Neural Networks 41 (2013) 188–201

Contents lists available at SciVerse ScienceDirect

Neural Networks

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

2013 Special Issue

Dynamic evolving spiking neural networks for on-line spatio- and spectro-temporal pattern recognition

Nikola Kasabov [a,b,c,*], Kshitij Dhoble [a], Nuttapod Nuntalid [a], Giacomo Indiveri [b,c]

[a] Knowledge Engineering & Discovery Research Institute (KEDRI), Auckland University of Technology, New Zealand

[b] Institute of Neuroinformatics (INI), University of Zurich, Switzerland

[c] ETH Zurich, Switzerland

ARTICLE INFO

Keywords:
Spatio-temporal pattern recognition
Spiking neural networks
Dynamic synapses
Evolving connectionist systems
Rank-order coding
Spike time based learning
Moving object recognition
EEG pattern recognition

ABSTRACT

On-line learning and recognition of spatio- and spectro-temporal data (SSTD) is a very challenging task and an important one for the future development of autonomous machine learning systems with broad applications. Models based on spiking neural networks (SNN) have already proved their potential in capturing spatial and temporal data. One class of them, the evolving SNN (eSNN), uses a one-pass rank-order learning mechanism and a strategy to evolve a new spiking neuron and new connections to learn new patterns from incoming data. So far these networks have been mainly used for fast image and speech frame-based recognition. Alternative spike-time learning methods, such as Spike-Timing Dependent Plasticity (STDP) and its variant Spike Driven Synaptic Plasticity (SDSP), can also be used to learn spatio-temporal representations, but they usually require many iterations in an unsupervised or semi-supervised mode of learning. This paper introduces a new class of eSNN, dynamic eSNN, that utilise both rank-order learning and dynamic synapses to learn SSTD in a fast, on-line mode. The paper also introduces a new model called deSNN, that utilises rank-order learning and SDSP spike-time learning in unsupervised, supervised, or semi-supervised modes. The SDSP learning is used to evolve *dynamically* the network changing connection weights that capture spatio-temporal spike data clusters both during training and during recall. The new deSNN model is first illustrated on simple examples and then applied on two case study applications: (1) moving object recognition using address-event representation (AER) with data collected using a silicon retina device; (2) EEG SSTD recognition for brain–computer interfaces. The deSNN models resulted in a superior performance in terms of accuracy and speed when compared with other SNN models that use either rank-order or STDP learning. The reason is that the deSNN makes use of both the information contained in the order of the first input spikes (which information is explicitly present in input data streams and would be crucial to consider in some tasks) and of the information contained in the timing of the following spikes that is learned by the dynamic synapses as a whole spatio-temporal pattern.

1. Introduction

Spatio- and spectro-temporal data (SSTD), that are characterised by a strong temporal component, are the most common types of data collected in many domain areas, including engineering (e.g. speech and video), bioinformatics (e.g. gene and protein expression), neuroinformatics (e.g. EEG, fMRI), ecology (e.g. establishment of species), environment (e.g. the global warming process), medicine (e.g. patients risk of disease or recovery over time), economics (e.g. financial time series, macroeconomics), etc. However, there is lack of efficient methods for modelling such data and for spatio-temporal pattern recognition that can facilitate new discoveries from complex SSTD and produce more accurate prediction of spatio-temporal events in autonomous machine learning systems.

The brain-inspired spiking neural networks (SNN) (e.g.: Belatreche, Maguire, & McGinnity, 2006; Brette et al., 2007; Gerstner, 1995; Hodgkin & Huxley, 1952; Izhikevich, 2004, 2006; Kistler & Gerstner, 2002; Maass & Zador, 1999), considered now as the third generation of neural networks, are a promising paradigm as these new generation of computational models are potentially capable of modelling complex information processes due to their ability to represent and integrate different information dimensions, such as time, space, frequency, phase, and to deal with large volumes of

* Corresponding author at: Knowledge Engineering & Discovery Research Institute (KEDRI), Auckland University of Technology, New Zealand. Tel.: +64 9 921 9506; fax: +64 9 921 9501.

E-mail addresses: nkasabov@aut.ac.nz (N. Kasabov), kdhoble@aut.ac.nz (K. Dhoble), nnuntalid@aut.ac.nz (N. Nuntalid), giacomo@ini.phys.ethz.ch (G. Indiveri).

0893-6080/$ – see front matter
doi:10.1016/j.neunet.2012.11.014

data in an adaptive and self-organising manner using information representation as trains of spikes.

With the development of new techniques to capture SSTD in a fast on-line mode, e.g.: address event representation (AER) devices, such as the artificial retina (Delbruck, 2007; Lichtsteiner & Delbruck, 2005) and artificial cochlea (van Schaik & Liu, 2005), the available wireless EEG equipment and with the advanced SNN hardware technologies (Indiveri, Chicca, & Douglas, 2009; Indiveri et al., 2011; Indiveri, Stefanini, & Chicca, 2010), new opportunities have been created, but this still requires efficient and suitable methods.

Thorpe and Gautrais (1998) introduced rank-order (RO) learning to achieve fast, one-pass learning of static patterns using one spike per synapse. This method was successfully used for image recognition (Thorpe, Guyonneau, Guilbaud, Allegraud, & Vanrullen, 2004). In Thorpe, Brilhault, and Perez-Carrasco (2010) RO was applied with the AER spiking encoding method for a fast image processing (static patterns) and reconstruction. The RO learning principle was also applied for a class of SNN, called evolving SNN (eSNN) (Kasabov, 2007; Wysoski, Benuskova, & Kasabov, 2010).

Guyonneau, VanRullen, and Thorpe (2005) demonstrated that a neuron using STDP tends to organise its synaptic weights to respond to earlier spikes. Masquelier, Guyonneau and Thorpe (Masquelier, Guyonneau, & Thorpe, 2008) demonstrated that a single LIF neuron with simple synapses can be trained with the STDP unsupervised learning rule to discriminate a repeating pattern of synchronised spikes on certain synapses from noise. The training requested hundreds iterations and the more the training was repeated, the earlier the beginning of the synchronised spiking pattern was detected from the input stream. This work was continued with multiple neurons, connected to each other with winner-takes-all connections, to respond to different repeating patterns from a common input stream Masquelier, Guyonneau, and Thorpe (2009).

Here we propose a combination of RO learning and a type of STDP unsupervised learning (SDSP—Spike Driven Synaptic Plasticity, Fusi, Annunziato, Badoni, Salamon, & Amit, 2000), so that a LIF neuron learns to recognise whole spatio-temporal pattern using only one iteration of training-on-line mode. For this purpose, the neuron first ‘utilises’, through RO learning, extra information given to it—the order of the incoming spikes (rather than learning this information in an unsupervised STDP mode using many iterations as in the works cited above) and then the neuron tunes the initial connection weights through SDSP learning over the rest of the spatio-temporal pattern. As the order of spikes is an information that can be easily calculated in a computational model, especially when using AER data, it makes sense to ‘free’ the spiking neurons from this task, that would require hundreds of learning repetitions, and make the learning process just one-pass, even for complex and large spatio-temporal patterns. Here, again, one neuron is dedicated to learn one pattern, but merging of neurons is also explained. One version of the deSNN — deSNNm, uses the neuronal Post Synaptic Potentials (PSP) to identify the winning neuron, similar to other implementations. But another version — deSNNs, defines the winning neuron based on a comparison between the synaptic weights over time.

The paper presents in Section 2 the principles of eSNN. In Section 3 temporal spike learning and more specifically — STDP and SDSP rules are presented. The new SNN model — dynamic evolving SNN (deSNN) is introduced in Section 4. The deSNN is first illustrated with simple examples and then demonstrated on a moving object classification problem, where AER data was collected using a silicon retina device (see Section 5). A second case study is the recognition of frame-based EEG SSTD (Section 6). The data used in both case study experiments is noisy by nature due to the characteristics of the processes and the measurement.

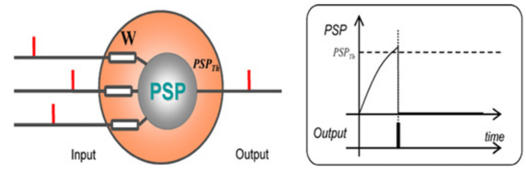

Fig. 1. Integrate-and-fire neuron with RO learning.

A comparative analysis of results between eSNN, deSNN, and a SNN that uses only SDSP learning rule, shows the advantage of the proposed deSNN model in terms of fast and accurate learning of both AER and frame-based SSTD. Section 7 discusses the implementation of the deSNN in a neuromorphic environment and Section 8 presents future directions.

2. Evolving connectionist systems (ECOS) and evolving spiking neural networks (eSNN)

2.1. ECOS

In general, eSNN use the principles of evolving connectionist systems (ECOS), where neurons are created (evolved) incrementally to capture clusters of input data either in an unsupervised way, e.g.: DENFIS (Kasabov & Song, 2002), or in a supervised way, e.g. EFuNN (Kasabov, 2001). All developed models of ECOS type, from simple ECOS (Kasabov, 2002; comprehensive review in Watts (2009)), to eSNN (Kasabov 2007, comprehensive review in Schliebs and Kasabov (2012)), and then—to the introduced in this paper dynamic eSNN, have been guided by the following 7 main principles (Kasabov, 2002):

(1) They evolve in an open space.
(2) They learn in on-line, incremental mode, possibly through one pass of incoming data propagation through the system.
(3) They learn in a ‘life-long’ learning mode.
(4) They learn both as individual systems and as an evolutionary population of systems.
(5) They use constructive learning and have evolving structures.
(6) They learn and partition the problem space locally, thus allowing for a fast adaptation and tracing the evolving processes over time.
(7) They facilitate different types of knowledge, mostly a combination of memory-based, statistical and symbolic knowledge.

2.2. Evolving spiking neural networks (eSNN)

The eSNN paradigm extends the early ECOS models with the use of integrate and fire (IF) model of a neuron (Kistler & Gerstner, 2002) and RO learning. This is schematically shown in Fig. 1.

