Neural Networks 121 (2020) 88–100

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Spiking Neural Networks and online learning: An overview and perspectives

Jesus L. Lobo[a,*], Javier Del Ser[a,b,c], Albert Bifet[d,e], Nikola Kasabov[f]

[a] TECNALIA, 48160 Derio, Spain
[b] Basque Center for Applied Mathematics (BCAM), 48009 Bilbao, Spain
[c] University of the Basque Country UPV/EHU, 48013 Bilbao, Spain
[d] Télécom ParisTech, París, C201-2, France
[e] University of Waikato, Hamilton, New Zealand
[f] Auckland University of Technology (AUT), Auckland, New Zealand

ARTICLE INFO

Article history:
Available online 7 September 2019

Keywords:
Online learning
Spiking Neural Networks
Stream data
Concept drift

ABSTRACT

Applications that generate huge amounts of data in the form of fast streams are becoming increasingly prevalent, being therefore necessary to learn in an online manner. These conditions usually impose memory and processing time restrictions, and they often turn into evolving environments where a change may affect the input data distribution. Such a change causes that predictive models trained over these stream data become obsolete and do not adapt suitably to new distributions. Specially in these non-stationary scenarios, there is a pressing need for new algorithms that adapt to these changes as fast as possible, while maintaining good performance scores. Unfortunately, most off-the-shelf classification models need to be retrained if they are used in changing environments, and fail to scale properly. Spiking Neural Networks have revealed themselves as one of the most successful approaches to model the behavior and learning potential of the brain, and exploit them to undertake practical online learning tasks. Besides, some specific flavors of Spiking Neural Networks can overcome the necessity of retraining after a drift occurs. This work intends to merge both fields by serving as a comprehensive overview, motivating further developments that embrace Spiking Neural Networks for online learning scenarios, and being a friendly entry point for non-experts.

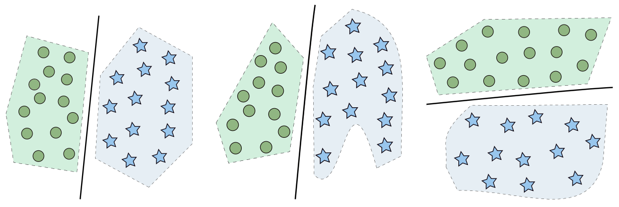
1. Introduction

The term *Big Data* has gained progressive momentum during the last decade, due to the feasibility of collecting data from almost any source and analyzing to achieve data-based insights that enable cost and time reductions, new product developments, optimized offerings, or smart decision making, among others profits. In these *Big Data* scenarios, some characteristics may play a relevant role: it is not feasible to store the whole dataset, traditional algorithms cannot handle data produced at high rates, and changes in data distribution may occur during learning process. An increasing number of applications are based on these training data continuously available (*stream learning*), and applied to real scenarios, such as mobile phones, sensor networks, industrial process controls and intelligent user interfaces, among others (Diez-Olivan, Del Ser, Galar, & Sierra, 2019; Žliobaitė, Pechenizkiy, & Gama, 2016). Some of these applications produce non-stationary data streams which are becoming increasingly prevalent, and where the process generating the data may change over time, producing changes in the patterns to be modeled (*concept drift*). This causes that predictive models trained over these streaming data become obsolete and do not adapt suitably to the new distribution. Especially in Online Learning (OL) scenarios, where only a single sample is provided to the learning algorithm at every time instant, there is a pressing need for new algorithms that adapt to these changes as fast as possible, while maintaining good performance scores. OL in the presence of concept drift has been a very hot topic during the last few years (Gama, Žliobaitė, Bifet, Pechenizkiy, & Bouchachia, 2014; Webb, Hyde, Cao, Nguyen, & Petitjean, 2016), and still remains under active debate in the community because of its numerous open challenges (Krawczyk, Minku, Gama, Stefanowski, & Woźniak, 2017; Losing, Hammer, & Wersing, 2018). The data mining community prefers to refer to OL in the presence of concept drift as *data stream mining* (Bifet, Gavaldà, Holmes, & Pfahringer, 2018; De Francisci Morales, Bifet, Khan, Gama, & Fan, 2016).

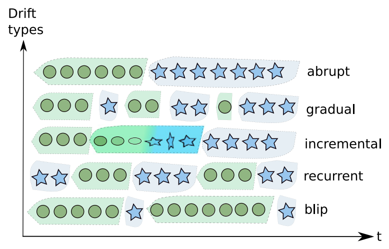
Many algorithms have been developed for stream learning based on Machine Learning (ML) techniques. Unfortunately, most

[*] Correspondence to: TECNALIA. P. Tecnologico Bizkaia, Ed. 700, 48160 Derio, Spain.
E-mail address: jesus.lopez@tecnalia.com (J.L. Lobo).

https://doi.org/10.1016/j.neunet.2019.09.004
0893-6080/© 2019 Elsevier Ltd. All rights reserved.



(a) Original, virtual ($p(\mathbf{x})$ changes, but not $p(y|\mathbf{x})$) and real ($p(y|\mathbf{x})$ changes) concepts respectively



(b) Types of drift

Fig. 1. (*a*) nature of drift depending on what is changing; (*b*) types of drift.

off-the-shelf models need to be retrained if they are used in an evolving environment, and fail to scale properly due to their learning algorithm. Artificial Neural Networks (ANNs) have been used in the last years to deal with these fast evolving information flows. In essence, they are a biologically inspired paradigm that mimics the process through which the brain acquires and processes sensory information. One of their most biologically plausible neuron models is a key ingredient of the so-called Spiking Neural Network (SNN) (Gerstner & Kistler, 2002), a popular and reputed model for its capacity to capture informational dynamics observed among real biological neurons, and to represent and integrate several information dimensions (e.g. time, space, frequency, phase, and to deal with large volumes of data) into a single model. The theory behind SNNs is currently mostly accepted to describe realistic *brain-like* information processing, which in addition eases their implementation on super-fast and reliable hardware platforms.

Considered nowadays as the third generation of ANNs (Maass, 1997), the advent of SNN was propelled by the need for a better understanding of the information processing skills of the mammalian brain, for which the community committed itself to the development of more complex biologically connectionist systems. Some SNNs are especially well-known in the OL research community for their ability to learn continuously and incrementally, which account for their continuous adaptability to non-stationary and evolving environments, and also their capacity to serve as drift detectors (Lobo, Del Ser et al., 2018). Besides, they have shown the ability to capture temporal associations between temporal variables in streaming data.

From all the rationale exposed above, the merging of both fields motivates further developments that embrace SNNs for OL scenarios, with emphasis on those requiring concept drift detection and adaptation. This work intends to serve as a suitable entry point in the literature for non-experts in both fields, and a catalyst material for future research efforts invested in this direction; it is organized as follows: while, Sections 2 and 3 provide a general introduction, and present challenges and future work for OL scenarios and SNNs respectively, Section 4 delves into the convergence of both fields. Finally, Section 5 draws the conclusions related to this study.

2. Online learning

In stream learning, data may arrive in chunks of data (*batch learning*) or in an online manner. In *batch learning* an entirely accessible group of samples (batch) is provided, and the learning algorithm is allowed to scan the batch before building/updating the model. However, in OL only a single sample is provided to the learning algorithm at every time instant, which is incrementally updated every time a new sample arrives. An OL environment imposes different computational constraints when compared to traditional batch learning setting:

- each sample is processed only once *on arrival*, and models must be able to process samples sequentially as soon as they are received, without putting the memory space and processing time restrictions at risk;
- the processing time of each sample must be small and constant, without exceeding the rate at which new samples arrive;
- the algorithm should use only a preallocated and finite amount of memory;
- a valid model must be available at every scan of the streams of data; and
- the learning algorithm must produce a model that is equivalent to the one that would be built in a batch learning scenario.

In *batch learning*, the evaluation procedure of the learning algorithm is determined by the set of samples used for training and testing. The question raised for OL in this regard is how to build a picture of accuracy over time. One of the most used schemes is *test-then-train*, where each sample is used to test the model before it is used for training, and then the accuracy can be incrementally updated. This scheme has the advantage that can be applied when memory is restricted and there is no holdout set for testing, getting the most out of the available dataset. Next, we introduce the problem of OL in the presence of concept drift, probably the most challenging aspect in OL, being a very hot topic research in the last decade (Webb et al., 2016).

