes that each neuron in the population fires only once after the presentation of a stimulus.

4.2.3 Latency Coding

The latency coding scheme depends on the relative timing of spikes (see Fig. 5b), which also has implications on whether a synapse is potentiated or depressed. For instance, if the relative timing of spikes between the pre-synaptic and post-synaptic neuron is less than 20 ms, then the long-term potentiation (LTP) occurs. Otherwise, long term depression (LTD) occurs.

4.2.4 Phase Coding

In this coding scheme (see Fig. 6a) it is assumed that neurons spike at different phases with respect to some referent oscillation and thus the phase of the pulse concerning the referent oscillation codes the information. Experimental studies in rats have shown that the

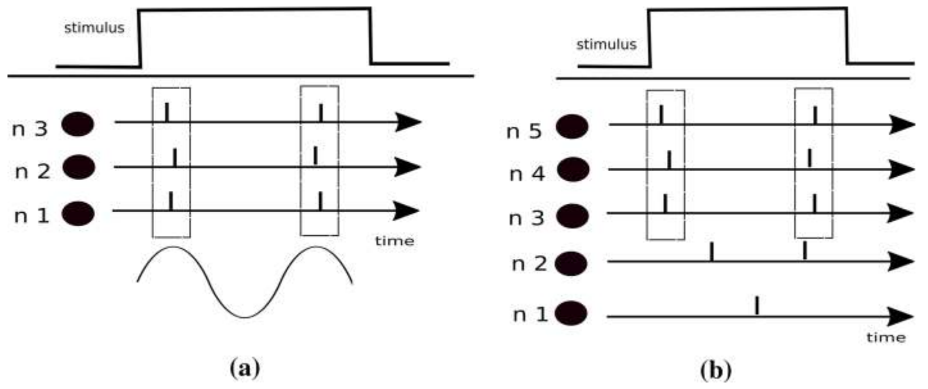


Fig. 6 Phase coding (**a**): the internal reference oscillation is depicted as a sinusoidal signal and the neurons $n1$, $n2$ and $n3$ spike at the same phase relative to this oscillation. Synchrony coding (**b**): neurons $n3$, $n4$ and $n5$ spike almost at the same time, as opposed to neurons $n1$ and $n2$ that are not synchronized in their spikes

information about the spatial location is encoded in the phase of a spike with respect to the hippocampal oscillation. Such oscillations, that are due to a population of neurons, are quite common in several areas of the brain [39].

4.2.5 Synchrony Coding

In synchrony coding model, neurons that spike synchronously are believed to encode the information. Based on several experimental studies, it has been shown that neurons tend to fire synchronously (see Fig. 6b) when they represent the different bits of information on the same object [39]. For example, it was shown that when the neurons in the visual cortex are activated with a single contour, they tend to synchronise their discharges, but not when the contour is moving in different directions [39].

4.3 Encoding Method Selection Criteria

Similar to the brain, where different spike encoding methods are applied in different sensory areas, such as retina, cochlea, olfactory, gustatory, etc., different encoding methods can be selected in the NeuCube architecture (see Sect. 6.2.1) for different types of continuous value input time-series data using different optimisation criteria. Such criteria can be:

(a) Minimizing the loss of information after decoding (recovering) the signal back from spike representation to continuous value. In [40] and [41] four types of encoding algorithms have been studied: Ben's Spiker algorithm (BSA); threshold-based encoding; step-forward (SF) encoding; moving-window (MW) encoding. BSA can follow smoothly changing signals if filter coefficients are scaled appropriately. SW encoding was most effective for all types of signals as it proved to be robust and easy to optimize. Signal-to-noise ratio (SNR) can be recommended as the error metric for parameter optimization. Different encoding algorithms can be suited to different EEG and fMRI data for example, the most commonly used one being SF [41]. Free software for finding the optimal encoding of time series for SNN is available.[1]

[1] https://github.com/KEDRI-AUT/snn-encoder-tools.

(b) Maximazing the classification/prediction accuracy at the output classification/prediction module of a NeuCube model [30]. Paper [30] also evaluates how much compression of the input data can be achieved through spike encoding without loss of classification accuracy at the output of an SNN system. It was demonstrated on several benchmark data that input information passed to a NeuCube classifier can be compressed by orders of magnitude without compromising the outcome.

5 Learning in SNNs

The learning methods for SNNs fall into two categories: (1) rate-based learning and (2) spike-based learning. This naturally arises from the previous section (Sect. 4), where we have divided information processing approaches in SNNs into rate-based and spike-based. The following Sects. 5.1 and 5.2 provide descriptions of the two learning categories.

5.1 Rate-Based Learning

Training deep-SNNs directly using backpropagation is not possible as gradients cannot be computed for spike inputs. Thus, an indirect approach of training an ANN with backpropagation and then converting it into an equivalent SNN by relating the activations of ANN units to firing rates of spiking neurons is used. The relation between the transfer function (i.e., the input-to-output relationship) of a spiking neuron and the activation of the rectified linear unit (ReLU) of ANNs, have been thoroughly described in [42,43]. Given a network of L layers, with weights $W_l, l \in 1, \ldots, L$ connecting layer $l-1$ to l with b_i^l being bias for neuron i in layer l, and assuming that the number of units in each layer is Nl, the activation in ANN is given by:

$$a_i^l := max\left(0, \sum_{j=1}^{N_l-1} W_{ij}^l a_j^{l-1} + b_i^l\right) \tag{8}$$

with initial condition $a^0 = x$ as input, which is assumed to be normalized so that $x_i \in [0, 1]$. In case of a SNN, the membrane potential $u_i^l(t)$, is driven by the input current $z_i^l(t)$:

$$z_i^l := \tau\left(\sum_{j=1}^{N_l-1} W_{ij}^l \Theta_j^{l-1} + b_i^l\right) \tag{9}$$

where $\Theta_j^{l-1} = \Theta(u_i^l(t-1) + z_i^l(t) - \theta)$ is a step function. The spiking neuron integrates the inputs $z_i^l(t)$ until the membrane potential $u_i^l(t)$ exceeds the threshold θ.

Given that an input pattern is presented with time step dt, the maximum firing rate is constrained by $1/dt$ and the firing rate is given by $r_i^l(T) = \sum_{t=1}^{T} \Theta_{t,i}^l / T$ which is simply the number of spikes generated divided by the total time the input is presented. The ANN-to-SNN conversion proceeds such that the firing rates r_i^l correlate with activations of ReLU units, a_i^l, such that $r_i^l(T) \rightarrow a_i^l/dt$ [44,45]. In the following paragraph, we briefly review some of the work that has employed ANN-to-SNN conversion.