The RO learning motivation is based on the assumption that most important information of an input pattern is contained in earlier arriving spikes (Thorpe and Gautrais 1989). It establishes a priority of inputs based on the order of the spike arrival on the input synapses for a particular pattern. This is a phenomenon observed in biological systems as well as an important information processing concept for some spatio-temporal problems, such as computer vision and control. RO learning makes use of the information contained in the order of the input spikes (events). This method has two main advantages when used in SNN: (1) fast learning (as the order of the first incoming spikes is often sufficient information for recognising a pattern and for a fast decision making and only one pass propagation of the input pattern may be sufficient for the model to learn it); (2) asynchronous, data-driven processing. As a consequence, RO learning is most appropriate for AER input data streams as the address-events are

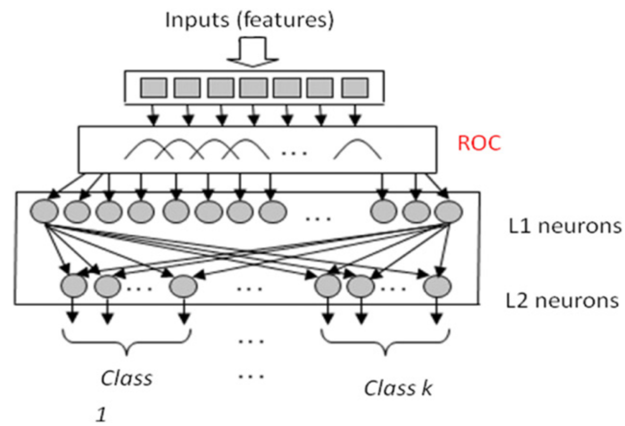

Fig. 2. Example of an eSNN for classification using population RO coding of inputs (Soltic & Kasabov, 2010). Each input is connected to several feature neurons representing overlapping Gaussian receptive fields and producing spikes according to how much the current input variable value belongs to the receptive field: the higher the membership degree—the earlier a spike is generated and forwarded to the output neurons for learning or recall. A pool of output neurons representing different input vectors or prototypes is evolved for each class. This model was used for odour recognition. In unsupervised learning, the output neurons are not labelled and not organised as class pools.

conveyed into the SNN 'one by one', in the order of their happening (Delbruck, 2007; Lichtsteiner & Delbruck, 2005).

Thorpe (2009) and Thorpe and Gautrais (1998) utilised RO learning to achieve fast, one-pass learning of static patterns (images). This idea was used in a class of SNN, called evolving SNN (eSNN) (Kasabov, 2007; Wysoski et al., 2010). An eSNN evolves its structure and functionality in an on-line manner, from incoming information. For every new input data vector, a new output neuron is dynamically allocated and connected to the input neurons (feature neurons). The neuron's connections are established using the RO rule for the output neuron to recognise this vector (frame, static pattern) or a similar one as a positive example. The weight vectors of the output neurons represent centres of clusters in the problem space and can be represented as fuzzy rules (Soltic & Kasabov, 2010).

In some implementations neurons with similar weight vectors are merged based on Euclidean distance between them. That makes it possible to achieve a very fast learning (only one pass may be sufficient), both in a supervised and in an unsupervised mode (Kasabov, 2007). When in an unsupervised mode, the evolved neurons represent a learned pattern (or a prototype of patterns). The neurons can be labelled and grouped according to their belonging to the same class if the model performs a classification task in a supervised mode of learning — an example is shown in Fig. 2.

During a *learning phase*, for each M-dimensional training input pattern (sample, example, vector) P_i a new output neuron i is created and its connection weights $w_{j,i}$ $(j = 1, 2, \dots, M)$ to the input (feature) neurons are calculated based on the order of the incoming spikes on the corresponding synapses using the RO learning rule:

$$w_{j,i} = \alpha \cdot mod^{order(j,i)} \tag{1}$$

where: α is a learning parameter (in a partial case it is equal to 1); *mod* is a modulation factor, that defines how important the order of the first spike is; $w_{j,i}$ is the synaptic weight between a pre-synaptic neuron j and the postsynaptic neuron i; $order(j, i)$ represents the order (the rank) of the *first* spike at synapse j, i ranked among all spikes arriving from all synapses to the neuron i; $order(j, i)$ has a value 0 for the first spike to neuron i and increases according to the input spike order at other synapses.

While the input training pattern (example) is presented (all input spikes on different synapses, encoding the input vector are presented within a time window of T time units), the spiking threshold Th_i of the neuron i is defined to make this neuron spike when this or a similar pattern (example) is presented again in the recall mode. The threshold is calculated as a fraction (C) of the total PSP_i (denoted as $\mathrm{PSP}_{i\max}$) accumulated during the presentation of the input pattern:

$$\mathrm{PSP}_{i\max} = \sum_{(j)} mod^{order(j,i)} \tag{2}$$

$$Th_i = C \cdot \mathrm{PSP}_{i\max}. \tag{3}$$

If the weight vector of the evolved and trained new neuron is similar to the one of an already trained neuron (in a supervised learning mode for classification this is a neuron from the same class pool), i.e. their similarity is above a certain threshold *Sim*, the new neuron will be merged with the most similar one, averaging the connection weights and the threshold of the two neurons (Kasabov, 2007; Wysoski et al., 2010). Otherwise, the new neuron will be added to the set of output neurons (or the corresponding class pool of neurons when a supervised learning for classification is performed). The similarity between the newly created neuron and a training neuron is computed as the inverse of the Euclidean distance between weight matrices of the two neurons. The merged neuron has weighted average weights and thresholds of the merging neurons.

While an individual output neuron represents a single input pattern, merged neurons represent clusters of patterns or prototypes in a transformed spatial—RO space. These clusters can be represented as fuzzy rules (Soltic & Kasabov, 2010) that can be used to discover new knowledge about the problem under consideration.

The eSNN learning is adaptive, incremental, theoretically—'lifelong', so that the system can learn new patterns through creating new output neurons, connecting them to the input neurons, and possibly merging the most similar ones. The eSNN implement the 7 ECOS principles from Section 1.

During the *recall phase*, when a new input vector is presented and encoded as input spikes, the spiking pattern is submitted to all created neurons during the learning phase. An output spike is generated by neuron i at a time l if the $\mathrm{PSP}_{i,}(l)$ becomes higher than its threshold Th_i. After the first neuron spikes, the PSP of all neurons are set to initial value (e.g. 0) to prepare the system for the next pattern for recall or learning.

The postsynaptic potential $\mathrm{PSP}_i(l)$ of a neuron i at time l is calculated as:

$$\mathrm{PSP}_i(l) = \sum_{t=0,1,2,\dots,l} \sum_{(j)} e_j(t) \cdot mod^{order(j,i)} \tag{4}$$

where: $e_j(t) = 1$ if there is a *first* spike at time t on synapse j; $order(j, i)$ is the rank order of the first spike at synapse j among all spikes to neuron i for this recall pattern.

The parameter C, used to calculate the threshold of a neuron i, makes it possible for the neuron i to emit an output spike before the presentation of the whole learned pattern (lasting T time units) as the neuron was initially trained to respond. As a partial case $C = 1$.

The recall procedure can be performed using different recall algorithms implying different methods of comparing input patterns for recall with already learned patterns in the output neurons:

(a) The first one is described above. Spikes of the new input pattern are propagated as they arrive to all trained output neurons and the first one that spikes (its PSP is greater that its threshold) defines the output. The assumption is that the neuron that best matches the input pattern will spike earlier based purely on the PSP (membrane potential). This type of eSNN is denoted as eSNNm.

(b) The second one implies a creation of a new output neuron for each recall pattern, in the same way as the output neurons were created during the learning phase, and then—comparing the connection weight vector of the new one to the already existing neurons using Euclidean distance. The closest output neuron in terms of synaptic connection weights is the 'winner'. This method uses the principle of transductive reasoning and nearest neighbour classification in the connection weight space. It compares spatially distributed synaptic weight vectors of a new neuron that captures a new input pattern and existing ones. We will denote this model as eSNNs.

The main advantage of the eSNN when compared with other supervised or unsupervised SNN models is that it is computationally inexpensive and boosts the importance of the order in which input spikes arrive, thus making the eSNN suitable for on-line learning with a range of applications. For a comprehensive study of eSNN see Wysoski et al. (2010) and for a comprehensive review — (Schliebs & Kasabov, 2012).

The problem of the eSNN is that once a synaptic weight is calculated based on the first spike using the RO rule, it is fixed and does not change to reflect on other incoming spikes at the same synapse, i.e. there is no mechanism to deal with multiple spikes arriving at different times on the same synapse. The synapses are static. While the synapses capture some (long term) memory during the learning phase, they have limited abilities (only through the PSP growth) to capture short term memory during a whole spatio-temporal pattern presentation. Learning and recall of complex spatio-temporal patterns in an on-line mode would need not only fast initial set of connection weights, based on the first spikes, but also dynamic changes of these synapses during the pattern presentation.

Section 4 proposes an extended eSNN model, called deSNN, that utilises the SDSP learning rule (Fusi et al., 2000) to implement dynamic changes of the synaptic weights, after they are initialised with the RO rule, in both learning and recall phases. Sections 5 and 6 demonstrate that the proposed deSNN performs better than either the eSNN or the SDSP alone for two different classes of spatio-temporal problems: moving object recognition based on AER, and EEG recognition based on temporal EEG frames. This is due to the combination of the fast RO learning and the dynamic synapses realised through the SDSP. The deSNN model is suitable for neuromorphic implementation (Section 7) and would make possible new engineering applications of the fast developing SNN technologies (Section 8).