2.1. Concept drift

Learning and adaptation to drift in non-stationary environments (Lu et al., 2018) requires modeling approaches capable of monitoring, tracking and adapting to eventual changes in the produced data. In this context, a drift may occur in the feature domain (new features appear, part of them disappear, or their value range evolves) or in the class domain (new classes emerge or some of them fade along time). More formally, concept drift between time step ts_0 and ts_1 can be defined as:

$$\exists x \in X : p_{ts_0}(\mathbf{x}, y) \neq p_{ts_1}(\mathbf{x}, y), \quad (1)$$

where $p_{ts_0}(\mathbf{x}, y)$ and $p_{ts_1}(\mathbf{x}, y)$ are the joint probability distributions at time steps ts_0 and ts_1, respectively, between the input variables $\mathbf{x}$ that conform a data sample and the target variable y. Specific terminology is often used to indicate the cause or nature of changes. In terms of what is changing (see Fig. 1), drifts can be:

Real drift when the posterior probabilities of classes $p(y|\mathbf{x})$ vary over time independently from variations in $p(\mathbf{x})$, having an impact on unconditional probability density functions, and

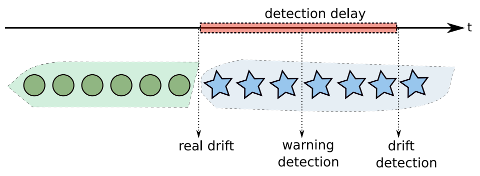


Fig. 2. Drift detection example.

Virtual drift when the distribution of the input data $p(\mathbf{x})$ changes without affecting the posterior probabilities of classes $p(y|\boldsymbol{x})$, but affects only the conditional probability density functions.

Concept drift can be also categorized in terms of speed and severity (Minku, 2011). On the one hand, in the case of speed, an *abrupt* drift may occur when a change happens suddenly between two classification contexts, whereas a *gradual* drift represents the case when dealing with a smooth transition between two concepts. When there are several intermediate concepts in between the old and the new concept, the change is *incremental*. Likewise, if previously known concepts reoccur after some time, it is considered as a *recurrent* drift. Finally, a challenge may emerge when an outlier (blip) can be mixed with concept drift; in this case no adaptation is needed because it is a temporary event that does not affect the future data and thus the subsequent learning of the algorithm. Fig. 1 also shows these types of drifts. On the other hand, severity can be regarded as the amount of changes that a new concept causes; therefore, a measure of severity can be computed as the percentage of the input space whose target class has changed after the drift. It is worth mentioning that many efforts have been devoted to the characterization of concept drift (Khamassi, Sayed-Mouchaweh, Hammami, & Ghédira, 2018; Webb et al., 2016), however, it is still an open issue in the state-of-the-art due to the complexity of characterizing manifold types of data changes over time.

Algorithms that handle concept drift can be designed for adaptation (Losing et al., 2018) or detection (Barros & Santos, 2018) purposes, even both when the active strategy requires a detection mechanism to trigger the adaptation process. The adaptation process can be carried out proactively (first detecting concept drift, so that only the model gets updated when a drift is detected, known as *active* or *informed* approaches), or passively (updating the model continuously every time new data samples are received, known as *passive* or *blind* approaches). *Passive* approaches are effective with gradual drifts and recurring concepts (although *active* ones are also able to do it but with more difficulties), and they are more recommendable for *batch learning*. *Active* approaches work well when the drift is abrupt, and they are more recommendable for OL.

Drift detectors are methods that can detect data distribution changes based on information about a base learner performance (e.g. classification errors) or the incoming data. Such changes usually trigger the need for updating, replacing or retraining the model (or the ensemble, or part of it). Drift detectors may return not only signals about drift occurrence, but also warning signals, which represent the moment when a change is suspected and a new training set representing the new concept should start being collected to retrain the models, as Fig. 2 shows.

2.2. Applications

The *Internet of Things (IoT)* has become one of the main applications of OL, since it is generating huge quantity of data continuously in real time. The IoT may be defined as sensors and actuators connected by networks to computing systems (Manyika et al., 2015), which can monitor/manage the health and actions of connected objects/machines in real-time. It has the potential to fundamentally shift the way we interact with our surroundings, or dramatically improve health outcomes with wearable devices and portable monitors. It has also the potential to deliver trillions of dollars in economic growth in the coming years. It will boost productivity, drive the emergence of new markets, and encourage innovation. In Middleton, Tsai, Yamaji, Gupta, and Rueb (2017), the authors predict that the IoT will grow at a 32% rate from 2016 through 2021, reaching an installed base of 25.1 billion units. In 2021, 7.6 billion "things" will ship, with 64% being consumer applications. The IoT will support total spending on endpoints and services of about $3.9 trillion in 2021. The characteristics of the IoT are:

Interconnectivity: anything can be interconnected with the global information and communication infrastructure,

Things-related services: the IoT is capable of providing thing-related services within the constraints of things (e.g. privacy protection and semantic consistency between physical things and their associated virtual things). In order to provide thing-related services within the constraints of things, both the technologies and information in the physical world will change,

Heterogeneity: IoT devices are heterogeneous as based on different hardware platforms and networks. They can interact with other devices or service platforms through different networks,

Dynamic changes: the state of devices change dynamically, e.g., sleeping and waking up, connected and/or disconnected as well as the context of devices including location and speed. Moreover, the number of devices can change dynamically,

Enormous scale: the number of devices that need to be managed and that communicate with each other will be at least an order of magnitude larger than the devices connected to the current Internet. The ratio of communication triggered by devices as compared to communication triggered by humans will noticeably shift towards device-triggered communication. Even more critical will be the management of the data generated and their interpretation for application purposes. This relates to semantics of data, as well as efficient data handling.

In IoT applications, OL algorithms are needed to manage the data currently generated, at an ever increasing rate, from applications such as: sensor networks, measurements in network monitoring and traffic management, log records or click-streams in web exploring, manufacturing processes, digital twins, call detail records, email, blogging, Twitter posts and others (Žliobaitė et al., 2016). In fact, all data generated can be considered as streaming data since it is obtained in specific intervals of time.

2.3. Available open software/frameworks

The references provided here contain software implementations for algorithms that can work on stationary and non-stationary scenarios. We do not claim this list to be exhaustive, but provides several opportunities for novices to get started, and for established researchers to expand their contributions, all the while advancing the OL field by tackling some of the open challenges described in the next subsection. These references have been divided into distributed and non-distributed categories in Table 1. Nowadays data are distributed in nature, for almost any

Table 1
Available Open Software/Frameworks for streaming distributed (D) and non-distributed (ND) architectures.

Framework	D/ND	Description	Reference
MOA	ND	Probably the most popular open source Java framework for OL	(Bifet et al., 2018)
Scikit-Multiflow	ND	Inspired by MOA and implemented in Python, it contains a collection of ML algorithms, datasets, tools, and metrics for OL evaluation	(Montiel, Read, Bifet, & Abdessalem, 2018)
Scikit-Learn	ND	Although it is mainly focused on batch learning, this framework also provides researchers with some OL methods (Multinomial Naive Bayes, Perceptron, a Stochastic Gradient Descent classifier, a Passive Aggressive classifier, among others)	(Pedregosa et al., 2011)
Spark streaming	D	An Apache project focused on building scalable fault-tolerant streaming applications	(Foundation, 2012; Zaharia et al., 2016)
Flink	D	An Apache framework and distributed processing engine for stateful computations over unbounded and bounded data streams. It has been designed to run in all common cluster environments, perform computations at in-memory speed and at any scale	(Carbone et al., 2015)
Storm	D	Another distributed real-time computation system under an Apache project to reliably process unbounded streams of data	(Iqbal & Soomro, 2015)
Beam	D	Apache project that provides a unified model for defining both batch and streaming data-parallel processing pipelines	(Foundation, 2017)
Samza	D	Another Apache project which offers a scalable data processing engine that allows us to process and analyze our data in real-time	(Noghabi et al., 2017)
SAMOA	D	It is a Scalable Advanced Massive Online Analysis tool like MOA but for distributed stream learning	(Morales & Bifet, 2015)

task information is gathered over a broad area, and streams are delivered at a much greater rate than ever before. Common devices that surround us are gaining sensors, computational power, and mechanisms, changing from static and inanimate devices into adaptive and reactive systems.