In a study by Merolla et al. [46], a two-layer (484 visible units and 256 hidden units) restricted Boltzmann machine (RBM) was trained on handwritten digits from MNIST database and the weights were learned using contrastive divergence algorithm, achieving an accuracy of 94% when tested on out-of-sample data. After learning the weights, the 256 hidden units were converted to integrate and fire neurons and two axons (for excitatory and inhibitory synapse) represented each of the 484 visible unit. Following a thresholding procedure, continuous weight matrices were converted to binary matrices and spiking RBM neurons, achieving an accuracy of 89%, which was less than what was achieved by RBM before conversion to spiking version. Using Siegert neurons as computational units, O'Connor et al. [47] trained each RBM in a time-stepped mode, using CD algorithm for training, with modification to encourage sparse and receptive fields in the hidden layer. For the conversion of RBMs to equivalent spiking network with LIF neurons, the firing rates of the Siegert neurons in RBMs are normalised and converted to activation probabilities. Hunsberger and Eliasmith [48] proposed to transfer a static convolutional neural network (CNN), modified from [49] to spiking neurons. They trained the static CNN by using a smoothed version of LIF rate equation, so that the gradient during backpropagation remains bounded. Furthermore, in order to simulate variability in filtered spike trains the static CNNs were trained with noise added to the output of each neuron for each training example. The static CNN was then converted to an SNN by replacing back the soft-LIF model with the normal LIF spiking model and removing the noise added during training. Esser et al. [50] used a training network sharing the same network topology as the deployment network, TrueNorth neurosynaptic chip [51]. The training network takes data (input and output of neurons, synaptic connections) in a continuous format, with the constraint that the values are within the range [0, 1], which can be obtained by re-scaling the pixels of input images. These continuous values can then be interpreted as a probability of a spike occurring or not when mapping to the corresponding hardware. The network was trained using backpropagation methodology and the probability whether a neuron will spike or not was derived using a complementary cumulative distribution of a Gaussian function. The performance of SNNs obtained after conversion of ANNs on benchmark tasks is still lower than what is achieved by state-of-the-art ANNs and to overcome this, Diehl et al. [44] used ReLUs and mapped the weights from the ReLU network to network with integrate-and-fire units. Furthermore, bias was fixed to zero throughout the training with backpropagation. Finally, the key step of weight normalisation was employed, which helped in reducing the errors due to replacing ReLUs with IF neurons.

The major drawback of rate-based learning, as an indirect learning approach in SNNs is the fact that, since traditional rate-coding is used in ANNs, processing times are longer and many spikes are needed to encode the input. Finally, we summarise some of the main challenges in rate-based learning:

1. *Negative values* Negative values can arise when using ANNs like CNNs for several reasons:

   - The sigmoid function used in CNNs can result in negative outputs.
   - Since weights and biases can both be negative, the output which is given as the sum of weights and biases can be negative.
   - Preprocessing step of input image or pattern can produce negative values.

   Although neurons with negative values can be represented as inhibitory neurons in SNNs, this would require a doubling of the neurons. An undesirable consequence of this would be a direct increase in hardware resources and power consumption. Also, adding inhibitory neurons to SNNs can lead to complicated interconnections.
2. *Bias* Representing biases in spiking networks is not a straightforward task, and the biases in each layer can be positive or negative.

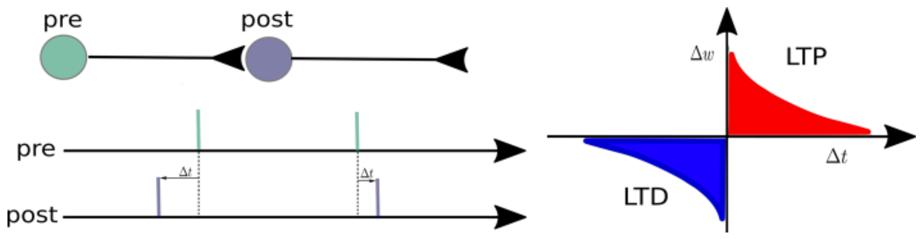


Fig. 7 The concept of STDP: the function on the right shows the change of the synaptic connection weight Δw as a function of the time difference Δt between a pre- and a post-synaptic spike arrival time. Positive Δt (pre-synaptic spike before the post-synaptic) leads to Long-Term Potentiation (LTP) of the synapse, with negative Δt (post-synaptic spike before the pre-synaptic) leads to Long-Term Depression (LTD) of the same synapse. Spike timings within the two pre- and post-synaptic spike pairings are shown on the left: first pairing results with a negative Δt value and the second pairing with a positive Δt value

3. *Max-pooling* Max-pooling in SNNs requires more neurons as we need a two-layer neural network with lateral inhibition to implement spatial max-pooling analogous to what is done in convolutional neural networks.

5.2 Spike-Based Learning

Spike-based learning is based on spike-timing dependent plasticity (STDP), which is believed to be the primary form of synaptic change in neurons [52] and involved in learning and memory formation in mammalian visual cortices. Bi and Poo provided the first evidence based on biological observations, on the importance of relative timing of the pre- and post-synaptic spikes in modulating the synaptic efficacy [53]. Interestingly, one of the first spike-timing-dependent learning algorithms was described by Gerstner already in 1993, where it was shown using SRM neurons that coding by spatio-temporal spike patterns was beneficial over mean firing rate [54]. Over the last few decades, evidence of STDP as a learning mechanism has been established in both in vivo and in vitro studies [55–57] and STDP has been widely used as learning algorithm in the field of AI. In this section we briefly review the concept of STDP and some works using STDP learning rule for training SNNs.

According to the STDP learning rule [58], the strength of the synaptic weight is proportional to the degree of correlation between the spikes in the pre-synaptic and post-synaptic neuron. As shown in the Fig. 7, if a spike occurs in the post-synaptic neuron shortly (under 20 ms) after the occurrence of a spike in the pre-synaptic neuron, the synaptic connection is strengthened, and this is known as long-term potentiation (LTP). On the contrary, if a post-synaptic neuron fires before the pre-synaptic neuron, long-term depression (LTD) occurs which decreases the synaptic weight. The STDP rule describing the change in weights Δw is generally given as:

$$\Delta w = \begin{cases} Ae^{\frac{|t_{pre}-t_{post}|}{\tau}}, t_{pre} - t_{post} < 0, A > 0 \\ Be^{\frac{|t_{pre}-t_{post}|}{\tau}}, t_{pre} - t_{post} > 0, B < 0 \end{cases} \tag{10}$$

where A and B are the learning rates for LTP and LTD, with τ being the time constant. Generally, we can say that the strength of a synapse is increased at the moment of post-synaptic firing by an amount that depends on the value of the trace left by the pre-synaptic spike. Similarly, the weight is depressed at the moment of pre-synaptic spikes by an amount proportional to the trace left by previous post-synaptic spikes.

It has been demonstrated that by selecting inputs which have low time jitter, STDP increases the post-synaptic spike time precision [19]. Several strategies for training SNNs based on STDP learning rule have been proposed. Kheradpisheh et al. [59] proposed an STDP-based learning approach for SNNs, whose architecture consisted of three convolutional layers comprising nonleaky integrate-and-fire neurons equipped with STDP and three pooling layers. The first layer uses a difference of Gaussian (DoG) filter which encodes the contrast strengths in the input image to latencies, leading to earlier firing of neurons for higher contrasts and vice-versa, thus implementing the rank-order coding scheme (see Sect. 4.2.2). The neurons in convolutional layers integrate these spikes and fire if a threshold is reached, whereas the pooling layers provide translational invariance using nonlinear max pooling. Their results showed that SNNs achieved accuracies of 99.1% on Caltech face/motorbike task and 98.4% on the MNIST dataset. On the ETH-80 datasets, which consists of images of various objects taken from different viewpoints, an accuracy of 82.8% was achieved. Diehl and Cook [44] presented an SNN with two-layers for digit recognition. In the first layer, the intensity of each pixel of the input image is converted to a Poisson-spike whose firing rate is proportional to the intensity. The second layer comprised excitatory and as many inhibitory neurons, with excitatory neurons being connected to inhibitory neurons in a one-to-one fashion, whereas each inhibitory neuron is connected to all the excitatory neurons except the one it received an input from. This arrangement provides lateral inhibition with competition among excitatory neurons. The changes in weights are given by different STDP-based learning rules that make use of exponential weight dependence. The demonstrated an accuracy of 95%, achieved on the MNIST benchmark. Tavanaei et al. [60] trained a SNN consisting of Izhikevich neurons (see Sect. 3.2) for isolated spoken digit reconstruction. The training is performed using the STDP learning rule which is a mixture of Hebbian and anti-Hebbian STDP in a supervised fashion, using a teacher signal at the output that determines the type of STDP to be used based on whether the desired output neurons spike (Hebbian) or not (anti-Hebbian). When tested on the Aurora dataset [61], the overall classification accuracy was 90.8% without noise and 70.2% under 10dB noise.