3. Spike-time learning methods

Spike-time learning is observed in auditory and visual information processing in the brain as well as in the motor control (Bohte, 2004; Morrison, Diesmann, & Gerstner, 2008). Its use in neuro-prosthetics is essential along with applications for a fast, real-time recognition and control of sequence of related processes (Bichler, Ouerlioz, Thorpe, Bourgoin, & Gamrat, 2011). Temporal coding accounts for the precise time of spikes and has been utilised in several learning rules, most popular being Spike-Time Dependent Plasticity (STDP) (Song, Miller, & Abbott, 2000) and its variant—SDSP (Brader, Senn, & Fusi, 2007; Fusi et al., 2000). SDSP was also implemented in a SNN hardware chip (Indiveri et al., 2009).

3.1. The spike time dependent plasticity (STDP) learning rule

The STDP learning rule represents a Hebbian form of plasticity (Hebb, 1949) in the form of long-term potentiation (LTP) and depression (LTD) (Song et al., 2000). Efficacy of synapses is strengthened or weakened based on the timing of post-synaptic action potentials in relation to the pre-synaptic spike (example is given in Fig. 3). If the difference in the spike time between the pre-synaptic and post-synaptic neurons is negative (pre-synaptic neuron spikes first) than the connection weight between the two neurons increases, otherwise—it decreases. Through STDP connected neurons learn consecutive temporal associations from data. Pre-synaptic activity that precedes post-synaptic firing can induce long-term potentiation (LTP), reversing this temporal order causes long-term depression (LTD).

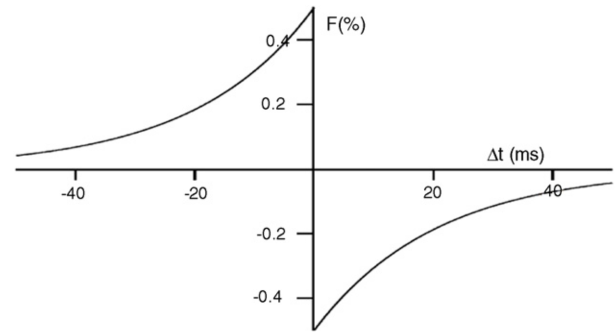

Fig. 3. An illustration of the STDP learning rule (Song et al., 2000). The change in efficacy of a synaptic weight (F) depends of the time difference between the pre-synaptic and post-synaptic spikes.

3.2. The spike driven synaptic plasticity (SDSP) learning rule

The SDSP is a semi-supervised learning method (Fusi et al., 2000), a variant of the STDP rule, that directs the change of the synaptic plasticity V_{w0} of a synapse w_0 depending on the time of spiking of the pre-synaptic neuron and the post-synaptic neuron. V_{w0} increases or decreases, depending on the relative timing of the pre- and post-synaptic spikes.

If a pre-synaptic spike arrives at the synaptic terminal while the post-synaptic neuron's membrane potential is above a given threshold Vmth (i.e. typically shortly before a postsynaptic spike is emitted), the synaptic efficacy is increased (potentiation). If the post-synaptic neuron's membrane potential is low (i.e. typically shortly after a spike is emitted) when the pre-synaptic spike arrives, synaptic efficacy is decreased (depression). This change in synaptic efficacy can be expressed as:

$$\Delta V_{w0} = \frac{I_{\text{pot}}(t_{\text{post}})}{C_p}, \quad \text{if Vmem}_t > \text{Vmth} \tag{5}$$

$$\Delta V_{w0} = -\frac{I_{\text{dep}}(t_{\text{post}})}{C_d} \quad \text{if Vmem}_t < \text{Vmth}. \tag{6}$$

The SDSP learning rule introduces a long term dynamic 'drift' of the synaptic weights either 'up' or 'down', depending on the value of the weight itself. If the weight is above a given threshold Vwth then the weight is slowly driven to a fixed high value. Conversely, if the weight is driven by the learning mechanism to a low value, below Vwth, then the weight is slowly driven to a fixed low value. These two values represent the two stable states of this bistable learning method. As the final weights, at the end of learning, can be encoded with 1 single bit, this learning rule lends itself to a very efficient implementation in hardware, both with analogue VLSI circuits, as well as FPGA implementations (Mitra, Fusi, & Indiveri, 2009).

The SDSP rule can be used also as a supervised learning algorithm, when a 'teacher signal', that drives the post-synaptic neuron's membrane potential high or low is applied along with the training spike pattern.

In Brader et al. (2007) the SDSP model has been successfully used to train and test a SNN for 293 character recognition (classes). Each character (a static image) is represented as 2000 bit feature vector, and each bit is transferred into spike rates, with 50 Hz spike burst to represent 1 and 0 Hz to represent 0. For each class, 20 different training patterns are used and 20 neurons are allocated, one for each pattern (altogether 5860) and trained for several thousand iterations. Rate coding of information was used rather than temporal coding, which is typical for unsupervised learning in SNN.

While successfully used for the recognition of mainly static patterns, the potential of the SDSP SNN model and its hardware realisation have not been fully explored for SSTD, definitely not for fast on-line learning of complex spatio-temporal patterns.

Masquelier, Guyonneau and Thorpe (T. Masquelier, R. Guyonneau and S. Thorpe, PlosONE, Jan2008) demonstrated that a single LIF neuron with simple synapses can be trained with the STDP unsupervised learning rule to discriminate a repeating pattern of synchronised spikes on certain synapses from noise. The training requested hundreds iterations and the more the training was repeated, the earlier the beginning of the synchronised spiking pattern was detected from the input stream.

The introduced in the next section deSNN utilises a combination of RO learning and SDSP learning, so that one LIF neuron is trained to recognise whole spatio-temporal input pattern on many synaptic inputs using only one iteration of training, on-line mode. For this purpose, the neuron first 'utilises' through RO learning important the information of the order the incoming spikes (rather than learning this information in an unsupervised STDP mode using many iterations) and then the neuron tunes the initial connection weights through STDP learning over the rest of the spatio-temporal pattern. For every spatio-temporal input pattern a new, separate output neuron is evolved to learn this pattern. Output neurons may be merged based on closeness.

3.3. Dynamic synapses

Both STDP and SDSP provide means for implementing synaptic plasticity that have been already utilised in other methods. A phenomenological model for the short-term dynamics of synapses has been proposed more than a decade ago by Maass and Markram (2002) and by Tsodyks, Pawelzik, and Markram (1998). The model which is based on experimental data of biological synapses, suggests that the synaptic efficiency (weight) is a dynamic parameter that changes with every pre-synaptic spike due to two short-term synaptic plasticity processes: facilitation and depression. This inherent synaptic dynamics empower neural networks with a remarkable capability for carrying out computations on temporal patterns (i.e., time series) and spatio-temporal patterns. Maass and Sontag (2000) in their theoretical analysis considering analogue input showed that with just a single hidden layer such networks can approximate a very rich class of nonlinear filters. However there is a need for similar study in the presence of many inputs that carry sequences of spikes in a temporal relationship. It is suggested also that dynamic synapses work as memory buffers (Maass, Natschlaeger & Markram, 2002) due to the fact that a current spike is influenced by previous spikes. Furthermore a SNN with dynamic synapses is showed to be able to induce a Finite State Machine mechanism (Natschläger & Maass, 2002). A number of studies have utilised dynamic synapses in practical applications. One of the first practical application of dynamic synapses was speech recognition (Namarvar, Liaw, & Berger, 2001) and later, image filtering (Mehrtash, Jung, & Klar, 2003).

The proposed in the next section deSNN model extends the eSNN with the introduction of dynamic synapses for the purpose of complex SSTD pattern recognition.

4. Dynamic evolving SNN (deSNN)

The main disadvantage of the RO learning in eSNN is that the model adjusts the connection weight of each synapse once only (based on the rank of the first spike on this synapse), which may be appropriate for static pattern recognition, but would not be efficient for complex SSTD. In the latter case the connection weights need to be further tuned based on the following spikes arriving on the same synapse over time and that is where the spike-time learning (e.g. STDP or SDSP) can be employed in order to implement dynamic synapses.

In the proposed deSNN both the RO and the SDSP learning rules are utilised. While the RO learning will set the initial values of the connection weights $\mathbf{w}(\mathbf{0})$ (utilising for example the existing event order information in an AER data), the SDSP rule will adjust these connection weights based on further incoming spikes (events) as part of the same learned spatio-temporal pattern. Adjusting synaptic weights due to every pre-synaptic spike was implemented in the dynamic synapse model of Maass and Markram (2002).

As in the eSNN, during a *learning phase*, for each training input pattern (sample, example, vector) P_i a new output neuron i is created and its connection weights $w_{j,i}$ to the input (feature) neurons are initially calculated as $w_{j,i}(0)$ based on the order of the incoming spikes on the corresponding synapses using the RO learning rule—formula (1).

Once a synaptic weight $w_{j,i}$ is initialised, based on the first spike at the synapse j, the synapse becomes dynamic and adjusts its weight through the SDSP algorithm. It increases its value with a small positive value (positive drift parameter) at any time t a new spike arrives at this synapse and decreases its value (a negative drift parameter) if there is no spike at this time.

$$\Delta w_{j,i}(t) = e_j(t) \cdot D \tag{7}$$

where: $e_j(t) = 1$ if there is a consecutive spike at synapse j at time t during the presentation of the learned pattern by the output neuron i and (-1) otherwise. In general, the drift parameter D can be different for 'up' and 'down' drifts.

All dynamic synapses change their values in parallel for every time unit t during a presentation of an input spatio-temporal pattern P_i learned by an output neuron i, some of them going up and some—going down, so that all synapses (not a single one) of the neuron could collectively capture some temporal relationship of spike timing across the learned pattern.