2.4. Recent trends and future challenges

Next, we summarize some of the most remarkable challenges and trends in OL (Benczúr, Kocsis, & Pálovics, 2018; Gomes, Barddal, Enembreck, & Bifet, 2017; Ramírez-Gallego, Krawczyk, García, Woźniak, & Herrera, 2017; Wang, Minku, & Yao, 2018):

Structured prediction: instead of having only one output attribute to predict, in structured prediction we may have several output attributes at the same time, that can be numeric or discrete. If they are discrete, we refer to it using the term *multi-label learning* and if they are numeric, the term *multi-target learning*,

Semisupervised and delayed learning: as the number of available class labels may be small, we may need to used semi-supervised techniques to get benefit of having huge quantities of unlabeled data to make predictions. Another interesting and real case scenario is the one where class labels arrive with delay. We may use also semi-supervised techniques to deal with this delayed setting,

Active learning: if there is a cost for obtaining the label of a sample, active learning can be used to decide which samples to select to pay the cost for the label, optimizing the cost and the number of labels used,

Data preprocessing: in high-dimensional data, using all attributes cannot be feasible, and we may need to preprocess the data to perform feature selection, or feature transformation. How to do that in an efficient way is still quite challenging,

Imbalanced learning and anomaly detection: in many applications data are not balanced, the distribution of the class labels is not uniform and may be evolving other time. One application of imbalanced learning is anomaly detection, where the problem consists of predicting when anomalies appear; as they appear with a very low frequency, it is a classical example of imbalanced learning.

Distributed computation: the traditional sequential learning algorithms that deal with data from modern devices which produce huge amount of data, usually work on a single machine which is limited by its memory and bandwidth. Besides, existing non-distributed streaming frameworks are unable to scale when handling very high incoming data rates. Distributed stream learning is a new emergent paradigm that addresses these issues, allowing to implement parallel computation on streams, combining the scalability of distributed processing with the efficiency of OL algorithms, and easing their deployment on top of state-of-the-art distributed stream processing engines. And,

The use of neural networks: how to execute only doing one pass over the data will be an important future area of research, considering that the standard deep learning techniques currently do several passes over the data.

Online optimization: dynamic optimization techniques are designed to deal with problems that evolve along time, a more realistic setup when the problem is defined over non-stationary scenarios (Del Ser et al., 2019). The use of optimization techniques might be deemed incompatible with an online learning setting, where fast decisions are required, but here dynamic optimization techniques could provide compatible solutions. For example, with the changes within the stream, the initial set parameters may no longer be sufficiently good, and it is required a new set of values for a quick adaptation to the change. In this respect, Lobo, Del Ser, Bilbao, Perfecto, and Salcedo-Sanz (2018) showed how it is possible to find a good solution for an optimization problem in a streaming non-stationary environment.

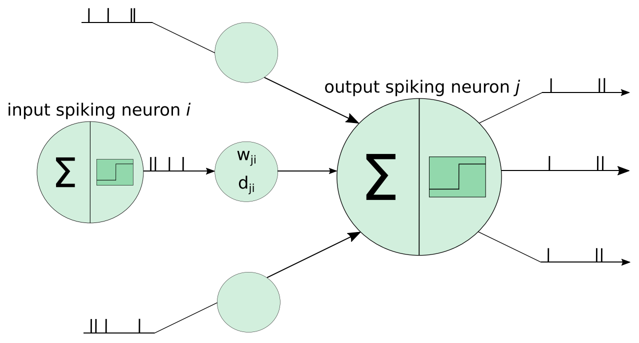


Fig. 3. Scheme for SNNs.

3. Spiking Neural Networks (SNNs)

The computational power of bio-inspired systems has attracted increasing attention from research community (Kasabov, 2019). Despite the lack of consensus about the information processing involved in brain, biological processes have served as reference for recent computational models. ANNs were developed as simplified versions of biological neural networks in terms of structure and function. SNNs have raised as the new generation of neural networks, a more biological realistic approach by utilizing spikes, incorporating the concepts of space and time through neural connectivity and plasticity. They deal with precise timing spikes being competitive with the traditional ANNs in terms of accuracy and computational power, and in some cases potentially better suited for hardware implementation due to their simple "integrate-and-fire" nature (see Section 3.3). In order to better understand in what aspects SNNs (3rd generation) differ from other classical ANNs (1st and 2nd generation), and which consequences these differences have, we first present Table 2 where these three generations are compared.

SNNs have optimal characteristics for hardware implementation, which is an interesting topic because online operation can be achieved and embedded hardware systems, enlarging the number of applications where SNNs can be applied (e.g. a custom VLSI chip Vogelstein, Mallik, Culurciello, Cauwenberghs, & Etienne-Cummings, 2005). In this regard, some of their advantages are:

- there are no multiplications as in traditional models,
- pulse processing can be implemented using shifts and adds,
- interconnections transmit only a single bit instead of real numbers, and
- sparse and asynchronous communication can also be easily implemented. However, it should be remarked that this prospective advantage does not manifest itself yet when implementing SNNs in a general purpose computer platform

We would also like to underline the importance of models interpretation. By having the possibility of building a model visualization or extracting the set of rules that govern the SNN model, we are able to interpret the internal mechanisms behind the model learning. In contrast to other ANNs where it is hard to look into the network and figure out exactly what or how it has learned, some SNNs allow us to know about their learning process (Soltic & Kasabov, 2010).

Despite their problems and limitations, they are one of the most promising techniques in ML. Now the scope is placed on SNNs (see Fig. 3), providing a general overview on this family of connectionist models, showing their principles, current applications, and future trends and challenges (Kasabov, 2018).

3.1. Biological inspiration

The idea of being inspired by the way brain performs neural computation for building computer-based algorithms has been present for more than almost eighty years (McCulloch & Pitts, 1943; Rosenblatt, 1958). These bio-inspired algorithms of neural computation are referred to as ANNs; they contain a set of computational units (neurons) interconnected via directed edges (synapses between neurons) which process information according to a specified set of rules and equations (the model of information processing in neural circuits). Different choices for their connectivity or dynamics have given rise to a vast range of different types of models which have been thoroughly studied in computer science over the last decades, and which have enjoyed a revived interest in recent years due to the success of Deep Learning (Hinton & Salakhutdinov, 2006; Tavanaei, Ghodrati, Kheradpisheh, Masquelier, & Maida, 2018).

Advances in consolidating the vast number of new findings and insights from neuroscience into such computational models in a biologically plausible way have been largely lacking in the mainstream ANN community. While it has been known for long that neurons communicate with spikes (electrical pulses, action potentials), it was only in the early 1990s when studies found evidence for biological brains making use of the exact timing of single spikes to encode information (Abeles, Bergman, Margalit, & Vaadia, 1993; Bair, 1994). This observation gave rise to SNNs after their property of explicitly modeling individual spikes, rather than average firing rates like their predecessors. The utilization of spikes brings together the definitions of time varying post-synaptic potential (PSP), firing threshold (ϑ), and spike latencies (Δ), as depicted in Fig. 4. They try to simulate the processes carried out between the neurons (synapses) in a network.

More than two decades have passed since one of the most influential papers on the topic was published (Maass, 1997). Nowadays SNNs can still be considered a niche of artificial intelligence research.

3.2. Data and information representation as spikes

Before presenting the input data to the SNN, it must be encoded into spike trains in order to apply the neuron model. The encoding part aims to generate spiking patterns that represent the input stimuli, and this is still an open issue in neuroscience where several significant questions rise, among others:

- what is the information contained in such a spatio-temporal pattern of spikes?
- what is the code used by neurons to transmit that information?
- how might other neurons decode the signal?

It becomes crucial to generate spike trains such that the task-relevant information content of the input stimuli is preserved. Thus, the decision of what information is lost, what is preserved and how effective the encoding was, is not trivial (Petro, Kasabov, & Kiss, 2019). But traditionally it has been shown that most of the relevant information is contained in the mean firing rate of neurons. In the literature we can find two main encoding schemes: *temporal encoding* and *rate-based encoding* (see Fig. 5). The first one is used over the latter one when patterns within the encoding window provide information about the stimulus that cannot be obtained from spike count. The *rate-based encoding* scheme is based on a spiking characteristic within a time interval (e.g. frequency), in the *temporal encoding* scheme the information is encoded in the time of spikes.