6 NeuCube

6.1 From Evolving Connectionist Systems to Dynamic Evolving SNNs

Evolving connectionist systems (ECOS) are generally modular systems that evolve both their structure and functionality from incoming information/data, in a way that is continuous, self-organised, online, adaptive and interactive [62,63]. Translating the principles of ECOS to SNNs, neurons are created (evolved) incrementally clustering the input data in either supervised or unsupervised way. This paradigm results in what we now call evolving spiking neural networks (eSNN) [63,64] that can learn patterns within the data by creating (evolving) and merging (connecting) spiking neurons. The dynamic eSNN (deSNN), introduced in [65], combines rank-order (see Sect. 4.2.2) and temporal (e.g. STDP, see Sect. 5.2) learning rules. All ECOS-type systems, from simple ones to eSNN and deSNN, are generally guided by the same main principles (i.e. they have evolving structures; they learn and partition the problem space locally, allowing for a faster adaptation; they learn in a continuous, on-line, incremental way [65]). It has been argued that SNNs are in general the most convenient way towards the creation of an integrative computational framework for processing various spatio- and spectro-temporal brain data (STBD) [15], such as EEG, fMRI, DTI, MEG, and NIRS. This is mainly because SNNs in their implementation follow the same computational principle that generates STBD: spiking information processing, as described in the previous sections of the paper. In the following subsection, we describe the latest version of an implemented SNN architecture, called NeuCube, first introduced in [15], with its specific input encoding, STDP learning and deSNN-based output representation. NeuCube is mainly designed for the creation of models that map, learn and help in the understanding of STBD. However, we will discuss some current problems in the area of brain-computer interface (BCI) and affective computing that could be addressed through an SNN-based architecture like NeuCube.

6.2 SNN Implementation—NeuCube Framework

General principles of the NeuCube architecture were first presented in [66], followed by a detailed description of the entire architecture in [15]. In this paper we focus on the latest version of the SNN-based part of the NeuCube architecture, depicted in Fig. 8. It consists of the following modules:

- Input module: implements input data encoding
- Representation module: 3D SNN reservoir module, i.e. the SNN cube (SNNc) module
- Output module: implements the output function, i.e. classification

The gene regulatory network (GRN) module, parameter optimisation module, and visualisation and knowledge extraction module are all optional and are left out for the purposes of this discussion. The MATLAB-based implementation of the architecture is shown in Fig. 9.

6.2.1 Input Module: Encoding

The goal of spike encoding (background described in Sect. 4) is the transformation of continuous input timeseries into sparse spike trains of exciting (+1) and inhibiting (−1) spikes. The thresholding representation (TR) algorithm is the most commonly used spike encoding algorithm. It is also called the Address Event Representation (AER) method (as in [67]). The method is based on thresholding the rate of change [x' in Eq. (14)] of the same input variable over time and is suitable when the input data is a stream (which is mostly the case).

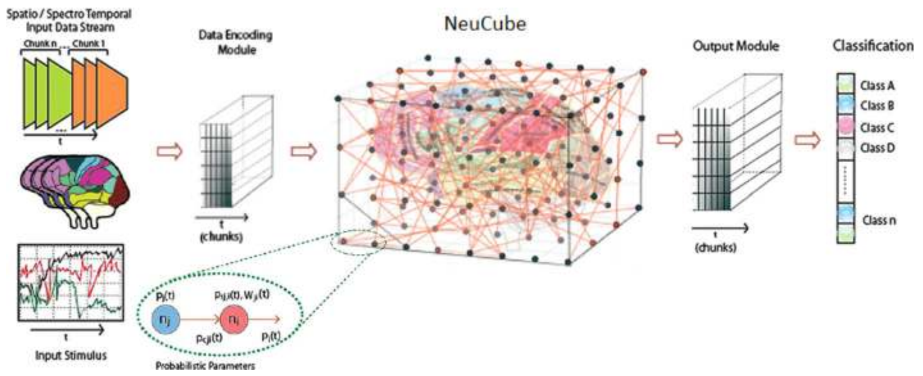


Fig. 8 A schematic representation of the SNN-based NeuCube architecture, consisting of: input data encoding module; 3D SNNc module; output function module (e.g. for classification or prediction). The gene regulatory networks (GRN) module is optional and is left out for the purposes of this paper. Adapted from [15]

The NeuCube implementation of the TR spike encoding algorithm is based on the variable threshold value that is calculated for each of the input data channels. The point of this variable threshold array is for it to be suitable concerning the specific signal dynamics, that could vary between different applications or even different input channels of the same application. For each of the input channels, the variable threshold is calculated based on one scalar input parameter (α_{TR}) in the following way:

$$VT(k) = \frac{1}{N}\sum_{i=1}^{N}(\mu + \sigma \cdot \alpha_{TR}) \tag{11}$$

$$\mu = \frac{1}{T}\sum_{j=1}^{L-1} x' \tag{12}$$

$$\sigma = \sqrt{\frac{\sum_{j=1}^{T-1}(x' - \mu)^2}{T-2}} \tag{13}$$

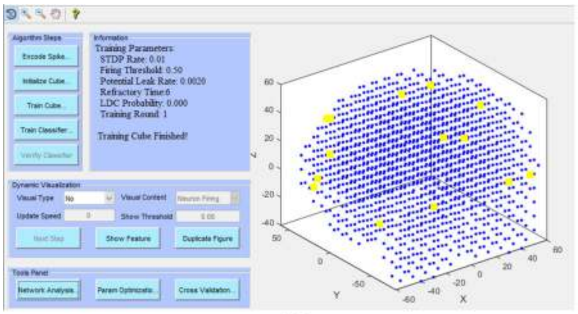
$$x' = |X(j+1, k, i) - X(j, k, i)| \tag{14}$$

where N is the number of samples, T is the signal length (number of time points per data sample), and k goes from 1 to the number of channels N_{input}. X is a $T \times N_{input} \times N$ data matrix, α_{TR} is the spike threshold parameter, and VT is the resulting variable threshold array. Equation (11) represents the threshold value for the kth input channel.

Once we have determined the threshold values for each of the channels (according to Eq. (11)), every signal in the dataset is transformed in the following way:

- Exciting spike train is a sparse signal of the same length as the input signal, with a value of 1 at each time step where the positive signal difference (rate of change) exceeds the variable threshold.
- Inhibiting spike train is a sparse signal of the same length as the input signal, with a value of 1 at each time step where the negative signal difference (rate of change) exceeds the variable threshold.
- The two spike trains are subtracted, to get a spike train of values 1, 0 and -1. These sparse representations are then used as inputs to the spatially located neurons from the SNNc module (yellow neurons in Fig. 9a).

Different spike encoding algorithms have different characteristics when representing the input data. Therefore, besides the described TR spike encoding algorithm, one can choose to use the BSA [68], MW algorithm, or SF algorithm, as these are all implemented in the current version of the NeuCube [69], but detailed explanations are here omitted. Our general recommendation when choosing a proper spike encoding algorithm would be to figure out what information the spike trains shall carry for the original signals. After that, the underlying patterns in the resulting spike trains will have higher interpretability and will hopefully yield a more successful SNNc representation. On the other hand, poor/inadequate spike encoding algorithm choice will essentially mean loss of useful information and can introduce information noise. Hence, spike encoding, being the first link in the processing chain, is also a highly important one. Algorithm 2 (appendix) sums up Eqs. 11–13, with the rest of the spike encoding procedure in a compact algorithmic form.


Training Parameters:
STDP Rate: 0.01
Firing Threshold: 0.50
Potential Leak Rate: 0.0020
Refractory Time:6
LDC Probability: 0.000
Training Round: 1

Training Cube Finished!