While an input training pattern (example) is presented (all input spikes on different synapses, encoding the input vector are presented within a time window of T time units), the spiking threshold Th_i of the neuron i is defined to make this neuron spike when this or a similar pattern (example) is presented in the recall mode. The threshold is calculated as a fraction (C) of the total PSP_i (denoted as PSP_{imax}) accumulated during the presentation of the whole input pattern:

$$\mathrm{PSP}_{imax} = \sum_{t=1,2,\dots,T} \sum_{j=1,2,\dots,M} f_j(t) \cdot w_{j,i}(t) \tag{8}$$

$$Th_i = C \cdot \mathrm{PSP}_{imax} \tag{9}$$

where: T represents the time units in which the input pattern is presented; M is the number of the input synapses to neuron i; $f_j(t) = 1$ if there is spike at time t at synapse j for this learned input pattern, otherwise it is 0; $w_{j,i}(t)$ is the efficacy of the (dynamic) synapse between j and i neurons calculated at time t with the use of formula (7).

The resulted deSNN model after training will contain the following information:

Table 1
The **deSNN training algorithm**.

1: Set deSNN parameters[a] (including: Mod, C, Sim, and the SDSP parameters)
2: **For** every input spatio-temporal spiking pattern P_i **Do**
2a. Create a new output neuron i for this pattern and calculate the initial values of connection weights $\mathbf{w_i}(0)$ using the RO learning formula (1).
2b. Adjust the connection weights $\mathbf{w_i}$ for consecutive spikes on the corresponding synapses using the SDSP learning rule (formula (7)).
2c. Calculate $\mathrm{PSP}_{i\max}$ using formula (8).
2d. Calculate the spiking threshold of the ith neuron using formula (9).
2e. **(Optional) If** the new neuron weight vector $\mathbf{w_i}$ is similar in its initial $\mathbf{w_i}(0)$ an final $\mathbf{w_i}(T)$ values after training to the weight vector of an already trained output neuron using Euclidean distance and a similarity threshold *Sim*, then merge the two neurons (as a partial case only initial or final values of the connection weights can be considered or a weighted sum of them)
Else
Add the new neuron to the output neurons repository.
End If
End For (Repeat for all input spatio-temporal patterns for learning)

[a] The performance of the deSNN depends on the optimal selection of its parameters as illustrated in the examples below.

– Number of input neurons M and output neurons N;
– Initial $\mathbf{w_i}(0)$ and final $\mathbf{w_i}(T)$ vectors of connection weights and spiking threshold Th_i for each of the output neurons i. The pairs $[\mathbf{w_i}(0), \mathbf{w_i}(T)]$, $i = 1, 2, \ldots, N$ would capture collectively dynamics of the learning process for each spatio-temporal pattern and each output neuron (as a partial case only initial or final values of the connection weights can be considered or a weighted sum of them).
– deSNN parameters.

The overall deSNN training algorithm is presented in Table 1.

Example 1. Fig. 4 illustrates the main idea of the deSNN learning algorithm. A single spatio-temporal pattern of four input spike trains is learned into a single output neuron. RO learning is applied to calculate the initial weights based on the order of the first spike on each synapse (shown in red).

Then STDP (in this case—SDSP) rule is applied to dynamically tune these connection weights. The SDSP algorithm increases the assigned connection weight of a synapse which is receiving a following spike and at the same time depresses the synaptic connections of synapses that do not receive a spike at this time. Due to a bi-stability drift in the SDSP rule, once a weight reaches the defined High value (resulting in LTP) or Low value (resulting in LTD), this connection weight is fixed to this value for the rest of the training phase. The rate at which a weight reaches LTD or LTP depends upon the set parameter values.

For example, if input spikes arrive at times (0, 1, 2) ms on the first synapse, and are shifted by 1 ms for the other 3 synapses as shown in Fig. 4, the four initial connection weights w_1, w_2, w_3, w_4 to the output neuron will be calculated as: 1, 0.8, 0.64, 0.512 correspondingly, when the parameter $mod = 0.8$. If the SDSP High value is 0.6 and Low value is 0, the first three weights will be fixed to 0.6 and the fourth one will drift up 2 times. If the drift parameter is set to 0.00025, the final weight value of the fourth synapse will be 0.5125. After training both the initial and the final weights can be memorised.

Example 2. In this example we consider 2 spatio-temporal patterns of 5 inputs each (Table 2) to be learned in two output neurons. The initial (after RO learning) and final (after SDSP learning) connection weights are shown in Table 2 and also in Fig. 5.

A code written in Python for learning the above 2 patterns in 2 output neurons, each having 5 inputs, is given in Appendix A. This code can be modified for many other deSNN simulations.

The synaptic drift caused by the SDSP makes synaptic weights dynamically learn spike time relationships between different input spike trains as part of the same spatio-temporal pattern. The hypothetical examples above are of a very small scale and highly simplified scenarios. In reality there are hundreds and thousands of input synapses to a neuron and hundreds and thousands of spikes at each synapse forming a complex spatio-temporal pattern to be learned, described by some statistical characteristics. Even a small synaptic drift can make a difference. This is illustrated in the two case studies in Sections 5 and 6.

The connection weights learned in a deSNN represent the input patterns in an internal, computational spatio-temporal space built by the model. How many different input patterns can be learned and discriminated in this space depends on the choice of the model parameters. This issue will be discussed in Section 8.

To summarise the learning of the deSNN, for every spatio-temporal input pattern, a new, separate output neuron is evolved to learn this pattern even the patterns may of the same class (for classification tasks) or very similar (for unsupervised learning). Output neurons may be merged based on closeness. Here we do not use winner-takes-all connections between output neurons (as it is the case in Masquelier et al. (2009) or in Brader et al. (2007)). Here it is a matter of a proper selection of the parameters for both RO and STDP learning that makes it possible for the deSNN to learn a whole spatio-temporal pattern.

So far, we have presented the learning phase of a deSNN model. In terms of *recall*, two types of deSNN are proposed that differ in the recall algorithms. They mainly correspond to the two types of eSNN from Section 2-eSNNs and eSNNm:

(a) deSNNm: After learning, only the initially created connection weights (with the use of the RO rule) are restored as long term memory in the synapses and the model. During recall on a new spatio-temporal pattern the SDSP rule is applied so that the initial synaptic weights are modified on a spike time basis according to the new pattern using formula (7) as it is during the SDSP learning phase. At every time moment t the PSP of all output neurons are calculated. The new input pattern is associated with the neuron i if the $\mathrm{PSP}_i(t)$ is above its threshold Th_i. The following formula is used:

$$\mathrm{PSP}_i(t) = \sum_{l=0,1,2,\ldots,t} \sum_{j=1,2,\ldots,M} f_j(l) \cdot w_{j,i}(l) \tag{10}$$

where: t represents the current time unit during the presentation of the input pattern for recall; M is the number of the input synapses to neuron i; $f_j(l) = 1$ if there is spike at time l at synapse j for this input pattern, otherwise it is 0; $w_{j,i}(l)$ is the efficacy of the dynamic synapse between j and i neurons at time l.

(b) deSNNs: This model corresponds to the eSNNs and is based on the comparison between the synaptic weights of a newly created neuron to represent the new spatio-temporal pattern for recall, and the connection weights of the created during training neurons. The new input pattern is associated with the closest output neuron based on the minimum distance between the weight vectors. As the synaptic weights are dynamic, the distance should be calculated in a different way

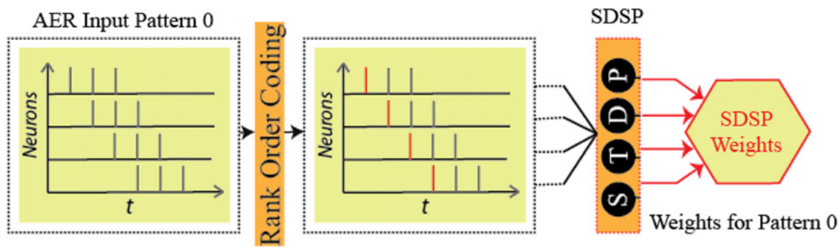

Fig. 4. A simple example to illustrate the main principle of the deSNN learning algorithm. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 2
Initial connection weights (after the ROC learning) and final (after the further SDSP learning) of each of two neurons trained on Pattern 1 and Pattern 2 correspondingly. Each pattern consists of 5 inputs and several spikes on each input, generated at different consecutive times (in ms).

Pattern 1			Pattern 2		
Spike times (ms)	ROC	SDSP	Spike times (ms)	ROC	SDSP
Input 1: 0.0, 1.0, 2.0, 3.0, 4.0	1.0000	0.9980	Input 1: 4.0, 5.0, 6.0, 7.0, 8.0	0.4096	0.0000
Input 2: 1.0, 2.0, 3.0, 4.0, 5.0	0.8000	0.7980	Input 2: 3.0, 4.0, 5.0, 6.0, 7.0	0.5120	0.0000
Input 3: 2.0, 3.0, 4.0, 5.0, 6.0	0.6400	0.0000	Input 3: 2.0, 3.0, 4.0, 5.0, 6.0	0.6400	0.0000
Input 4: 3.0, 4.0, 5.0, 6.0, 7.0	0.5120	0.0000	Input 4: 1.0, 2.0, 3.0, 4.0, 5.0	0.8000	0.7980
Input 5: 4.0, 5.0, 6.0, 7.0, 8.0	0.4096	0.0000	Input 5: 0.0, 1.0, 2.0, 3.0, 4.0	1.0000	0.9980
Time of a pattern presentation $T = 8.0$ ms					

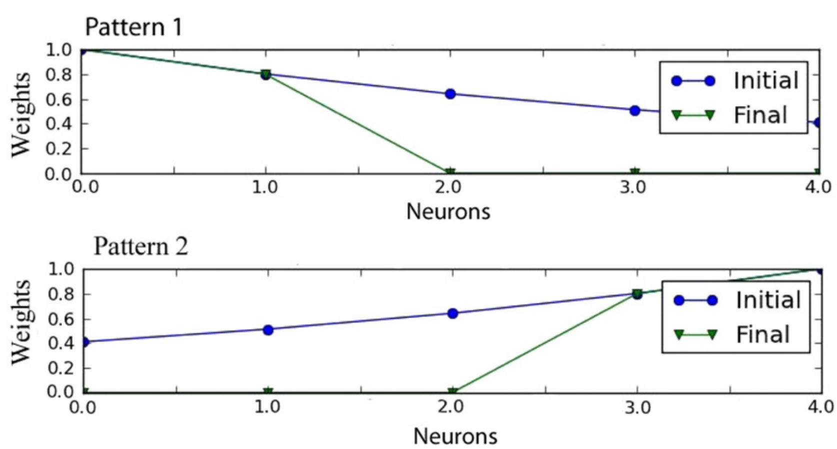

Fig. 5. Initial and final synaptic weights of the two output neurons for Example 2.

than the distance measured in the eSNN possibly using both the initial $\mathbf{w}(0)$ and the final $\mathbf{w}(T)$ connection weigh vectors learned during training and recall. As a partial case, only the final weight vector $\mathbf{w}(T)$ can be used.