Rate-based encoding schemes ("rate as a spike count", "rate as a spike density", and "rate as a population activity") correspond

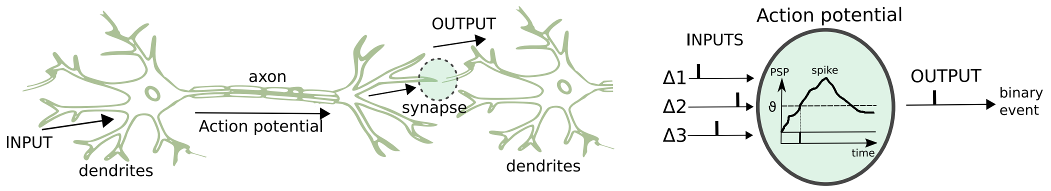


Fig. 4. Biological neuron and its association with an artificial spiking neuron (Gerstner & Kistler, 2002).

Table 2
Comparison between the different generations of ANNs. The SSTD term refers to Spatio- and Spectro-Temporal Data, VLSI refers to Very Large Scale Integrated, MLP refers to Multilayer Perceptron, CNN refers to Convolutional Neural Network, and RNN refers to Recurrent Neural Network.

	[Input, Output]	Advantages and disadvantages	Examples	References
1st gen.	[Binary, Binary]	Advantages: (1) Universal for digital computation (2) Very intuitive interpretation as spikes (3) Low number of parameters that need to be optimized Disadvantages: (1) Restricted to binary outputs (2) Limited in an analogue world (3) Not include notion of time, they code fires (spikes)	Perceptron, MLP	(McCulloch & Pitts, 1943)
2nd gen.	[Real, Real]	Advantages: (1) Universal for digital computation (2) More biologically plausible than 1st generation (3) They are receptive for the large class of gradient based optimization techniques (4) Introduce an *implicit* notion of time by comparing rates instead of fires (5) Computationally more powerful than 1st generation (6) Work well in OL Disadvantages: (1) Interpretation as spikes is not intuitive (2) Hebbian learning is not directly applicable since units are computing rates instead of fires (3) Moderate parallelization of computations (4) Moderate dealing with SSTD (5) Sensitive to parameter values (6) Large number of parameters that need to be optimized	CNNs, RNNs, etc.	(Hochreiter & Schmidhuber, 1997; Rumelhart, Hinton, Williams, et al., 1988; Werbos, 1982)
3rd gen.	[Real, Real]	Advantages: (1) Universal for digital computation (2) Introduce an *explicit* notion of time by comparing rates instead of fires (3) More biologically plausible than 2nd generation (4) Computationally powerful (5) Successful hardware implementations (i.e. neuromorphic VLSI) (6) Massive parallelization of computation (7) Excellent dealing with SSTD (8) Work well in OL Disadvantages: (1) Sensitive to parameter values (2) Large number of parameters that need to be optimized (3) No rigid theory yet (see Section 3.2) (4) Unknown behavior for different types of spatio-temporal data (5) The reduced intuition when gradient based optimization techniques are not available (discontinuity of spikes) (6) The biological insight is not sufficient to formulate strong theories (7) The lack of a coherent framework for the many different models and coding schemes	SNNs	(Maass, 1997; Ponulak & Kasinski, 2011)

to three different notions of mean firing rate (either an average over time, or an average over several repetitions of the experiment, or an average over a population of neurons respectively). *Temporal encoding* schemes are based on spike timing: "time-to-first-spike" (when a code for the timing of the first spike contains all information about the new stimulus), "phase" (when we can apply a "time-to-first-spike" encoding scheme also where the reference signal is not a single event, but a periodic signal), and "correlations and synchrony" (where we use spikes from other neurons as the reference signal for a spike code).

3.3. Spiking neuron models

A spiking neuron model is a mathematical description of the properties of neurons that generate electrical potentials across

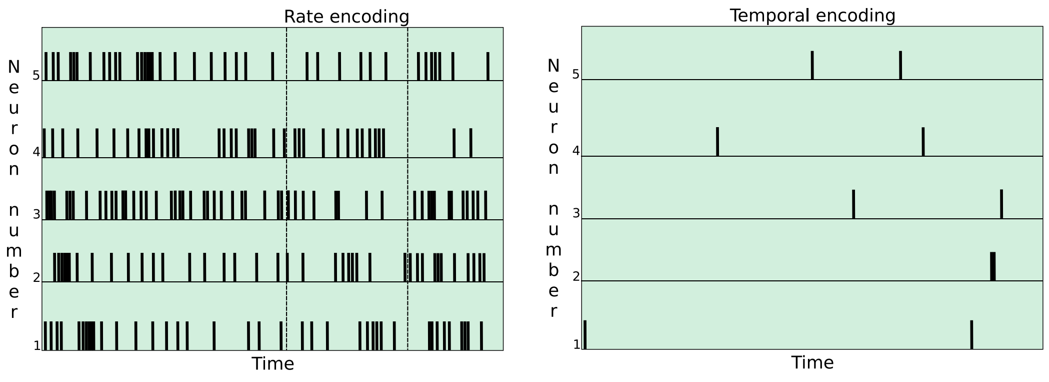
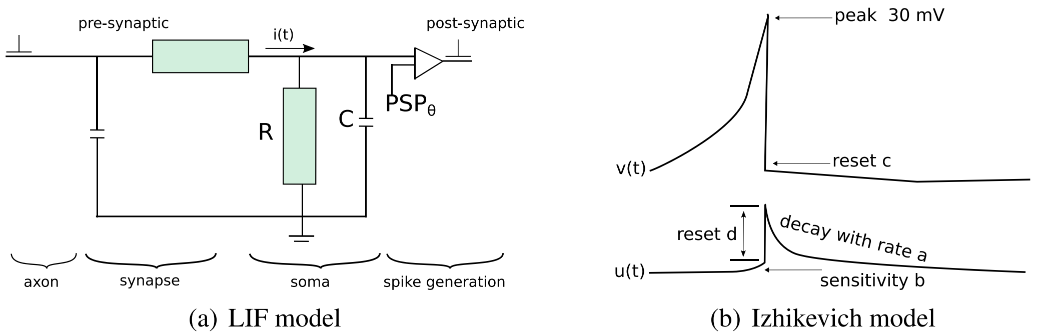


Fig. 5. *Temporal encoding* versus *Rate-based encoding*.



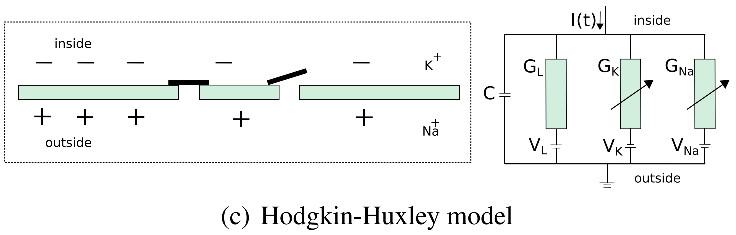


Fig. 6. Three of the most representative neuron models.

their cell membrane. Some of the most relevant neuron models are described subsequently, and graphically represented in Fig. 6.

Leaky Integrate-and-Fire (LIF) (Gerstner & Kistler, 2002) model, where a neuron is considered as an electrical circuit and the current potential is computed with an appropriate equation. Model consists of capacitor C in parallel with resistor R, driven by a current $I(t)$:

$$I(t) = I_R + I_{cap} \tag{2}$$

The standard form of the model is defined as:

$$\tau_m \frac{du}{dt} = -u(t) + RI(t), \tag{3}$$

where $\tau_m = RC$ is the membrane time constant. Here spikes are events characterized by a firing time t^f:

$$t^f : u(t^f) = \vartheta, \tag{4}$$

and after t^f, the potential is reset to a resting potential u_r. In a more general form, the LIF model can also include a refractory period, in which the dynamics are interrupted for an absolute time Δ^{abs}. LIF model is simple and computationally effective, and it is the most widely used spiking neuron model despite other more biologically realistic models. One of the most used approximations of the LIF model is the Spike Response Model (SRM), where each time a neuron receives an input from a previous neuron, its internal state (membrane potential) changes. In summary, a neuron emits a spike each time its membrane potential reaches a threshold value (θ) (see Fig. 4). Then, after emitting the spike, the neuron goes through a phase of high hyperpolarization during which it is impossible to emit a second spike for some time (refractory period).