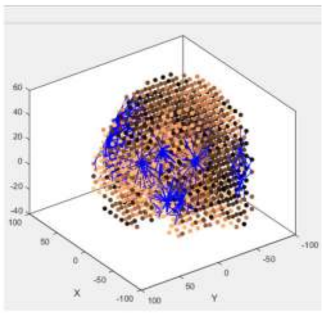


Fig. 9 Current version of the MATLAB-based software implementation of the NeuCube architecture: an exemplary classification task with 3-class EEG data. **a** Shows the brain/cube (SNNc) neuron coordinates, with the information regarding the unsupervised STDP training phase. **b** Shows the connectivity of the SNNc after training (positive connections are represented in blue and negative in red; a brighter neuron has more connections)

6.2.2 3D SNNc Initialization

The 3D SNN reservoir module, also called the SNN cube (SNNc), is essentially a group of spatially located spiking neurons, usually with known coordinates of the input neurons. The 3D structure of the neurons and their connections require a visualisation that goes beyond a traditional 2D connectivity/weight matrix. Figure 9 shows a 3D SNNc of $N_{cube} = 1471$ brain-mapped spiking neurons, whose coordinates are based on the Talairach Atlas, a human brain template [70]. SNNc structure (spiking neuron coordinates) can be defined automatically (by specifying how many equally-spaced neurons shall be created for x, y, and z coordinates; resulting in a cuboid shape), or by loading the coordinates from a file. In our case (coordinates based on the Talairach Atlas), the resulting shape was brain-like, but generally, the resulting shape of the SNNc can be arbitrary. In principle, either Talairach or MNI [71] brain templates can be used to initialize a brain-like SNNc structure before variable mapping and learning. In some cases, the first can be suitable for cortical data, such as EEG, while the second could be better used for whole brain data such as fMRI. Both methods are used and compared on fMRI data in [72] and [73].

The spike trains, obtained after encoding data using the TR algorithm (Sect. 6.2.1), are entered into the SNNc via corresponding brain-mapped input neurons. The number of input neurons N_{input} is usually defined by the number of channels arising from the loaded dataset. Coordinates of the input neurons (input mapping locations) are defined either manually, automatically (by graph matching), or from a file. These should correspond to the locations of the origins where the data channels were collected (if such locations exist). In Fig. 9a, the yellow input neuron coordinates naturally correspond to specific EEG electrode locations. These input neuron coordinates need to be a subset of the SNNc coordinates (usually $N_{input} \ll N_{cube}$).

L2 norm is calculated to get distances between pairs of neurons resulting in a $N_{cube} \times N_{cube}$ matrix of distances L_{dist}. The connections between the neurons in the SNNc are initialised using the small-world connectivity (SWC) approach, where a radius is defined as a parameter for connecting neurons within this radius (SWR). This results in an SNNc of sparsely connected neurons. Another parameter, LDC (long distance connectivity), can be used to initialise connections beyond the SWR. Generally, the LDC probability parameter would represent the probability of establishing a connection between two neurons with $L_{dist\ (ij)} > SWR$. Initially, all connections $C_{(ij)}$ between all neurons in the cube are set to 1. The connection flag between two neurons is set to zero (disconnected) if $L_{dist\ (ij)} > SWR$. A connection between a neuron i and a neuron j means that i is a pre-synaptic neuron in that connection, and j is post-synaptic. In a situation where a connection is 'bidirectional', we make a random choice leaving only one of the two. After connection initialization, 80% of the weights (matrix W) are expected to be positive and 20% are expected to be negative:

$$W_{(ij)} = sgn(rand\,(1) - 0.2) \cdot \ rand\,(1) \cdot \frac{1}{L_{dist\ (ij)}} \tag{15}$$

$rand\,(1)$ generates pseudorandom values drawn from the standard uniform distribution on the open interval (0, 1). It is also important to note that all input neurons can only be pre-synaptic neurons. Algorithm 3 (Appendix) sums up the SNNc initialization procedure described in this subsection.

6.2.3 3D SNNc STDP-Based Training

After the input data encoding (Sect. 6.2.1), and connection and weight initialization (Sect. 6.2.2), the SNNc is trained in an unsupervised manner using the STDP learning rule (introduced in Sect. 5.2), based on the training data. This step was described in a compact algorithmic form in [74]. Here we provide a more detailed, precise description of the NeuCube's implementation of STDP learning. The algorithm relies on the following parameters:

- D (potential leak rate): the rate of passive potential degradation through inactivity
- R (refractory time): determining a period of resting between spikes
- η (STDP rate): learning rate, used for weight updating
- β (firing threshold): potential threshold for generating a spike
- N_{iter}: number of training iterations (passes through the training data)

A parameter that determines the LDC probability (explained in Sect. 6.2.2) is here omitted, as it is not crucial for the understanding of NeuCube's STDP implementation.

At each timestamp (considering there are T timestamps for each datasample), N_{input} spike states are fed to the corresponding input neurons of the SNNc, and potential propagations are calculated. Let us consider the (i, j) neuron pair in which i is the pre-synaptic and j is the post-synaptic neuron. If a pre-synaptic neuron i fires, and neuron j is not in refractory time:

$$P_j(t) \ = \ P_j(t-1) \ + \ W_{(i,j)} \tag{16}$$

If at any time t, any neuron k exceeds the firing threshold potential β, it fires and its potential $P_k\,(t)$ is reset to 0. Its refractory counter R_k is set to R (refractory time). If at any time t, any neuron k did not exceed the firing threshold potential β, its potential is reduced: $P_k(t) \ = \ P_k(t-1) \ - \ D$, and refractory counter updates: $R_k = R_k - 1$.

Following the general STDP rule (Sect. 5.2), if a neuron i as pre-synaptic fires at time t and post-synaptic j has fired last at t_j^f, the connection weight is updated:

$$W_{(i,j)} \ = \ W_{(i,j)} \ - \ \eta/(t \ - \ t_j^f \ + \ 1) \tag{17}$$

On the contrary, if a post-synaptic neuron j fires at time t and a pre-synaptic neuron i has fired last at t_i^f:

$$W_{(i,j)} \ = \ W_{(i,j)} + \ \eta/(t \ - \ t_i^f \ + \ 1) \tag{18}$$

At the beginning of each new training iteration n_{iter} (of N_{iter}) the learning rate η is adapted to $\frac{\eta}{\sqrt{n_{iter}}}$. It is important to note that all times considered in the learning algorithm are discrete (meaning t, t_j^f, t_i^f, R_k and R are all integers). Algorithm 4 (appendix) sums up NeuCube's unsupervised STDP-based learning procedure described in this subsection.

6.2.4 deSNN Representation and Supervised Learning

After the unsupervised training, described in the previous section, the SNNc connectivity (Fig. 9b) can be analysed and observed, hopefully yielding better interpretability of the data when compared to traditional data processing and learning methods. This functionality additionally allows to identify differences in the activity of various brain regions in case of STBD or to identify input channel interactions in cases of other types of input data. Furthermore, we end up with a trained network that will, once we feed it with a specific input spike sequence, produce an array of spatio-temporal neuron firing events that are a result of both the input spike sequence and specific neuron connections that have emerged from the training data. This array of spatio-temporal events forms a pattern that can be encoded into a representation that is generally suitable as input to any traditional supervised learning algorithm (i.e. SVM, ANN, kNN).

The same data that was used for unsupervised STDP-based training is propagated again through the trained SNNc and a supervised model is trained to classify the spatio-temporal spiking pattern left in the SNNc into predefined classes. The output module from Fig. 8 represents the step where the SNNc spatio-temporal spiking pattern is transformed into a representation (or output neurons). The deSNN representation learning used in the NeuCube version presented in this paper is based on the RO learning rule (principles explained in Sect. 4.2.2).