5. deSNN for moving object recognition with AER

Many of the real-time machine vision systems have an inherent limitation of processing information on a frame by frame basis, mainly due to the redundant information present within and across the frames. However, this drawback can be overcome with the use of AER as illustrated in Fig. 6. In AER an event is generated based on the corresponding changes in the log intensity of the signal. This is the case in the artificial silicon retina sensory device (Lichtsteiner & Delbruck, 2005). It mimics aspects of our biological vision system which utilises asynchronous spike events captured by the retina. This allows for fast and efficient processing since it discards irrelevant redundant information by capturing only information corresponding to the temporal changes in log intensity.

Here we have used AER SSTD of a moving object collected through the DVS silicon retina device. The object is a moving irregular wooden bar in front of the camera (Dhoble, Nuntalid, Indivery, & Kasabov, 2012). Two classes of movements are recorded as: 'crash' and 'no crash'. For the 'crash' samples, the object is recorded as it approaches the camera. For 'no crash' movements, motions such as 'up/down' are recorded at a fixed distance from the camera. The size of the recorded area is 7000 pixels. Each movement is recorded 10 times, 5 used for training and 5 for testing. Five models are created, trained and tested, using different learning rules: SDSP; eSNNs; eSNNm; deSNNs and deSNNm. There is no merge of neurons in the eSNN and deSNN models. 5 output neurons are evolved for each of the 2 classes, each neuron trained on a single training example. The parameter C for the eSNN and the deSNN models has been optimised between 0 and 1 (with a step of 0.1). All parameters and their values used in the models are presented in Table 3. The classification results along with the number of training iterations are shown in Table 4. The best classification result, in terms of number of true positive plus true negative examples divided to the total number of examples, is 0.9 obtained for a threshold using parameter $C = 0.55$.

The results show that when using deSNNm on AER data a higher accuracy of classification is achieved when compared with the other models. This is because in addition to the useful information contained in the order of the incoming spikes across all synapses,

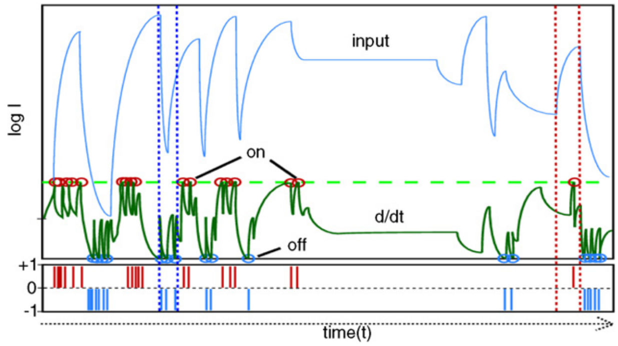

Fig. 6. The figure shows an idealised AER pixel encoding of video data. The ON and OFF events represent significant changes in log I (intensity of the signal). A positive change greater than a threshold generates an excitatory spike event, while a negative change generates an inhibitory spike event; no change—no spike. *Source:* Adapted from Lichtsteiner and Delbruck (2005).

Table 3
deSNN parameter settings for the moving object recognition experiment.

Neurons and synapses	
Excitatory synapse time constant	2 ms
Inhibitory synapse time constant	5 ms
Neuron time constant (tau mem)	20 ms
Membrane leak	20 mV
Spike threshold (Vthr)	800 mV
Reset value	0 mV
Fixed inhibitory weight	0.20 V
Fixed excitatory weight	0.40 V
Thermal voltage	25 mV
Refractory period	4 ms
Learning related parameters (SDSP)	
Up/Down weight jumps (Vthm)	5 × (Vthr/8)
Calcium variable time constant (tau ca)	5 × (tau mem)
Steady-state asymptote for Calcium variable (wca)	50 mV
Stop-learning threshold 1 (stop if Vca < thk1)	1.7 × wca
Stop-learning threshold 2 (stop LTD if Vca > thk2)	2.2 × wca
Stop-learning threshold 2 (stop LTP if Vca > thk3)	8 × (wca–wca)
Plastic synapse (NMDA) time constant	9 ms
Plastic synapse high value (wp hi)	6 mV
Plastic synapse low value (wp lo)	0 mV
Bistability drift	0.25
Delta weight	0.12 × wp hi
Other parameters/values	
Input size	7000 spike train
Simulation time	1600 ms
mod (for rank order)	0.8

Table 4
Classification accuracy on the case study moving object recognition task when different SNN classifiers are used on the same training and test data sets.

	SDSP SNN	eSNNs	eSNNm	deSNNs	deSNNm
Classification accuracy on the test samples	70%	40%	60%	60%	90%
Number of training iterations	5	1	1	1	1

what also matters is the intensity of the following incoming spikes at every synapse for this particular pattern. The higher the intensity, the higher the chances of a synapse to further increase its efficacy which is obtained through the use of the SDSP learning rule and properly selected parameters. For a single application of the SDSP rule (first column in Table 4) the accuracy did not increase further with the increase of the number of the training iterations.

An illustration of the learning process of an input pattern 'crash' over 1600 ms is shown in Fig. 7a. This figure shows the spike raster plot of a single AER of a 'crash' pattern (top figure; the dots represent spikes of 7000 input neurons representing spatially distribute pixels over 1600 ms), and also the changes of the weights (middle figure) and the membrane potential (low figure) for output neuron 0 during the one-pass learning in a deSNNs.

This experiment demonstrates the feasibility of using deSNNm for moving object recognition on a 'crash/no crash' example, which can lead to important applications of avoiding collisions between fast moving objects (e.g. cars, rockets, space objects). The classification performance of a trained deSNNm is significantly different from a random classification as it is illustrated with the ROC curve (Fig. 7b).

6. deSNN for EEG pattern recognition with frame-based representation and BSA spike transformation algorithm

In contrast to the previous case study where AER was used to encode input SSTD, here recordings (frames) of EEG signals over time are used for learning and recognition using the RIKEN EEG dataset (see (Kasabov, 2007)). The dataset was collected in the RIKEN Brain Science Institute in Japan. It includes 3 meaningful stimulus conditions (classes): Class1 — EEG data recorded from a subject when auditory stimulus was presented; Class2 — Visual stimulus is presented; Class3 — Mixed auditory and visual stimuli are presented. A 'No stimulus' EEG data was also collected, but it is ignored for this experiment. The EEG data was acquired using a 64 electrode EEG system that was filtered using a 0.05–500 Hz band-pass filter and sampled at 2 kHz. An EEG SSTD sample has a length of 50 ms in actual time. 11 samples of each class were selected, 80% of them used for training and 20% for testing. The simulation time was extended 10 times (to 500 ms) for this experiment where the amount of EEG recordings within an input pattern is the same—2000.

Here we have used BSA spike encoding scheme (Nuntalid, Dhoble, & Kasabov, 2011; Schrauwen & Van Campenhout, 2003) to represent an EEG vector (frame) into spikes. This encoding scheme has already been used for encoding spectro-temporal data (sound). EEG signals are both spatio-temporal and spectro-temporal and that is the reason we have chosen the BSA encoding. The key benefit of using BSA is that the frequency and amplitude features are smoother in comparison to the HSA (Hough Spike Algorithm) encoding scheme (Schrauwen & Van Campenhout, 2003). Moreover, due to the smoother threshold optimisation curve, the representation is also less susceptible to changes in the filter and the threshold. Studies have shown that this method offers an improvement of 10–15 dB over the HSA spike encoding scheme.

The parameters of the deSNN and the eSNN models trained on the EEG data are shown in Table 5 and the classification results—in Table 6. The best accuracy is obtained with the use of the deSNNs model. All models in this experiment were run only for 1 iteration of training (one-pass) in order to make a fair comparison between them.

The results when using only RO learning (eSNN) or only SDSP (SDSP SNN) are not as good as the results when deSNNs was used. This is for the following reasons:

(a) Using only RO learning is not sufficient when frame based input data is transformed into spikes through the BSA algorithm;
(b) Using only SDSP learning ignores the importance of the first/initial spikes from each spike trains, but these spikes carry important information for brain activity;
(c) deSNNm could in principle produce good results, but it requires a fine tuning of the parameters including a proper choice of the C parameter for the calculation of the neuronal spiking thresholds, which in this case was difficult to find as there was no automated optimisation procedure applied;
(d) deSNNs performed better that deSNNm due to the high density of spikes in the EEG spatio/spectro temporal patterns and the fact that deSNNs is more robust to choosing the simulation parameter values than the deSNNm.