Hodgkin–Huxley (Hodgkin & Huxley, 1952) model, where a semipermeable cell membrane separates the interior of the cell from the extracellular liquid, acting as a capacitor (C). When an input current $I(t)$ is injected into the cell, it may add further charge on C, or leak through the channels in the cell membrane. Because of active ion transport through the cell membrane, the ion concentration inside the cell is different from that in the extracellular liquid. The Nernst potential generated by the difference in ion concentration is represented by a battery. A detailed description of the influences of the conductance of three ion channels (Na, K and L) on the spike activity of the giant axon of squid is given by the following equation:

$$\sum_{ch} i_{ch}(t) = G_{Na}\cdot m^3\cdot h\cdot(v_c - v_{Na}) + G_K\cdot n^4\cdot(v_c - v_K)\cdot G_L\cdot(v_c - v_L), \tag{5}$$

and its differential equations:

$$\frac{dm}{dt} = \alpha_m(v_c) \cdot (1 - m) - \beta_m(v_C) \cdot m \quad (6)$$

$$\frac{dn}{dt} = \alpha_n(v_c) \cdot (1 - n) - \beta_n(v_C) \cdot n \quad (7)$$

$$\frac{dh}{dt} = \alpha_h(v_c) \cdot (1 - h) - \beta_h(v_C) \cdot h \quad (8)$$

In the equations, G_{Na}, G_K, G_L are the conductance of the sodium, potassium, and leakage channels; V_{Na}, V_K, and V_L are constants called reverse potentials; m and n control the Na channel and variable h controls the K channel; α and β are empirical functions of v_c.

Izhikevich (Izhikevich, 2007) model claims to be as biological plausible as the Hodgkin–Huxley model with computational efficiency of LIF models. It is defined by the following equations:

$$v' = 0.004v^2 + 5v + 140 - u + I \quad (9)$$

$$u' = a(bv - u) \quad (10)$$

Here it is considered that if $v \geq 30$ mV, then $v \longleftarrow c$, $u \longleftarrow u + d$, where a, b, c and d are parameters of the model, v represents the membrane potential, and u the membrane recovery.

3.4. Learning methods and algorithms in SNN

Synaptic plasticity is the capacity of synaptic connections to change their strength, which is the basis of the learning and memory processes in biological neural networks. Several synaptic plasticities coexist, mainly differing on the time scale and the conditions required for the induction. Regarding the time scale, some of them decay on the order of about 10–100 ms, while others, such as Long-Term Potentiation (LTP) or Long-Term Depression (LTD), persist during a longer time. Referred to the conditions for the induction, some synaptic plasticities depend only on the history of pre-synaptic stimulation (independently of the post-synaptic response), while others depend on the coincidence of pre-synaptic and post-synaptic activity, on the temporal order of their spikes, and on other factors (e.g. a concentration of specific chemicals).

In supervised learning, Supervised Hebbian Learning (SHL) (Legenstein, Naeger, & Maass, 2005) rule provides probably the most straightforward solution for from a biologically realistic view. Here, a spike-based Hebbian process is supervised by an additional "teaching" signal that reinforces the post-synaptic neuron to fire at the target times and to remain silent at other times. Remote Supervised Method (ReSuMe) (Ponulak, 2005) and SpikeProp (Bohte, Kok, & La Poutre, 2002) are two of the most representative algorithms in supervised learning for SNN. In unsupervised learning, by modifying synaptic strengths of Hebbian processes, the connections are reorganized within a neural network and, under certain conditions, may lead to an emergence of new functions (input clustering, pattern recognition, source separation, dimensionality reduction, formation of associative memories or self-organizing maps). It was demonstrated that the change in the synaptic efficacy after several repetitions of the experiment was a function of the relative differences of the spike times; and that whereas pre-synaptic spikes that precede post-synaptic ones induce potentiation, the reversed order of spikes induce synaptic depression. This phenomenon is called Spike-Time Dependent Plasticity (STDP) (Bi & Poo, 2001).

3.5. Applications

Many practical applications of SNNs can be found in the fields of motor control (e.g. DeWolf, Stewart, Slotine, & Eliasmith, 2016 for the motor control of the arm), robotics control based (Bing, Meschede, Röhrbein, Huang, & Knoll, 2018), trajectory tracking, decision making with application to financial market, spatial navigation and path planning, decision making and action selection, rehabilitation, image and odor recognition, spatial navigation and mental exploration of the environment, etc. (Ponulak & Kasinski, 2011).

It is worth mentioning that there is a great opportunity for the use of SNNs in the *Green Artificial Intelligence* paradigm (Villani, Bonnet, Rondepierre, et al., 2018), which is becoming an important subfield of Artificial Intelligence. It is based on the reduction of the usage of resources while training and testing our models, using techniques of energy efficiency, energy aware computing, hardware accelerators, and embedded systems. Here, the use of SNN in OL allows for very fast real-time simulations of large networks and a low computational cost.

3.6. Available hardware and software/frameworks

Again, we do not claim this subsection to be an entire list, but Table 3 offers some of the most known SNNs software and hardware platforms to get started.

3.7. Recent trends and challenges

Previous subsections have been underlined that SNNs are biologically more realistic than traditional ANNs, and computationally more powerful. The question that arises here is why community still works with continuous neural networks instead of the theoretically superior SNNs. Now, we present some of the most relevant challenges for the wide use of SNNs, and some of their future trends:

Which model to apply?: there is no unified framework or an agreed upon base neuron model of SNNs yet. Different models are often applied, which makes tough the task of carrying out insightful comparisons. Besides, it is well known the trade-off between the biological plausibility and the computational cost of the model: the model choice ranges from LIF (simple but efficient, they are usually chosen by computer scientists and engineers) to the Hodgkin–Huxley (sophisticated but slow, they are usually chosen by neuroscience researchers).

Information encoding: although it is already known that brains do not work with real numbers but with timed spikes, the crux of the issue is how information is encoded with such spikes, being one of the big unresolved challenges of neuroscience. Here we find again a lack of agreement on what constitutes a good encoding scheme.

Learning based on spike timing: how to achieve a good learning algorithm is still one of the biggest challenges of SNNs, mainly motivated by two reasons. On the one hand, the biological learning method of brains is not still well understood, so it is hard to imitate it with artificial learning methods. On the other hand, the discontinuous nature

Table 3
Well-known SNNs software (S) and hardware (H) platforms.

Platform	S/H	Language	Description	References
Brian	S	Python	It is an open source simulator for SNN	(Goodman & Brette, 2008), http://briansimulator.org/
Cypress	S	C++	It is a simulation environment that can perform efficient discrete event simulations of SNNs with LIF models, as well as hybrid simulations involving both LIF neurons and cells with voltage-gated conductances. It is especially recommended for simulations that involve cells with complex anatomical and biophysical properties	(Carnevale & Hines, 2006)
Neuron	S	Python	It is a simulation environment that can perform efficient discrete event simulations of SNNs with LIF models, as well as hybrid simulations involving both LIF neurons and cells with voltage-gated conductances. It is especially recommended for simulations that involve cells with complex anatomical and biophysical properties	(Carnevale & Hines, 2006)
Nest	S	C++, Python, Cython	It is a simulation environment best suited for models that focus on the dynamics, size, and structure of neural systems rather than on the detailed morphological and biophysical properties of individual neurons	(van Albada et al., 2018)
PyNN	S	Python	It is for simulator-independent specification of neuronal network models. Once the model has been implemented with PyNN API and Python, we can run it without modification on any simulator that PyNN supports (Neuron, Nest, Brian) or on the supported neuromorphic hardware systems, such as SpiNNaker and BrainScaleS	http://neuralensemble.org/PyNN/
NeuCube	S	Java	It is a 3D evolving Spiking Neural Networks (eSNN) computational neurogenetic model to map, learn and mine brain data	(Kasabov, 2014)
PCSIM	S	Written in Python (although its computational core is written in C++)	It is a software package for simulation of neural circuits primarily designed for distributed simulation of large scale networks of SNNs	(Pecevski, Natschläger, & Schuch, 2009)
ANNarchy	S	The core of the library is written in C++, but it provides an interface in Python for the definition of the network	It is a neural simulator designed for distributed rate-coded or SNNs (or both)	(Vitay, Dinkelbach, & Hamker, 2015)
SpiNNaker	H	–	It is a massively parallel computing platform mainly targeted towards neuroscience, robotics, and computer science. It is based on numerical models running in real time on custom digital multi-core chips using the ARM architecture. Next generation small scale test chips of the SpiNNaker architecture is available for first test users since early 2018	(Furber, Galluppi, Temple, & Plana, 2014)
BrainScaleS	H	–	It is a project that aims at understanding and emulating function and interaction of multiple spatial and temporal scales in brain information processing. It is based on physical (analogue or mixed-signal) emulations of neuron, synapse and plasticity models with digital connectivity, running up to 10 thousand times faster than real time. Next generation small scale test chips of the BrainScaleS architecture are available for first test users since early 2018	https://brainscales.kip.uni-heidelberg.de/index.html

of spikes makes difficult the design of calculus based approaches.