The algorithm parameters are:

- α (*mod*): important for weight update on first spike occurrence
- d (*drift*): used for weight update on subsequent spikes

Propagating a single training sample through the trained SNNc network results with a sparse $T \times N_{total}$ neuron activation matrix representing the trajectories of all neurons spiking states through the time T (which is, as already described, a dataset-specific datasample time length). We form a $1 \times N_{total}$ array of output weights W_o, that will practically represent the output representation. Initially, all weights are zeros. These weights are updated by going through the $T \times N_{total}$ activation matrix for all neurons, at each timestamp. If a neuron i fires for the first time we set its weight to:

$$W_{o\ (i)} = \alpha^{order} \tag{19}$$

with $order$ being the firing counter. We start at $order = 0$ and each time a neuron fires for the first time, the counter is incremented $order = order + 1$. Otherwise, if a neuron fires again, the weight is updated by the rule:

$$W_{o\ (i)} = W_{o\ (i)} + d. \tag{20}$$

and if it does not fire, the weight is updated by the rule:

$$W_{o\ (i)} = W_{o\ (i)} - d. \tag{21}$$

To practically interpret the resulting representation array: W_o is a relatively sparse representation where the neurons that fired among the first, and most often afterwards, contribute most to the final position of a data sample in the $1 \times N_{total}$ deSNN representation space.

Therefore, the input neuron activations will contribute most significantly to the final representation for $\alpha < 1$.

The described algorithm provides a representation for each datasample in the dataset, including the validation/test data as well, although the SNNc connectivity is learned from just the training data in an unsupervised manner. With the $1 \times N_{total}$ representation, we can further address the problem via classical supervised learning, corresponding to the final module in the NeuCube schematic (Fig. 8). The method used for this part of the algorithm could be chosen arbitrarily, from one of the many available and widely used supervised learning algorithms. The current MATLAB-based NeuCube employs a version of the kNN (k-nearest neighbours) algorithm, but a detailed explanation of the classification module is here avoided as it exceeds the SNN nature of the paper. Algorithm 5 (Appendix) sums up NeuCube's deSNN representation algorithm that is described in this section. A compact summary of the entire NeuCube-based supervised learning chain is bellow in Algorithm 1, where we integrate all previous Sects. (6.2.1, 6.2.2, 6.2.3, 6.2.4) and accompanying Algorithms (2, 3, 4 and 5).

Algorithm 1 NeuCube-based supervised learning

```
Require: $X_{train} \in \mathbb{R}^{T \times N_{input}}$, $X_{test} \in \mathbb{R}^{T \times N_{input}}$, $Y_{train} \in classes$, $Y_{test} \in classes$
    {$hyperparameters := encoding\ pars.,\ SNNc\ struct.\ pars.,\ STDP\ pars.,\ \hat{Y_{test}} := f_{supervised}(X_{train}, Y_{train}, X_{test})$}
Ensure: $\hat{Y_{test}} \in classes$
1: $N_{train} \leftarrow \#(X_{train})$ {number of samples in the train dataset}
2: $N_{test} \leftarrow \#(X_{test})$ {number of samples in the test dataset}
3: $S_{train} \leftarrow f_{encode}(X_{train}, encoding\ pars.)$ {see algorithm 2 and Sect. 6.2.1}
4: $S_{test} \leftarrow f_{encode}(X_{test}, encoding\ pars.)$
5: $W \leftarrow f_{initialize}(SNNc\ struct.\ pars.)$ {see algorithm 3 and Sect. 6.2.2}
6: $W \leftarrow f_{STDP}(W, S_{train},\ STDP\ pars.)$ {see algorithm 4 and Sect. 6.2.3}
7: $\widetilde{W}_{train} \leftarrow f_{deSNN}(W, S_{train})$ {see algorithm 5 and Sect. 6.2.4}
8: $\widetilde{W}_{test} \leftarrow f_{deSNN}(W, S_{test})$
9: $\hat{Y_{test}} \leftarrow f_{supervised}(\widetilde{W}_{train}, Y_{train}, \widetilde{W}_{test})$ {arbitrary supervised learning algorithm like kNN gives}
10: performance evaluation by comparing $Y_{test}$ and $\hat{Y_{test}}$
```

7 Conclusion and Future Work

In this paper, we describe the development and discuss implementational aspects of spiking neural networks. The goal is to promote the use of SNN (as the third generation of ANN) and more specifically—the NeuCube architecture, as an off-the-mainstream approach in neural network processing. Comparative advantages of SNNs are clear, like computational speed, power consumption and biological plausibility. However, technical challenges continue to exist. As there is not yet a robust information theory supporting the design and implementation of SNNs [69], choice of the network structure (including input neuron locations) and other hyperparameters for each specific application is still based on heuristic measures and expert knowledge.

As argued in previous works [15,69], NeuCube's 3D SNN structure allows for the integrative modelling of various STBD, thus opening new opportunities for a better understanding and interpretation of STBD. The feasibility of using NeuCube in solving STBD-based tasks was already demonstrated in various domains, like EEG-based brain-machine interface for limb movement [75] or EEG-based classification of activities of daily living (ADL) in neurorehabilitation [76]. The network structure is relatively intuitive when it comes to STBD-based problems (brain-like structure shown in Fig. 9), but the real challenge is designing the

SNN structure for non-STDB-based classification and pattern recognition tasks. Successful examples of such efforts exist, like multi-variable-based personalised early stroke prediction [77] or video-based age classification [69], but still warrant for further exploration.

Finally, we summarise some open research problems in SNN and NeuCube applications that we find most promising, some of them already raised in [41]:

1. Integrating into one SNN model STBD multimodal data, such as EEG [78], audio-visual [79], fMRI and DTI [80].
2. Integrating into one SNN model heterogeneous data, such as quantum-, molecular-, brain-, physiological-, environmental information [41].
3. Defining optimal depth and length in both space and time of the knowledge representation extracted from a trained SNN [41].
4. Using SNN for the development of new information theories, such as information compression [30].
5. Human-machine and machine-machine transfer learning based on a common structural template for STBD, such as the Talairach Atlas [70] or MNI [71].
6. Learning long spatio-temporal patterns in the context of associative memory.
7. Developing methods for self-optimisation of learning in SNN: *learning to learn*.
8. Building societies of SNN machines for distributed deep learning and transfer learning.
9. Towards a symbiosis between humans and SNN machines.

Some of the research problems from the list above have been addressed previously (i.e. 1, 2, 3 and 4), but nevertheless, remain open. On the other hand, some will most probably remain unaddressed in the near future (i.e. 7, 8 and 9). Spiking neural networks and the NeuCube framework are currently at the point where a vast expansion of work in points 1–5 is expected. Some potential applications that we find most promising are in the area of sleep stage classification where vast amounts of data-intensive laboratory-based polysomnography (PSG) recordings exist [81], thus allowing for the possibility of having a pre-trained SNNc for transfer learning EEG-based BCI applications (relates to the research problem 5). Another highly challenging, but promising application is the area of affective computing where semi-STBD-based datasets are available [82,83]. The primary challenge here would be the integration of pure STBD (EEG), which reflects the central nervous system activity with the peripheral measures of autonomic activity like (but not limited to) electrocardiography (ECG) and electrodermal activity (EDA) into a single SNN-based structure (relates to research problems 1 and 2). The question is can these peripheral measures of autonomic nervous system (ANS) activity, be observed as activity within the limbic structures of the brain (i.e. amygdala, hippocampus, thalamus, hypothalamus), and as such be simultaneously integrated with STBD data like EEG and NIRS into a single specifically designed SNN structure? This would also open questions on how to efficiently encode the peripheral physiological data into spikes suitable for simultaneous processing with spike encoded STBD.