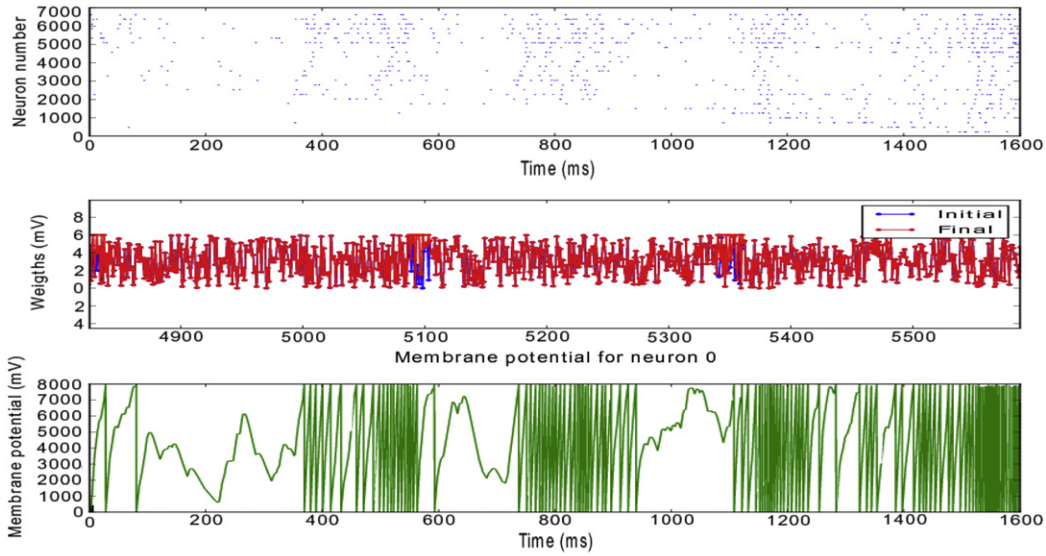

Fig. 7a. This figure shows the spike raster plot of a single AER of an input pattern denoted as 'crash' (top figure; the dots represent spikes of 7000 input neurons representing spatially distribute pixels over 1600 ms), and also the changes of some of the weights (middle figure) and the membrane potential (low figure) for output neuron 0 during the one-pass learning in a deSNN.

Table 5
Parameter setting for the case study on EEG spatio/spectro temporal pattern recognition.

Neurons and synapses	
Excitatory synapse time constant	2 ms
Inhibitory synapse time constant	5 ms
Neuron time constant (tau mem)	20 ms
Membrane leak	20 mV
Spike threshold (Vthr)	800 mV
Reset value	0 mV
Fixed inhibitory weight	0.20 V
Fixed excitatory weight	0.40 V
Thermal voltage	25 mV
Refractory period	4 ms
Learning related parameters (SDSP)	
Up/Down weight jumps (Vthm)	5 × (Vthr/8)
Calcium variable time constant (tau ca)	5 × (tau mem)
Steady-state asymptote for Calcium variable (wca)	50 mV
Stop-learning threshold 1 (stop if Vca $<$ thk1)	1.7 ×wca
Stop-learning threshold 2 (stop LTD if Vca $>$ thk2)	2.2 × wca
Stop-learning threshold 2 (stop LTP if Vca $>$ thk3)	8 × (wca–wca)
Plastic synapse (NMDA) time constant	9 ms
Plastic synapse high value (wp hi)	6 mV
Plastic synapse low value (wp lo)	0 mV
Bistability drift	0.25
Delta weight	0.12 ×wp hi
Other parameters/values	
Input size (64 electrodes EEG)	64 spike train
Simulation time	500 ms
mod (for rank order)	0.8

Table 6
Classification results on the EEG case study of SSTD using different SNN models.

	SDSP SNN	eSNNs	eSNNm	deSNNs	deSNNm
Classification accuracy on the test data	66.67%	66.67%	50%	100%	83.33%
Number of training iterations	1	1	1	1	1

Figs. 8a and 8b shows part of the simulations of deSNN on EEG data. The top figure shows a raster plot of input spikes of one spatiotemporal EEG sample, the bottom gives the synaptic weights information of deSNN before SDSP learning (initiated, using RO—shown in blue) and after learning through SDSP (in green). It can be seen that the synaptic weights of an output neuron for a particular EEG pattern, change tremendously during the 500 ms time of learning from $\mathbf{w}(0)$ to $\mathbf{w}(T)$, e.g. channels 19, 26 and 62. This is obtained in the dynamic synapses of the deSNN, but cannot be learned in the eSNN static synapses. That confirms again the importance of both first spikes and their dynamic changes during the time of a whole EEG pattern presentation.

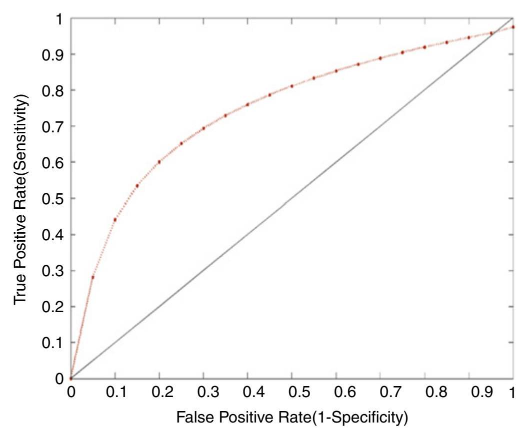

Fig. 7b. A ROC classification curve on the test data for EEG case study data using a frame based representation, BSA algorithm for spike transformation of EEG signals and deSNNs for learning and classification.

To generalise, learning and capturing changes of input signals in a set of dynamic synaptic weights is the key to the success of the deSNN models for some specific tasks. Fig. 8a also illustrates the feasibility of deSNN to handle high density of spikes in a short temporal window which is the nature of EEG data. The spike rate of this data is different from the spike rate of the Moving Object Recognition data with AER. Using deSNNs for EEG data classification produces significantly better results than a random classification as it is illustrated with the ROC curve in Fig. 8b.

As EEG data is widely used to measure brain SSTD for a wide range of applications, including medical applications and BCI (Ferreira, Almeida, Georgieva, Tomé, & Silva, 2010; Isa, Fetz, & Muller, 2009; Lalor et al., 2005; Neuper, Muller, Kubler, Birbaumer, & Pfurtscheller, 2003; Tanaka, Matsunaga, & Wang, 2005; Wolpaw, McFarland, & Vaughan, 2000), the use of deSNN could be further

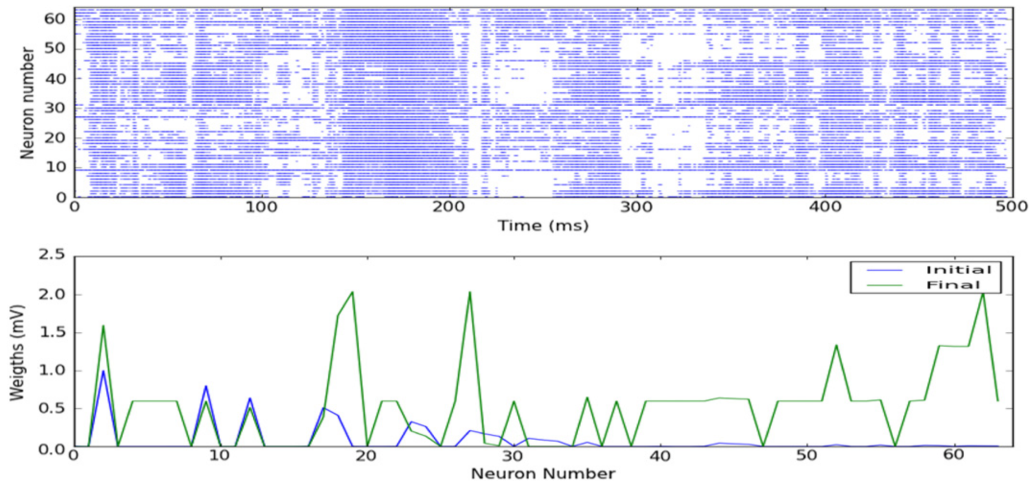

Fig. 8a. The figure shows an EEG SSTD spike raster plot (top figure; 64 input neurons on the *y*-axis over 50 ms real time on the *x*-axis—represented as 500 ms of simulation) and the weight changes of a single output neuron from a deSNN model during learning (lower figure). The initial weights are obtained through RO learning and the final weights—after the SDSP learning.

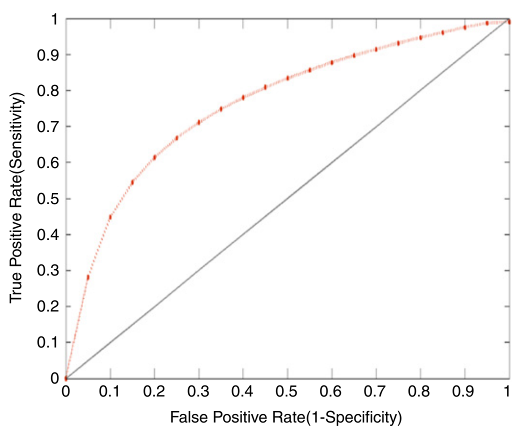

Fig. 8b. A ROC classification curve on test data of the EEG classification case study problem when using frame-based representation, BSA for a spike transformation of EEG signals and deSNNs for learning and classification.

studied in more concrete scenarios depending on the specification of each particular application.

7. Neuromorphic implementation of deSNN

Implementing deSNN on hardware SNN chips will enable the development of autonomous machine learning systems for a wide range of practical engineering applications (Indiveri & Horiuchi, 2011). The feasibility of implementing the deSNN model on some particular SNN chips is discussed here.

The SDSP learning rule applied on the LIF model of a neuron has already been successfully implemented in analogue VLSI technology (Mitra et al., 2009) making it possible for a deSNN model to be implemented on a chip. In this implementation, the silicon synapses comprise bi-stability circuits for driving a synaptic weight to one of two possible analogue values (either potentiated or depressed). These circuits drive the synaptic-weight voltage with a current that is superimposed on that generated by the on-line spike-driven weight update mechanism and which can be either positive or negative. If, on short time scales, the synaptic weight is increased above a set threshold by the network activity via the weight update learning mechanism, the bi-stability circuits generate a constant weak positive current. In the absence of activity (and hence learning) this current will drive the weight towards its potentiated state. If the weight update mechanism decreases the synaptic weight below the threshold, the bi-stability circuits will generate a negative current that, in the absence of spiking activity, will actively drive the weight towards the analogue value, encoding its depressed state. The chip allows for different types of dynamic synapses to be implemented, including the Tsodyks' model.