Reinforcement learning: In contrast with supervised learning, instructions are not always provided together with data, but are often given implicitly at a later point in time in the form of rewards, or penalties (negative rewards). This is the field of reinforcement learning, common in particular in the field of robotics, which could be seen as particularly relevant for SNNs because of this domain as it often requires very fast processing of large amounts of input data (e.g. from video and other sensors). SNNs can be very powerful and efficient in theory making them potential candidates for use in robotics and related disciplines once conventional methods fail to improve their speed below the required limits. Yet, this is just speculation at this point as both algorithms and hardware necessary for very fast SNN implementation are still not around.

4. SNNs in online learning scenarios

Data streams may exhibit temporal dependencies between class labels, which formally occur whenever the current sample label y^t is influenced by previous sample labels (y^{t-1}, y^{t-2}, …). Temporal dependence can help to determine how the input features relate with each other over time. SNNs leverage spike information representation so as to build spike-time learning rules that have shown the ability to capture temporal associations between temporal variables in streaming data. Additionally, STDP and Hebbian learning are biologically plausible local learning rules in SNN models; the use of some SNNs (e.g. eSNNs) in OL allows for a very fast real-time and reducing the computational complexity of the learning process, given its locality which lends itself well for parallel implementation. In terms of adaptation to the drift, most off-the-shelf classification models need to be retrained if they are used in changing environments, and fail to scale properly. Some SNNs can overcome this drawback, e.g. the evolving nature of eSNNs (based on the merging process of similar neurons) makes possible the accumulation of knowledge as data become available, without the requirement of storing and retraining the model with past samples. Finally, they have also shown to be very competitive as concept drift detectors (Lobo, Del Ser et al., 2018).

4.1. Existing SNN approaches for online learning: Drawbacks and trade-offs

To date, a number of supervised and unsupervised learning methods have been developed for SNNs. Despite a review of some of these methods can be found in Kasiński and Ponulak (2006), Wang, Belatreche, Maguire, and McGinnity (2010), here we provide an updated summary of them, firstly introducing each method, and then moving on to a comparison of them.

SpikeProp (Bohte et al., 2002) is a learning rule based on gradient descent for training SNNs. As Table 4 shows, it presents some drawbacks that were overcome in McKennoch, Liu, and Bushnell (2006), Schrauwen and Van Campenhout (2004). The Supervised Hebbian Learning rule (SHL) (Legenstein et al., 2005) is a spike-based Hebbian process that is supervised by an additional "teaching" signal that reinforces the post-synaptic neuron to fire at the target times and to remain silent at other times. It is suitable for training in single-layer networks, and it is more recommendable the use of multi-layer feedforward or recurrent neural networks for many other tasks because they are capable of performing more complex computation than single-layer networks (Ponulak & Kasinski, 2011). In ReSuMe (Ponulak, 2005) an instructive signal modulates synaptic plasticity not to have a marginal direct effect on the post-synaptic somatic membrane potential. Despite this method was claimed to be suitable for OL, the network structure used is fixed and does not adapt to incoming stimuli. SHL and ReSuMe are suitable for training in single-layer networks, and it is more recommendable the use of multi-layer feedforward or recurrent neural networks for many other tasks because they are capable of performing more complex computation than single-layer networks (Ponulak & Kasinski, 2011). Tempotron (Gütig & Sompolinsky, 2006), which uses a gradient-descent approach as learning rule, is used to train LIF neurons to distinguish between two classes of patterns by firing at least one action potential or by remaining quiescent. It is equivalent to a special case of ReSuMe under certain conditions. In Chronotron (Florian, 2012), which was developed to improve on the Tempotron, two supervised learning rules were proposed to allow neurons to fire spikes at the desired times by using an error function to calculate the distance between the actual and desired spike times. More recently, SpikeTemp method proposed in Wang, Belatreche, Maguire, and McGinnity (2017) offers an enhanced rank-order-based learning method for SNNs with an adaptive structure where the precise times of incoming spikes are used to determine the required change in synaptic weights.

Despite most of them claim to work for OL, we would like to clarify a few things, however. Until the arrival of SpikeTemp, the above-mentioned approaches use an SNN with a fixed structure, where the sizes of the hidden and output layers must be specified beforehand, and they are trained in an offline mode. Thus, they can only be applied when the number of classes is known beforehand. Besides, these approaches cannot be applied to problems where data are continuously changing as they will need to be retrained. However, biological neural networks are known for their ability to learn continuously and incrementally which account for their continuous adaptation to changing non-stationary environments. Therefore, to allow an SNN to interact with such dynamic environments, it is required that both its structure and weights dynamically adapt to new data. In addition, catastrophic forgetting should be avoided when new information is learned.

Aside from SpikeTemp, one of the most promising SNNs for OL is the evolving Spiking Neural Network (eSNN) (Schliebs & Kasabov, 2013), based on the Thorpe model (Thorpe, Delorme, & Van Rullen, 2001) which is a simplified version of LIF model (it simplifies the leaky operation of the computational neuron). It is suitable for those scenarios where stringent constraints on computational cost and processing time prevail. Recently, it has been modified in Lobo, Laña et al. (2018) to improve their adaptation to the drift, and to consider OL in a more realistic form by limiting the size of the neuron repository, avoiding its incremental growing, which is unfeasible in OL scenarios. Also in Dora, Subramanian, Suresh, and Sundararajan (2016) to add a self-regulating mechanism (SRESN) of its learning process, or in Wang, Belatreche, Maguire, and McGinnity (2015) with SpikeComp to introduce an adaptive compact structure for SNNs.

Finally, Table 4 presents the drawbacks and trade-offs of these SNN approaches for OL. To date, with these few exceptions (to the best of our knowledge), there is a lack of efficient and scalable SNN-based algorithms for OL scenarios.

4.2. Topics of future interest

Next, we summarize some of the most interesting topics for the future in the field of SNNs and OL:

***Lifelong Machine Learning* or *Continual Learning*:** it involves the capability of the model to smoothly update its captured knowledge to take simultaneously into account different tasks and data distributions, yet still being able to reuse and retain captured knowledge and skills recurrently occurring over time (Chen & Liu, 2016). Hence, it is a paradigm requiring notably higher time scales where data (and tasks) become available only during certain periods of time and probably with no access to previously seen data. Thereby, it is imperative to build on top of previously learned knowledge, from where the connection between this paradigm and OL emerges.

Deep SNN learning: future research should address deep learning of spatio-temporal patterns for streaming data, deep knowledge representation and its online adaptation. Concretely, two issues would deserve special attention:

- what has the SNN learned and how it is changing during OL?,
- how this can be represented as new information or knowledge?

Some related research has been carried out in this direction in (Kasabov, 2018; Tavanaei et al., 2018). Regarding the building of deep SNNs, we highlight the efforts recently invested in (Wu et al., 2019).

Human–computer interaction: a particularly interesting field for future application of SNNs in OL is of human–computer interaction where biological spiking neurons need to communicate with software in a potentially very rapid manner.

Others: parameter optimization in evolving scenarios, visualization of SNN models, spatial mapping of input variables into a SNN architecture, neuromorphic hardware systems for real-time applications, etc., should be tackled in a near future.

Table 4
Existing SNN implementations for OL: drawbacks and trade-offs.