The version of NeuCube described in this paper is the so-called Module M1 (official NeuCube modules range from M1-M4 [69]) for research and teaching purposes, and can be found free of charge at http://www.kedri.aut.ac.nz/neucube. For commercial use or access to the full set of modules, please contact the corresponding author directly or via this web page. The NeuCube is PCT patent protected.

A Appendix: Algorithms

Algorithm 2 NeuCube's TR spike encoding: $f_{encode} : \mathbb{R}^{T \times N_{input}} \rightarrow \{-1, 0, 1\}^{T \times N_{input}}$

```
Require: $X_{in} \in \mathbb{R}^{T \times N_{input}}$, $\{hyperparameters := \alpha_{TR}\}$
Ensure: $\widetilde{W}_{train} \in \{-1, 0, 1\}^{T \times N_{input}}$
1: $N \leftarrow \#(X_{in})$ {number of data samples in the dataset}
2: for $k = 1$ to $N_{input}$ do
3:   $VT_k \leftarrow 0$
4:   for $i = 1$ to $N$ do
5:     $x \leftarrow$ channel $k$ of the $i$th sample in $X_{in}$
6:     $x' \leftarrow |\delta x|$
7:     $\mu \leftarrow mean(x')$
8:     $\sigma \leftarrow st.dev.(x')$
9:     $VT_k \leftarrow VT_k + (\mu + \sigma \cdot \alpha_{TR})$
10:  end for
11:  $VT_k \leftarrow \frac{VT_k}{N}$
12: end for
13: for $k = 1$ to $N_{input}$ do
14:   for $i = 1$ to $N$ do
15:     $x \leftarrow$ channel $k$ of the $i$th sample in $X_{in}$
16:     $x' \leftarrow \delta x$
17:     $x_{out} \leftarrow 0^{T \times 1}$
18:     for $j = 2$ to $T$ do
19:       if $x'_j > VT_k$ then
20:         $x_{out(j)} \leftarrow 1$
21:       else if $x'_j < -VT_k$ then
22:         $x_{out(j)} \leftarrow -1$
23:       end if
24:     end for
25:     store spike train $x_{out}$ in $X_{out}$ for channel $k$, sample $i$
26:   end for
27: end for
```

Algorithm 3 NeuCube's weight and connection initialization: $f_{initialize}$

```
Require: $X_{brain} \in \mathbb{R}^{1\times 3}, X_{input} \subset X_{brain}$
    $\{hyperparameters := SWR, p \in [0, 1]\}$
Ensure: $C : \{0, 1\}^{N_{cube}\times N_{cube}}, W : \mathbb{R}^{N_{cube}\times N_{cube}}$
 1: $N_{cube} \leftarrow \#(X_{brain})$ {number of defined neuron coordinates}
 2: $L_{dist} : \mathbb{R}^{N_{cube}\times N_{cube}} \leftarrow$ distances between all pairs of neurons
 3: $C_{ij} \leftarrow 1, \forall i, j$
 4: $W_{ij} \leftarrow 0, \forall i, j$
 5: for $i = 1$ to $N_{cube}$ do
 6:   for $j = 1$ to $N_{cube}$ do
 7:     if $L_{dist(ij)} > SWR$ then
 8:       $C_{ij} \leftarrow 0$
 9:     else
10:       $W_{ij} \leftarrow sgn(rand - p) \cdot rand \cdot L^{-1}_{dist(ij)}$ {$rand$ represents a randomly generated real number from
          the interval [0, 1]}
11:       if $C_{ij} = 1 \wedge C_{ji} = 1$ then
12:         if $rand > 0.5$ then
13:           $C_{ij} \leftarrow 0$
14:           $W_{ij} \leftarrow 0$
15:         else
16:           $C_{ji} \leftarrow 0$
17:           $W_{ji} \leftarrow 0$
18:         end if
19:       end if
20:     end if
21:   end for
22: end for
```

Algorithm 4 NeuCube's unsupervised SNNc weight learning: f_{STDP}

```
Require: $C_{init} \in \{0,1\}^{N_{cube} \times N_{cube}}$, $W_{init} \in \mathbb{R}^{N_{cube} \times N_{cube}}$, $S_{in} \in \{-1,0,1\}^{T \times N_{input}}$
    $\{hyperparameters := D, R, \eta, \beta, N_{iter}\}$
Ensure: $W_{out} : \mathbb{R}^{N_{cube} \times N_{cube}}$
 1: $N \leftarrow \#(X_{in})$ {number of samples in the (training) dataset}
 2: $\chi \leftarrow [1, 2, \ldots, N_{cube}]$ {all neuron indices}
 3: $P_k, \forall k \in \chi \leftarrow 0$ {initialize neuron potentials}
 4: $R_k, \forall k \in \chi \leftarrow 0$ {initialize neuron refractory time counters}
 5: find $inputneuron$ indices $\iota \subset \chi$
 6: for $n_{iter} = 1$ to $N_{iter}$ do
 7:   $\eta' \leftarrow \frac{\eta}{\sqrt{n_{iter}}}$
 8:   for $i = 1$ to $N$ do
 9:     $s \leftarrow$ all $N_{input}$ channel spikes of the $i$th sample in $S_{in}$ {inhibiting spikes ($-1$) are passed to virtual
        inhibitory neurons that accept only the inhibiting spikes}
10:     for $t = 1$ to $T$ do
11:       find $firingneuron$ indices $\tau = \{firingneurons$ in $\iota\} \cup \{k \in \chi \backslash \iota, P_k > \beta\}$
12:       for all $j \in \tau$ do
13:         find post synaptic neuron indices $\gamma$
14:         for all $k \in \gamma$ and $R_k = 0$ do
15:           $P_k \leftarrow P_k + w_{jk}$ {postsynaptic neuron potential update}
16:         end for
17:       end for
18:       $P_k \leftarrow 0, \forall k \in \tau$ {firing neurons reset potential}
19:       $R_k \leftarrow R, \forall k \in \tau$ {firing neurons reset refractory counter}
20:       $P_k \leftarrow max(0, P_k - D), \forall k \in \chi \backslash \iota \backslash \tau$
21:       $R_k \leftarrow max(0, R_k - 1), \forall k \in \chi \backslash \iota \backslash \tau$
22:       for all $j \in \tau$ do
23:         find post synaptic neuron indices $\gamma$
24:         for all $k \in \gamma$ do
25:           $w_{jk} \leftarrow w_{jk} - \eta(t - t_k^f)$
26:         end for
27:         find pre synaptic neuron indices $\gamma$
28:         for all $k \in \gamma$ do
29:           $w_{jk} \leftarrow w_{jk} + \eta(t - t_k^f)$
30:         end for
31:       end for
32:     end for
33:   end for
34: end for
```

Algorithm 5 NeuCube's deSNN output representation: f_{deSNN}

```
Require: $W \in \mathbb{R}^{N_{cube} \times N_{cube}}$, $S_{in} \in \{-1, 0, 1\}^{T \times N_{input}}$
    $\{hyperparameters := \alpha, d\}$
Ensure: $\widetilde{W} \in \mathbb{R}^{1 \times N_{cube}}$
1: $\widetilde{W} \leftarrow 0^{1 \times N_{cube}}$ {initialize representation to 0}
2: $F \leftarrow 0^{1 \times N_{cube}}$ {initialize the neuron firing flags}
3: $c \leftarrow 0$ {initialize the neuron firing order counter}
4: $S_{cube} \leftarrow$ propagate $S_{in}$ through the SNN defined by $W$ {$S_{cube}$ is a sparse matrix of all neuron firings
   through the data sample length period $T$}
5: for $i = 1$ to $T$ do
6:   for $j = 1$ to $N_{cube}$ do
7:     if $S_{cube}(i, j) = 1$ then
8:       if $F(j) = 0$ then
9:         {neuron fires for the first time}
10:        $\widetilde{W}(j) \leftarrow \alpha^c$
11:        $F(j) \leftarrow 1$
12:      else
13:        $\widetilde{W}(j) \leftarrow \widetilde{W}(j) + d$
14:      end if
15:      $c \leftarrow c + 1$
16:    else
17:      $\widetilde{W}(j) \leftarrow \widetilde{W}(j) - d$
18:    end if
19:  end for
20: end for
```