Another SNN chip that implements LIF model of a neuron is the recently proposed programmable SRAM SNN chip (Moradi & Indiveri, 2011). It is characterised by the following: 32 × 32 SRAM matrix of weights, each 5 bits (values between 0 and 31); 32 neurons of the adaptive, exponential IF model of a neuron; each neuron has 2 excitatory and 2 inhibitory inputs to which any of the 32 input dendrites (rows of weights) can be connected; AER for input data, for changing the connection weights and for output data streams; does not have any learning rule hardware implemented, so it allows to experiment with different supervised and unsupervised learning rules; learning (changing of the synaptic weights) is calculated outside the chip (in a computer, connected to the chip) in an asynchronous manner (only synaptic weights that need to change at the current time moment are changed (calculated) and then loaded into the SRAM) applying suitable learning rule and parameter settings.

The fact that modifying connection weights is done asynchronously outside the chip and then the weights are loaded in the SRAM allows for the deSNN learning algorithm to be implemented on this chip. After an input is applied to the AER circuits, the output from the neurons is produced and the deSNN learning algorithm implemented off-chip is then used to change connection weights accordingly. The new values of the weights are entered into the SRAM also asynchronously.

deSNN is also implementable on other recently proposed SNN chips of the same class, such as the digital IBM SNN chip (Arthur et al., 2012) as well as on FPGA systems (Mitra et al., 2009). Despite the fast, on-pass learning in the deSNN models, in terms of large scale modelling of millions and billions of neurons using the SpiNNaker SNN supercomputer system (Jin et al., 2010) for simulation purposes would be appropriate, especially at the level of parameter optimisation. Potentially, the deSNN can be used to implement real-time sensory processing neuromorphic architectures, which integrate audio-visual data, of the type shown in Fig. 9.

8. Discussions, conclusions and further directions

The paper presents a new dynamic eSNN model, deSNN, that combines rank-order (RO) and spike-time learning for

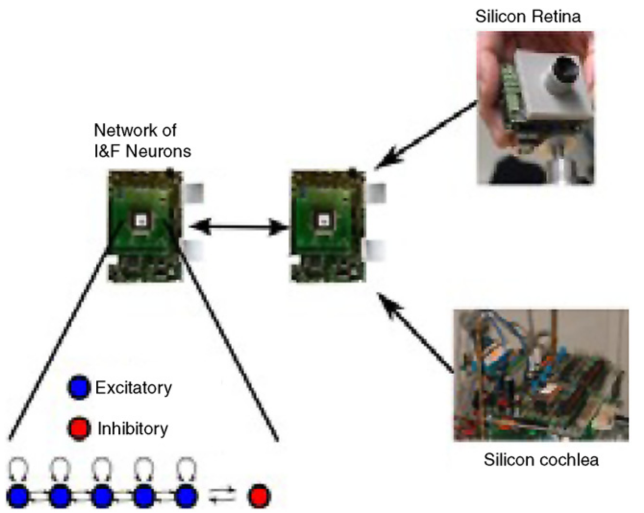

Fig. 9. A schematic diagram of a multi-sensory AER processing neuromorphic architecture to exploit the use of the deSNN learning method for practical applications.

fast, on-line supervised or unsupervised learning, modelling and pattern recognition of SSTD, also suitable for efficient hardware implementation. The model is characterised by the following features:

- one pass propagation of a SSTD during learning;
- evolving and merging neurons and connections in an incremental, adaptive, 'life-long' learning mode;
- utilising dynamic synapses that are modifiable during both learning and recall;
- storing 'history' of learning in terms of initial $\mathbf{w}(0)$ and final $\mathbf{w}(T)$ connection weights in both learning and recall;
- the stored connection weights can be interpreted as clusters of spatio-temporal patterns that can be represented as spatio-temporal fuzzy rules, similar to the rules described in Soltic and Kasabov (2010).

The method is illustrated on two different case studies—moving object recognition using AER data, and EEG frame-based SSTD recognition. Each of these case studies used noisy data due to the character of the processes and the way data is collected. The question how much robust to noise the method is (e.g. what is the critical signal-to-noise ratio after which the method cannot be efficient) is still to be investigated.

The deSNN model worked well on both small number of inputs (e.g. 64) and large number of inputs (e.g. 7000). It was efficient when used on both shorter input patterns (e.g. 500 ms) and medium ones (several seconds), which temporal patterns are typical for fast processes in nature and in the brain. Longer temporal sequences, e.g. minutes, will be attempted in the future in a comparative way using both single deSNN and using a reservoir spatio-temporal SNN filter to capture some spatio-temporal patterns from data before using deSNN as an output module to classify the patterns of the reservoir (see: Kasabov, 2012a, 2012b; Schliebs, Fiasch'e, & Kasabov, 2012; Schliebs, Hamed, & Kasabov, 2011; Verstraeten, Schrauwen, D'Haene, & Stroobandt, 2007).

There are still many open questions and issues for further analysis, e.g.:

- What are the optimal parameter values for a particular task?
- Can the learning process be recovered from the $\mathbf{w}(0)$ and the $\mathbf{w}(T)$ vectors?
- How to evaluate if neurons i and j should be merged based on the weight vector pairs $[\mathbf{w_i}(0), \mathbf{w_i}(T)]$ and $[\mathbf{w_j}(0), \mathbf{w_j}(T)]$?
- What type of dynamic synapse model would be more efficient to use for a given task?
- Which spike should be considered as 'first spike' for RO learning?

A major issue for the future development of deSNN models and systems is the optimisation of its numerous parameters. One way is to combine the local learning of synaptic weights with global optimisation of SNN parameters. Two optimisation approaches will be investigated, namely:

- Using evolutionary computation methods, including: genetic algorithms, particle swarm optimisation, quantum inspired evolutionary computation methods (Defoin-Platel, Schliebs, & Kasabov, 2009; Nuzly, Kasabov, & Shamsuddin, 2010; Schliebs, Defoin-Platel, Worner, & Kasabov, 2009; Schliebs, Kasabov, & Defoin-Platel, 2010). These methods will explore the performance of many deSNN in a population, each having different parameter settings until a close to optimum performing model can be found.
- Using gene regulatory network (GRN) models. Genes and proteins define parameters for brain information processing that has inspired the development of neurogenetic SNN models (Benuskova & Kasabov, 2007; Kasabov, 2010; Kasabov, Benuskova, & Wysoski, 2005; Kasabov, Schliebs, & Kojima, 2011). These models operate at two levels—a GRN level of slow changes of the gene parameter values and SNN level of fast information processing that is affected by the gene parameter changes.

Genes control SNN parameters, but how are gene values optimised? Nature has been continuously optimising genes for millions of years now through evolution. Applying evolutionary algorithms to optimise genes in GRN that control SNN parameters for a specific problem represented as SSTD is a next step in the development of this model.

A further study on the deSNN model will enable more efficient real time applications such as: EEG pattern recognition for BCI (Ferreira et al., 2010; Lalor et al., 2005); fMRI pattern recognition (Sona, Veeramachaneni, Olivetti, & Avesani, 2007); neuro-rehabilitation robotics (Wang et al., 2012), neuro-prosthetics (Isa et al., 2009); cognitive robots (Bellas, Duro, Faiña, & Souto, 2010); personalised modelling (Kasabov & Hu, 2010) for the prognosis of fatal events such as stroke (Barker-Collo, Feigin, Parag, Lawes, & Senior, 2010) and degenerative progression of brain disease, such as AD (Kasabov, 2013; Kasabov et al., 2011).

As STDP learning is now implementable on memristor type electronics and fast image processing hardware (Thorpe, 2012), it makes it feasible to attempt implementation of the deSNN model on such hardware for future fast-, one-pass-, on-line-, real time spatio-temporal pattern recognition tasks.

Acknowledgements

This paper is supported by the EU FP7 Marie Curie project EvoSpike PIIF-GA-2010-272006, hosted by the Institute for Neuroinformatics at ETH/UZH Zurich (http://ncs.ethz.ch/projects/evospike), by the EU ERC Grant "neuroP" (257219), and also by the Knowledge Engineering and Discovery Research Institute (KEDRI, http://www.kedri.info) of the Auckland University of Technology. We acknowledge the discussions with Tobi Delbruck and Fabio Stefanini from INI, Stefan Schliebs and Ammar Mohemmed from KEDRI, and the constructive suggestions by the reviewers of this paper and the organisers of the special issue.