SpikeProp	*Advantages*	(1) It is able to solve complex classification problems (2) It is computationally powerful
	Drawbacks	(1) There is no mechanism to "prop-up" synaptic weights once the postsynaptic neuron no longer fires for any input pattern (2) Only the first spike produced by a neuron is relevant and the rest of the time course of the neuron is ignored. Whenever a neuron fires a single spike, it is not allowed to fire again. For this reason the method is suitable to implement only the "time-to-first-spike" coding scheme (3) It is too slow to be used in OL (4) The large number of synaptic connections makes difficult to scale up when a high dimensional dataset is considered (5) They are often regarded as non-biologically plausible because they require a non-local spread of error signals from one synapse to another (6) Neither the original `SpikeProp` method nor any of the proposed modifications allow learning patterns composed of more than one spike per neuron
SHL	*Advantages*	(1) It provides probably the most straightforward solution for from a biologically realistic view
	Drawbacks	(1) As during training ("teaching") signal currents suppress all undesired fires, the only correlations of pre-synaptic and post-synaptic activities happen around the target firing times. In other occasions, this correlation is not present, and there is no mechanism to weaken these synaptic weights that make the neuron to fire at undesired times during the testing phase (2) synapses change their weights even when the neuron fires already exactly at the desired times. Therefore, `SHL` can achieve stable solutions only by adding additional constraints or more learning rules
ReSuMe	*Advantages*	(1) It solves the problems of `SpikeProp` and `SHL` rule (2) It is independent of the neuron models used (3) It can efficiently learn the desired temporal and spatio-temporal patterns of spikes (4) The learning process converges quickly
	Drawbacks	(1) Despite it claims to be suitable for OL, the network structure used is fixed and does not adapt to incoming stimuli (2) It is unable to predict inputs after just one presentation of the training samples (3) Although it uses STDP, which is biologically plausible, its local nature limits its learning capacity
SpikeTemp	*Advantages*	(1) It is able to predict inputs after just one presentation of the training samples (2) It is scalable for a wide range of datasets with various dimensions and numbers of classes (3) it is more efficient than the existing rank-order learning method
	Drawbacks	(1) The time cost increases compared with the rank-order-based approach
Tempotron	*Advantages*	(1) It can learn to categorize a broad range of input classes in which category information is not encoded in spike count but, rather, in the latencies of single spikes or in pairwise and higher-order patterns of synchrony
	Drawbacks	(1) It can only be applied to single-layered networks (2) It is restricted to outputting 0 or 1 spikes during a predetermined interval, and thus the output did not encode spike timing information
Chronotron	*Advantages*	(1) Its innovation is to base the supervised training on a more sophisticated distance measure, VictorPurpora (VP) distance metric, between two spike trains
	Drawbacks	(1) The strengths of the synapses are trained in a batch mode with a fixed network structure which makes it unsuitable for OL
eSNN	*Advantages*	(1) Its neural model allows for a very fast real-time simulation of large networks and a low computational cost (2) Their evolving nature (spiking neurons evolves incrementally over time to infer temporal patterns from data) allows to accumulate knowledge as data arrive, without storing and retraining the model with past data (3) They can serve as drift detectors in non-stationary environments (Lobo, Del Ser et al., 2018) (4) They are recommendable for non-stationary environments, because changes in the input stream data are encoded immediately as binary events or spikes, which is one of the most suitable data encoding strategies for adapting to drifts
	Drawbacks	(1) Its neuron repository grows indefinitely with each new arriving sample in OL (2) they update/merge neurons by averaging weights estimated using rank order. It is possible that two input patterns have the same rank (spikes occur in the same order) but their precise spike times are far apart. This may result in an SNN with more number of neurons and loss of knowledge, stored in the network

5. Conclusions

In this work we have analyzed the OL and SNNs fields from an introductory perspective to serve as an entry point for the application of SNNs to OL, which is a very hot topic in the research community due to the large number of real applications based on stream data, even more in those scenarios where data are affected by non-stationary events, provoking the so-called concept drift. SNNs are considered the third generation of neural networks, and have revealed themselves as one of the most successful approaches to model the behavior and learning potential of the brain, allowing for a very fast real-time simulation of large networks and a low computational cost. They have also shown a very good behavior in drift detection and drift adaptation situations, often present in OL scenarios. All of this leads us to consider both fields in an incredibly interesting intersection. Still, much progress has to be made in both fields for tackling their respective open challenges, but we should be aware of the importance of merging OL and SNNs in order to solve real problems with the computational power of these bio-inspired systems.

Acknowledgments

This work was supported by Basque Government (Spain) through the project *VIRTUAL* (KK-2018/00096).

References

Abeles, M., Bergman, H., Margalit, E., & Vaadia, E. (1993). Spatiotemporal firing patterns in the frontal cortex of behaving monkeys. *Journal of Neurophysiology*, *70*(4), 1629–1638.

Bair, W. (1994). Reliable temporal modulation in cortical spike trains in the awake monkey. In *Proceedings of the symposium on dynamics of neural processing*.

Barros, R. S. M., & Santos, S. G. T. C. (2018). A large-scale comparison of concept drift detectors. *Information Sciences*, *451*, 348–370.

Benczúr, A. A., Kocsis, L., & Pálovics, R. (2018). Online machine learning in big data streams. *CoRR*, abs/1802.05872.

Bi, G.-q., & Poo, M.-m. (2001). Synaptic modification by correlated activity: Hebb's postulate revisited. *Annual Review of Neuroscience*, *24*(1), 139–166.

Bifet, A., Gavaldà, R., Holmes, G., & Pfahringer, B. (2018). *Machine learning for data streams with practical examples in MOA*. MIT Press, https://moa.cms.waikato.ac.nz/book/.

Bing, Z., Meschede, C., Röhrbein, F., Huang, K., & Knoll, A. C. (2018). A survey of robotics control based on learning-inspired spiking neural networks. *Frontiers in Neurorobotics*, *12*, 35.

Bohte, S. M., Kok, J. N., & La Poutre, H. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, *48*(1–4), 17–37.

Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., & Tzoumas, K. (2015). Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, *36*(4).

Carnevale, N. T., & Hines, M. L. (2006). *The NEURON book*. Cambridge University Press.

Chen, Z., & Liu, B. (2016). Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *10*(3), 1–145.

De Francisci Morales, G., Bifet, A., Khan, L., Gama, J., & Fan, W. (2016). Iot big data stream mining. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM.

Del Ser, J., Osaba, E., Molina, D., Yang, X.-S., Salcedo-Sanz, S., Camacho, D., et al. (2019). Bio-inspired computation: Where we stand and what's next. *Swarm and Evolutionary Computation* , *48*, 220–250.

DeWolf, T., Stewart, T. C., Slotine, J.-J., & Eliasmith, C. (2016). A spiking neural model of adaptive arm control. *Proceedings of the Royal Society B*, *283*(1843), 20162134.

Diez-Olivan, A., Del Ser, J., Galar, D., & Sierra, B. (2019). Data fusion and machine learning for industrial prognosis: Trends and perspectives towards industry 4.0. *Information Fusion*, *50*, 92–111.

Dora, S., Subramanian, K., Suresh, S., & Sundararajan, N. (2016). Development of a self-regulating evolving spiking neural network for classification problem. *Neurocomputing*, *171*, 1216–1229.

Florian, R. V. (2012). The chronotron: A neuron that learns to fire temporally precise spike patterns. *PLoS One*, *7*(8), e40233.

Foundation, A. S. (2012). Apache spark streaming. https://spark.apache.org/streaming. (Accessed 30 May 2019).

Foundation, A. S. (2017). Apache beam. https://beam.apache.org. (Accessed 30 May 2019).

Furber, S. B., Galluppi, F., Temple, S., & Plana, L. A. (2014). The spinnaker project. *Proceedings of the IEEE*, *102*(5), 652–665.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, *46*(4), 44.

Gerstner, W., & Kistler, W. M. (2002). *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge University Press.

Gomes, H. M., Barddal, J. P., Enembreck, F., & Bifet, A. (2017). A survey on ensemble learning for data stream classification. *ACM Computing Surveys*, *50*(2), 23.

Goodman, D. F., & Brette, R. (2008). Brian: A simulator for spiking neural networks in python. *Frontiers in Neuroinformatics*, *2*, 5.

Gütig, R., & Sompolinsky, H. (2006). The tempotron: A neuron that learns spike timing–based decisions. *Nature Neuroscience*, *9*(3), 420.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, *313*(5786), 504–507.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Hodgkin, A. L., & Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, *117*(4), 500–544.

Iqbal, M. H., & Soomro, T. R. (2015). Big data analysis: Apache storm perspective. *International Journal of Computer Trends and Technology*, *19*(1), 9–14.

Izhikevich, E. M. (2007). *Dynamical systems in neuroscience*. MIT Press.

Kasabov, N. K. (2014). Neucube: A spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data. *Neural Networks*, *52*, 62–76.

Kasabov, N. (2018). *Time-space, spiking neural networks and brain-inspired artificial intelligence*. Springer.