References

1. Hubel DH, Wiesel TN (1959) Receptive fields of single neurones in the cat's striate cortex. J Physiol 148(3):574–591
2. Fukushima K (1979) Neural network model for a mechanism of pattern recognition unaffected by shift in position-neocognitron. IEICE Tech Rep A 62(10):658–665
3. Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the brain. Psychol Rev 65(6):386
4. Hannun A, Case C, Casper J, Catanzaro B, Diamos G, Elsen E, Prenger R, Satheesh S, Sengupta S, Coates A et al (2014) Deep speech: scaling up end-to-end speech recognition. arXiv preprint arXiv:1412.5567
5. Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein M et al (2015) Imagenet large scale visual recognition challenge. Int J Comput Vis 115(3):211–252
6. Ouyang W, Wang X (2013) Joint deep learning for pedestrian detection. In: Proceedings of the IEEE international conference on computer vision, pp 2056–2063
7. Cireşan D, Meier U, Schmidhuber J (2017) Multi-column deep neural networks for image classification. arXiv preprint arXiv:1202.2745
8. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M et al (2016) Mastering the game of go with deep neural networks and tree search. Nature 529(7587):484
9. Šarlija M, Jurišić F, Popović S (2017) A convolutional neural network based approach to QRS detection. In: 10th international symposium on image and signal processing and analysis (ISPA). IEEE, pp 121–125
10. Ganapathy N, Swaminathan R, Deserno TM (2018) Deep learning on 1-d biosignals: a taxonomy-based survey. Yearb Med Inform 27(01):098–109
11. Drubach D (2000) The brain explained. Prentice Hall Health, Upper Saddle River
12. Bengio Y, Lee D-H, Bornschein J, Mesnard T, Lin Z (2015) Towards biologically plausible deep learning. arXiv preprint arXiv:1502.04156
13. Maass W (1997) Networks of spiking neurons: the third generation of neural network models. Neural Netw 10(9):1659–1671
14. Trentin E, Schwenker F, El Gayar N, Abbas HM (2018) Off the mainstream: advances in neural networks and machine learning for pattern recognition. Neural Process Lett 48(2):643–648

15. Kasabov NK (2014) Neucube: a spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data. Neural Netw 52:62–76
16. Herz AV, Gollisch T, Machens CK, Jaeger D (2006) Modeling single-neuron dynamics and computations: a balance of detail and abstraction. Science 314(5796):80–85
17. Lapicque L (1907) Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization. J Physiol Pathol Gen 9:620–635
18. Abbott LF (1999) Lapicque's introduction of the integrate-and-fire model neuron (1907). Brain Res Bull 50(5–6):303–304
19. Gerstner W, Kistler WM (2002) Spiking neuron models: single neurons, populations, plasticity. Cambridge University Press, Cambridge
20. Izhikevich EM (2003) Simple model of spiking neurons. IEEE Trans Neural Netw 14(6):1569–1572
21. Izhikevich EM (2004) Which model to use for cortical spiking neurons? IEEE Trans Neural Netw 15(5):1063–1070
22. Hodgkin AL, Huxley AF (1952) A quantitative description of membrane current and its application to conduction and excitation in nerve. J Physiol 117(4):500–544
23. Wilson C, Callaway J (2000) Coupled oscillator model of the dopaminergic neuron of the substantia nigra. J Neurophysiol 83(5):3084–3100
24. FitzHugh R (1961) Fitzhugh-nagumo simplified cardiac action potential model. Biophys J 1:445–466
25. Hindmarsh JL, Rose R (1984) A model of neuronal bursting using three coupled first order differential equations. Proc R Soc Lond B 221(1222):87–102
26. Morris C, Lecar H (1981) Voltage oscillations in the barnacle giant muscle fiber. Biophys J 35(1):193–213
27. Katsumata S, Sakai K, Toujoh S, Miyamoto A, Nakai J, Tsukada M, Kojima H (2008) Analysis of synaptic transmission and its plasticity by glutamate receptor channel kinetics models and 2-photon laser photolysis. In: Proceedings of ICONIP
28. Huguenard JR (2000) Reliability of axonal propagation: the spike doesn't stop here. Proc Nat Acad Sci 97(17):9349–9350
29. Kasabov N (2010) To spike or not to spike: a probabilistic spiking neuron model. Neural Netw 23(1):16–19
30. Sengupta N, Kasabov N (2017) Spike-time encoding as a data compression technique for pattern recognition of temporal data. Inf Sci 406:133–145
31. Adrian ED (1926) The impulses produced by sensory nerve endings. J Physiol 61(1):49–72
32. Gautrais J, Thorpe S (1998) Rate coding versus temporal order coding: a theoretical approach. Biosystems 48(1–3):57–65
33. Lestienne R (2001) Spike timing, synchronization and information processing on the sensory side of the central nervous system. Prog Neurobiol 65(6):545–591
34. Bohte SM (2004) The evidence for neural information processing with precise spike-times: a survey. Nat Comput 3(2):195–206
35. Thorpe SJ (1990) Spike arrival times: a highly efficient coding scheme for neural networks. In: Eckmiller R, Hartmann G, Hauske G (eds) Parallel processing in neural systems and computers. North-Holland Elsevier, pp 91–94
36. Brette R (2015) Philosophy of the spike: rate-based vs. spike-based theories of the brain. Front Syst Neurosci 9:151
37. Mohemmed A, Schliebs S, Matsuda S, Kasabov N (2011) Method for training a spiking neuron to associate input-output spike trains. In: Engineering applications of neural networks. Springer, pp 219–228
38. Thorpe S, Gautrais J (1998) Rank order coding. In: Computational neuroscience. Springer, pp 113–118
39. Buzsaki G (2006) Rhythms of the brain. Oxford University Press, Oxford
40. Petro B, Kasabov N, Kiss RM (2019) Selection and optimization of temporal spike encoding methods for spiking neural networks. IEEE Trans Neural Netw Learn Syst 31(2):358–370
41. Kasabov NK (2018) Time-space. Spiking neural networks and brain-inspired artificial intelligence. Springer, Berlin
42. Rueckauer B, Lungu I-A, Hu Y, Pfeiffer M (2016) Theory and tools for the conversion of analog to spiking convolutional neural networks. arXiv preprint arXiv:1612.04052
43. Diehl PU, Cook M (2015) Unsupervised learning of digit recognition using spike-timing-dependent plasticity. Front Comput Neurosci 9:99
44. Diehl PU, Neil D, Binas J, Cook M, Liu S-C, Pfeiffer M (2015) Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In: 2015 international joint conference on neural networks (IJCNN). IEEE, pp 1–8
45. Cao Y, Chen Y, Khosla D (2015) Spiking deep convolutional neural networks for energy-efficient object recognition. Int J Comput Vis 113(1):54–66

46. Merolla P, Arthur J, Akopyan F, Imam N, Manohar R, Modha DS (2011) A digital neurosynaptic core using embedded crossbar memory with 45pj per spike in 45 nm. In: Custom integrated circuits conference (CICC), 2011 IEEE. IEEE, pp 1–4
47. O'Connor P, Neil D, Liu S-C, Delbruck T, Pfeiffer M (2013) Real-time classification and sensor fusion with a spiking deep belief network. Front. Neurosci. 7:178
48. Hunsberger E, Eliasmith C (2015) Spiking deep networks with LIF neurons. arXiv preprint arXiv:1510.08829
49. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems, pp 1097–1105
50. Esser SK, Appuswamy R, Merolla P, Arthur JV, Modha DS (2015) Backpropagation for energy-efficient neuromorphic computing. In: Advances in neural information processing systems, pp 1117–1125
51. Merolla PA, Arthur JV, Alvarez-Icaza R, Cassidy AS, Sawada J, Akopyan F, Jackson BL, Imam N, Guo C, Nakamura Y et al (2014) A million spiking-neuron integrated circuit with a scalable communication network and interface. Science 345(6197):668–673
52. Gerstner W, Kempter R, van Hemmen JL, Wagner H (1996) A neuronal learning rule for sub-millisecond temporal coding. Nature 383(6595):76
53. Bi G-Q, Poo M-M (1998) Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. J Neurosci 18(24):10464–10472
54. Gerstner W, Ritz R, Van Hemmen JL (1993) Why spikes? Hebbian learning and retrieval of time-resolved excitation patterns. Biol Cybern 69(5–6):503–515
55. Cassenaer S, Laurent G (2007) Hebbian STDP in mushroom bodies facilitates the synchronous flow of olfactory information in locusts. Nature 448(7154):709
56. Jacob V, Brasier DJ, Erchova I, Feldman D, Shulz DE (2007) Spike timing-dependent synaptic depression in the in vivo barrel cortex of the rat. J Neurosci 27(6):1271–1284
57. Mu Y, Poo M-M (2006) Spike timing-dependent ltp/ltd mediates visual experience-dependent plasticity in a developing retinotectal system. Neuron 50(1):115–125
58. Song S, Miller KD, Abbott LF (2000) Competitive hebbian learning through spike-timing-dependent synaptic plasticity. Nat Neurosci 3(9):919
59. Kheradpisheh SR, Ganjtabesh M, Thorpe SJ, Masquelier T (2018) Stdp-based spiking deep convolutional neural networks for object recognition. Neural Netw 99:56–67
60. Tavanaei A, Maida AS (2017) A spiking network that learns to extract spike signatures from speech signals. Neurocomputing 240:191–199
61. Hirsch H-G, Pearce D (2000) The aurora experimental framework for the performance evaluation of speech recognition systems under noisy conditions. In: ASR2000-automatic speech recognition: challenges for the new Millenium ISCA Tutorial and Research Workshop (ITRW)
62. Kasabov N et al (1998) Evolving fuzzy neural networks-algorithms, applications and biological motivation. Methodologies for the conception, design and application of soft computing. World Sci 1:271–274
63. Kasabov NK (2007) Evolving connectionist systems: the knowledge engineering approach. Springer, Berlin
64. Wysoski SG, Benuskova L, Kasabov N (2010) Evolving spiking neural networks for audiovisual information processing. Neural Netw 23(7):819–835
65. Kasabov N, Dhoble K, Nuntalid N, Indiveri G (2013) Dynamic evolving spiking neural networks for on-line spatio-and spectro-temporal pattern recognition. Neural Netw 41:188–201
66. Kasabov N (2012) Neucube evospike architecture for spatio-temporal modelling and pattern recognition of brain signals. In: IAPR workshop on artificial neural networks in pattern recognition. Springer, pp 225–243
67. Lichtsteiner P, Delbruck T (2005) A 64 $\times$ 64 AER logarithmic temporal derivative silicon retina. In: Research in microelectronics and electronics PhD, vol 2. IEEE, pp 202–205
68. Nuntalid N, Dhoble K, Kasabov N (2011) EEG classification with BSA spike encoding algorithm and evolving probabilistic spiking neural network. In: International conference on neural information processing. Springer, pp 451–460
69. Kasabov N, Scott NM, Tu E, Marks S, Sengupta N, Capecci E, Othman M, Doborjeh MG, Murli N, Hartono R et al (2016) Evolving spatio-temporal data machines based on the neucube neuromorphic framework: design methodology and selected applications. Neural Netw 78:1–14
70. Talairach J, Tournoux P (1988) Co-planar stereotaxic atlas of the human brain: 3-dimensional proportional system: an approach to cerebral imaging
71. Evans AC, Collins DL, Mills S, Brown E, Kelly R, Peters TM (1993) 3D statistical neuroanatomical models from 305 MRI volumes. In: Nuclear science symposium and medical imaging conference. 1993 IEEE conference record. IEEE, pp 1813–1817

72. Kasabov NK, Doborjeh MG, Doborjeh ZG (2016) Mapping, learning, visualization, classification, and understanding of fmri data in the neucube evolving spatiotemporal data machine of spiking neural networks. IEEE Trans Neural Netw Learn Syst 28(4):887–899
73. Kasabov N, Zhou L, Doborjeh MG, Doborjeh ZG, Yang J (2016) New algorithms for encoding, learning and classification of fmri data in a spiking neural network architecture: a case on modeling and understanding of dynamic cognitive processes. IEEE Trans Cogn Dev Syst 9(4):293–303
74. Abbott A, Sengupta N, Kasabov N (2016) Which method to use for optimal structure and function representation of large spiking neural networks: a case study on the neucube architecture. In: International joint conference on neural networks (IJCNN), 2016 . IEEE, pp 1367–1372
75. Taylor D, Scott N, Kasabov N, Capecci E, Tu E, Saywell N, Chen Y, Hu J, Hou Z-G (2014) Feasibility of neucube SNN architecture for detecting motor execution and motor intention for use in BCI applications. In: International joint conference on neural networks (IJCNN), 2014 . IEEE, pp 3221–3225
76. Hu J, Hou Z-G, Chen Y-X, Kasabov N, Scott N (2014) EEG-based classification of upper-limb ADL using SNN for active robotic rehabilitation. In: 2014 5th IEEE RAS & EMBS international conference on biomedical robotics and biomechatronics. IEEE, pp 409–414
77. Othman M, Kasabov N, Tu E, Feigin V, Krishnamurthi R, Hou Z, Chen Y, Hu J (2014) Improved predictive personalized modelling with the use of spiking neural network system and a case study on stroke occurrences data. In: 2014 international joint conference on neural networks (IJCNN). IEEE, pp 3197–3204
78. Doborjeh ZG, Kasabov N, Doborjeh MG, Sumich A (2018) Modelling peri-perceptual brain processes in a deep learning spiking neural network architecture. Sci Rep 8(1):8912
79. Paulun L, Wendt A, Kasabov NK (2018) A retinotopic spiking neural network system for accurate recognition of moving objects using neucube and dynamic vision sensors. Front Comput Neurosci 12:42
80. Sengupta N, McNabb CB, Kasabov N, Russell BR (2018) Integrating space, time, and orientation in spiking neural networks: a case study on multimodal brain data modeling. IEEE Trans Neural Netw Learn Syst 99:1–15
81. Oreilly C, Gosselin N, Carrier J, Nielsen T (2014) Montreal archive of sleep studies: an open-access resource for instrument benchmarking and exploratory research. J Sleep Res 23(6):628–635
82. Koelstra S, Muhl C, Soleymani M, Lee J-S, Yazdani A, Ebrahimi T, Pun T, Nijholt A, Patras I (2012) Deap: A database for emotion analysis; using physiological signals. IEEE Trans Affect Comput 3(1):18–31
83. Soleymani M, Lichtenauer J, Pun T, Pantic M (2012) A multimodal database for affect recognition and implicit tagging. IEEE Trans Affect Comput 3(1):42–55