Kasabov, N. (2019). Deep learning of multisensory streaming data for predictive modelling with applications in finance, ecology, transport and environment. In *Time-space, spiking neural networks and brain-inspired artificial intelligence* (pp. 619–658). Berlin, Heidelberg: Springer.

Kasiński, A., & Ponulak, F. (2006). Comparison of supervised learning methods for spike time coding in spiking neural networks. *International Journal of Applied Mathematics and Computer Science*, *16*, 101–113.

Khamassi, I., Sayed-Mouchaweh, M., Hammami, M., & Ghédira, K. (2018). Discussion and review on evolving data streams and concept drift adapting. *Evolving Systems*, *9*(1), 1–23.

Krawczyk, B., Minku, L. L., Gama, J., Stefanowski, J., & Woźniak, M. (2017). Ensemble learning for data stream analysis: A survey. *Information Fusion*, *37*, 132–156.

Legenstein, R., Naeger, C., & Maass, W. (2005). What can a neuron learn with spike-timing-dependent plasticity? *Neural Computation*, *17*(11), 2337–2382.

Lobo, J. L., Del Ser, J., Bilbao, M. N., Perfecto, C., & Salcedo-Sanz, S. (2018). Dred: an evolutionary diversity generation method for concept drift adaptation in online learning environments. *Applied Soft Computing*, *68*, 693–709.

Lobo, J. L., Del Ser, J., Laña, I., Bilbao, M. N., & Kasabov, N. (2018). Drift detection over non-stationary data streams using evolving spiking neural networks. In *International symposium on intelligent and distributed computing* (pp. 82–94). Springer.

Lobo, J. L., Laña, I., Del Ser, J., Bilbao, M. N., & Kasabov, N. (2018). Evolving spiking neural networks for online learning over drifting data streams. *Neural Networks*, *108*, 1–19.

Losing, V., Hammer, B., & Wersing, H. (2018). Incremental on-line learning: A review and comparison of state of the art algorithms. *Neurocomputing*, *275*, 1261–1274.

Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., & Zhang, G. (2018). Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*.

Maass, W. (1997). Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, *10*(9), 1659–1671.

Manyika, J., Chui, M., Bisson, P., Woetzel, J., Dobbs, R., Bughin, J., et al. (2015). *Unlocking the potential of the internet of things*. McKinsey.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, *5*(4), 115–133.

McKennoch, S., Liu, D., & Bushnell, L. G. (2006). Fast modifications of the spikeprop algorithm. In *The 2006 IEEE international joint conference on neural network proceedings* (pp. 3970–3977). IEEE.

Middleton, P., Tsai, T., Yamaji, M., Gupta, A., & Rueb, D. (2017). *Forecast: Internet of things — Endpoints and associated services, worldwide*. Gartner.

Minku, L. L. (2011). *Online ensemble learning in the presence of concept drift* (Ph.D. thesis), University of Birmingham.

Montiel, J., Read, J., Bifet, A., & Abdessalem, T. (2018). Scikit-multiflow: A multi-output streaming framework. *Journal of Machine Learning Research (JMLR)*, *19*(72), 1–5, http://jmlr.org/papers/v19/18-251.html.

Morales, G., & Bifet, A. (2015). SAMOA: Scalable advanced massive online analysis. *Journal of Machine Learning Research (JMLR)*, *16*(1), 149–153.

Noghabi, S. A., Paramasivam, K., Pan, Y., Ramesh, N., Bringhurst, J., Gupta, I., et al. (2017). Samza: Stateful scalable stream processing at linkedin. *Proceedings of the VLDB Endowment*, *10*(12), 1634–1645.

Pecevski, D., Natschläger, T., & Schuch, K. (2009). PCSIM: A parallel simulation environment for neural circuits fully integrated with python. *Frontiers in Neuroinformatics*, *3*, 11.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research (JMLR)*, *12*, 2825–2830.

Petro, B., Kasabov, N., & Kiss, R. M. (2019). Selection and optimization of temporal spike encoding methods for spiking neural networks. *IEEE transactions on neural networks and learning systems*.

Ponulak, F. (2005). *Resume-new supervised learning method for spiking neural networks, (vol. 42)*. Institute of Control and Information Engineering, Poznan University of Technology.

Ponulak, F., & Kasinski, A. (2011). Introduction to spiking neural networks: Information processing, learning and applications. *Acta Neurobiologiae Experimentalis*, *71*(4), 409–433.

Ramírez-Gallego, S., Krawczyk, B., García, S., Woźniak, M., & Herrera, F. (2017). A survey on data preprocessing for data stream mining: Current status and future directions. *Neurocomputing*, *239*, 39–57.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386.

Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1988). Learning representations by back-propagating errors. *Cognitive Modeling*, *5*(3), 1.

Schliebs, S., & Kasabov, N. (2013). Evolving spiking neural network – a survey. *Evolving Systems*, *4*(2), 87–98.

Schrauwen, B., & Van Campenhout, J. (2004). Improving Spikeprop: Enhancements to an error-backpropagation rule for spiking neural networks. In *Proceedings of the 15th ProRISC workshop*. vol. 11 (pp. 301–305).

Soltic, S., & Kasabov, N. (2010). Knowledge extraction from evolving spiking neural networks with rank order population coding. *International Journal of Neural Systems*, *20*(6), 437–445.

Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., & Maida, A. (2018). Deep learning in spiking neural networks. *Neural Networks*.

Thorpe, S., Delorme, A., & Van Rullen, R. (2001). Spike-based strategies for rapid processing. *Neural Networks*, *14*(6–7), 715–725.

van Albada, S. J., Rowley, A. G., Senk, J., Hopkins, M., Schmidt, M., Stokes, A. B., et al. (2018). Performance comparison of the digital neuromorphic hardware spinnaker and the neural network simulation software NEST for a full-scale cortical microcircuit model. *Frontiers in Neuroscience*, *12*, 291.

Villani, C., Bonnet, Y., Rondepierre, B., et al. (2018). *For a meaningful artificial intelligence: Towards a French and European strategy*. Conseil national du numérique.

Vitay, J., Dinkelbach, H. Ü., & Hamker, F. H. (2015). Annarchy: A code generation approach to neural simulations on parallel hardware. *Frontiers in Neuroinformatics*, *9*, 19.

Vogelstein, R. J., Mallik, U., Culurciello, E., Cauwenberghs, G., & Etienne-Cummings, R. (2005). Saliency-driven image acuity modulation on a reconfigurable array of spiking silicon neurons. In *Advances in neural information processing systems* (pp. 1457–1464).

Wang, J., Belatreche, A., Maguire, L., & McGinnity, M. (2010). Online versus offline learning for spiking neural networks: A review and new strategies. In *2010 IEEE 9th International conference on cyberntic intelligent systems* (pp. 1–6). IEEE.

Wang, J., Belatreche, A., Maguire, L. P., & McGinnity, T. M. (2015). Spikecomp: An evolving spiking neural network with adaptive compact structure for pattern classification. In *International conference on neural information processing* (pp. 259–267). Springer.

Wang, J., Belatreche, A., Maguire, L. P., & McGinnity, T. M. (2017). Spiketemp: An enhanced rank-order-based learning approach for spiking neural networks with adaptive structure. *IEEE Transactions on Neural Networks and Learning Systems*, *28*(1), 30–43.

Wang, S., Minku, L. L., & Yao, X. (2018). A systematic study of online class imbalance learning with concept drift. *IEEE Transactions on Neural Networks and Learning Systems*, (99), 1–20.

Webb, G. I., Hyde, R., Cao, H., Nguyen, H. L., & Petitjean, F. (2016). Characterizing concept drift. *Data Mining and Knowledge Discovery*, *30*(4), 964–994.

Werbos, P. J. (1982). Applications of advances in nonlinear sensitivity analysis. In *System modeling and optimization* (pp. 762–770). Springer.

Wu, J., Chua, Y., Zhang, M., Yang, Q., Li, G., & Li, H. (2019). Deep spiking neural network with spike count based learning rule. arXiv preprint arXiv:1902.05705.

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., et al. (2016). Apache spark: A unified engine for big data processing. *Communications of the ACM*, *59*(11), 56–65.

Žliobaitė, I., Pechenizkiy, M., & Gama, J. (2016). An overview of concept drift applications. In *Big data analysis: New algorithms for a new society* (pp. 91–114). Springer.