Appendix. Python code of deSNN for the simulation of Example 2

```
############################################
# Authors: N.Nuntallid and K.Dhoble, KEDRI, AUT, NZ 2012
#(www.kedri.info)
############################################
from pylab import *
from brian import *
from brian.utils.progressreporting import ProgressReporter
from time import time
from core.learner import *
from core.utils import *
import os
import operator
############################################
# Parameters and constants for the training set
############################################
defaultclock.dt= 0.2 * ms
### Basic neuron and synapse parameters ###
tau_exc = 2*ms # excitatory synapse time constant
tau_exc_inh = 0.2*ms # feedforward connection time constant
tau_inh = 5*ms # inhibitory synapse time constant
tau_mem = 20*ms # neuron time constant
El = 20*mV # membrane leak
Vthr = 800*mV # spike threshold
Vrst = 0*mV # reset value
winh = 0.20*volt # fixed inhibitory weight
wexc = 0.40*volt # fixed excitatory weight
#wexc_inh = 1 * volt # fixed feedforward excitatory weight
UT = 25*mV # thermal voltage
refr = 4*ms # refractory period
### Learning related parameters ###
Vthm = 0.75*Vthr #5*Vthr/8. # Up/Down weight jumps
tau_ca = 5*tau_mem # Calcium variable time constant
wca = 50 * mV # Steady-state asymptote for Calcium variable
th_low = 1.7*wca # Stop-learning threshold 1 (stop if Vca<thk1)
th_down = 2.2*wca # Stop-learning threshold 2 (stop LTD if Vca>thk2)
th_up = 8*wca–wca # Stop-learning threshold 2 (stop LTP if Vca>thk3)
tau_p = 9* ms # Plastic synapse (NMDA) time constant
wp_hi = 0.6* volt # Plastic synapse high value
wp_lo = 0 * mvolt # Plastic synapse low value
wp_drift = .25 # Bi-stability drift
wp_thr= (wp_hi - wp_lo)/2.+wp_lo # Drift direction threshold
wp_delta = 0.12*wp_hi # Delta Weight
###########Equations#########
eqs_neurons = Equations('''
dv/dt=(El-v+ge+ge_p+ge_inh-gi_out)*(1./tau_mem): volt
dge_p/dt=-ge_p*(1./tau_p): volt
dge/dt=-ge*(1./tau_exc): volt
dgi/dt=-gi*(1./tau_inh): volt
dge_inh/dt=-ge_inh*(1./tau_exc_inh): volt
gi_out = gi*(1-exp(-v/UT)): volt # shunting inhibition
''')
eqs_reset = '''
v=Vrst
'''
#############Architecture of the deSNN ##############
input_size = 5
neurons_class = 1 #Number of neurons in each class
number_class = 2 #Number of class in the output layer
output_size = number_class*neurons_class
out = []
#Connection weights between the input layer and the output layer
SIM_TIME = 8*ms
seed(1)
mod=0.8
######## Read all files from defined directory path #######
path = 'sdsp_testweight/' ## directory path of the input patterns (stimuli)
listing = os.listdir(path)
# Get data Files
for infile in listing:
print "Reading from file: " + infile,"\n##################"
##———Spiketrain stimulus from file——-##
spiketimes=inputfile_to_spikes(path+infile)
########################
s=sorted(spiketimes, key=operator.itemgetter(1))
rankW=zeros((input_size,1))
for i in xrange(len(s)):
rankW[s[i][0]][0]=float(mod**i)
wp0=rankW
print "Rank Order Weights:\n",wp0
##—Convert imported/selected spike trains to Brian (spike train) format–
##
inputSpikeTrain = SpikeGeneratorGroup(input_size, spiketimes)
net = Network(inputSpikeTrain)
net.reinit()
#———— Neurons —————#
# Create Output layer Neurons
neurons = NeuronGroup(N=output_size, model= eqs_neurons, threshold=Vthr, reset= Vrst)#, refractory=refr) # Output layer
# Create Inhibitory neuron group
inh_neurons = NeuronGroup(N=output_size, model = eqs_neurons, threshold = Vthr, reset = Vrst)
#———— Connections ————#
wexc_inh = (0.8+(rand(len(inputSpikeTrain), len(inh_neurons))*0.5)) *volt
c_inter = Connection(inputSpikeTrain, inh_neurons, 'ge_inh', structure = 'dense')
c_inter.connect(inputSpikeTrain, inh_neurons, wexc_inh)
c_inh = Connection(inh_neurons, neurons, 'gi')
c_inh.connect_full(inh_neurons, neurons, weight = winh)
# Connection between the input layer and the output layer
synapses = Connection(inputSpikeTrain, neurons, 'ge_p', structure = 'dynamic')
synapses.connect(inputSpikeTrain, neurons, wp0)
# STDP equation
eqs_stdp='''
x: 1 # fictional presynaptic variable
dC/dt = -C/tau_ca: volt # your postsynaptic calcium variable
V: volt # a copy of the postsynaptic v
'''
stdp=STDP(synapses, eqs=eqs_stdp, pre='w += (V>Vthm)*
(C<th_up)*(th_low<C)*wp_delta - (V<=Vthm)*(C<th_down)*
(th_low<C)*wp_delta; x', post='C += wca; V', wmax=wp_hi)
stdp.post_group.V = linked_var(neurons,'v')
#—————record spike activities——————#
spikes = SpikeMonitor(inputSpikeTrain, record=True)
outspikes = SpikeMonitor(neurons, record=True)
M = StateMonitor(neurons,'v',record=0)
############################################
@network_operation
def drift_equation():
synapses.W = DenseConnectionMatrix(bistable_drift
(synapses.W.todense(), len(inputSpikeTrain), len(neurons)))
def bistable_drift(w, a, b):
w = w.flatten()
```

```
up_idx = w>wp_thr
down_idx = w<=wp_thr
w[up_idx] += wp_drift*defaultclock.dt
w[w>wp_hi] = wp_hi
w[down_idx] -= wp_drift*defaultclock.dt
w[w<wp_lo] = wp_lo
return w.reshape(a,b)
print "SDSP Weights: \n",synapses.W
run(SIM_TIME)
```

References

Arthur, J. V., Merolla, P. A., Akopyan, F., Alvarez, R., Cassidy, A., Chandra, A., et al. (2012). Building block of a programmable neuromorphic substrate: a digital neurosynaptic core. In *Proc. IJCNN 2012*. Brisbane: IEEE Press.

Barker-Collo, S., Feigin, V. L., Parag, V., Lawes, C. M. M., & Senior, H. (2010). Auckland stroke outcomes study. *Neurology*, *75*(18), 1608–1616.

Belatreche, A., Maguire, L. P., & McGinnity, M. (2006). Advances in design and application of spiking neural networks. *Soft Computing*, *11*(3), 239–248.

Bellas, F., Duro, R. J., Faiña, A., & Souto, D. (2010). MDB: artificial evolution in a cognitive architecture for real robots. *IEEE Transactions on Autonomous Mental Development*, *2*, 340–354.

Benuskova, L., & Kasabov, N. (2007). *Computational neurogenetic modelling*. New York: Springer, p. 290.

Bichler, O., Ouerlioz, D., Thorpe, S., Bourgoin, J.-P., & Gamrat, C. (2011). Unsupervised features extraction from asynchronous silicon retina spike-timing-dependent plasticity. In *Proc. IJCNN 2011* (pp. 859–866). IEEE Press.

Bohte, S. M. (2004). The evidence for neural information processing with precise spike-times: a survey. *Natural Computing*, *3*.

Brader, J., Senn, W., & Fusi, S. (2007). Learning real-world stimuli in a neural network with spike-driven synaptic dynamics. *Neural Computation*, *19*(11), 2881–2912.

Brette, R., et al. (2007). Simulation of networks of spiking neurons: a review of tools and strategies. *Journal of Computational Neuroscience*, *23*, 349–398.

Defoin-Platel, M., Schliebs, S., & Kasabov, N. (2009). Quantum-inspired evolutionary algorithm: a multi-model EDA. *IEEE Transactions on Evolutionary Computation*, *13*(6), 1218–1232. Dec..

Delbruck, T. (2007). jAER open source project. http://jaer.wiki.sourceforge.net.

Dhoble, K., Nuntalid, N., Indivery, G., & Kasabov, N. (2012). On-line spatiotemporal pattern recognition with evolving spiking neural networks utilising address event representation, rank oder- and temporal spike learning. In *Proc. WCCI 2012* (pp. 554–560). IEEE Press.

Ferreira, A., Almeida, C., Georgieva, P., Tomé, A., & Silva, F. (2010). Advances in EEG-based biometry. In A. Campilho, & M. Kamel (Eds.), *LNCS*: *vol. 6112*. *ICIAR 2010* (pp. 287–295). Springer.

Fusi, S., Annunziato, M., Badoni, D., Salamon, A., & Amit, (2000). Spike-driven synaptic plasticity: theory, simulation, VLSI implementation. *Neural Computation*, *12*(10), 2227–2258.

Gerstner, W. (1995). Time structure of the activity of neural network models. *Physical Review*, *51*, 738–758.

Guyonneau, R., VanRullen, R., & Thorpe, S. (2005). Neurons tune to the earliest spikes through STDP. *Neural Computation*, *17*(4), 859–879. April 2005.

Hebb, D. (1949). *The organization of behavior*. New York: John Wiley and Sons.

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

Indiveri, G., Chicca, E., & Douglas, R. J. (2009). Artificial cognitive systems: from VLSI networks of spiking neurons to neuromorphic cognition. *Cognitive Computation*, *1*(2), 119–127.

Indiveri, G., & Horiuchi, T. (2011). Frontiers in neuromorphic engineering. *Frontiers in Neuroscience*, *5*, 118.

Indiveri, G., et al. (2011). Neuromorphic silicon neuron circuits. *Frontiers in Neuroscience*, *5*, 1–23.

Indiveri, G., Stefanini, F., & Chicca, E. (2010). Spike-based learning with a generalized integrate and fire silicon neuron. In *2010 IEEE int. symp. circuits and syst., ISCAS 2010* (pp. 1951–1954). Paris, May 30–June 02.

Isa, T., Fetz, E. E., & Muller, K. (2009). Recent advances in brain-machine interfaces. *Neural Networks*, *22*(9), 1201–1202. Brain-Machine Interface.

Izhikevich, E. M. (2004). Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks*, *15*(5), 1063–1070.

Izhikevich, E. M. (2006). Polychronization: computation with spikes. *Neural Computation*, *18*, 245–282.

Jin, X., Lujan, M., Plana, L. A., Davies, S., Temple, S., & Furber, S. (2010). Modelling spiking neural networks on SpiNNaker. *Computing in Science & Engineering*, *12*(5), 91–97. ISSN: 1521-961.

Kasabov, N. (2001). Evolving fuzzy neural networks for on-line supervised/unsupervised, knowledge-based learning. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, *31*(6), 902–918.

Kasabov, N. (2002). *Evolving connectionist systems: methods and applications in bioinformatics, brain study and intelligent machines*. London, New York, Heidelberg: Springer Verlag.

Kasabov, N. (2007). *Evolving connectionist systems — The knowledge engineering approach*. Springer.

Kasabov, N. (2010). To spike or not to spike: a probabilistic spiking neuron model. *Neural Networks*, *23*(1), 16–19.

Kasabov, (2012a). Evolving spiking neural networks and neurogenetic systems for spatio- and spectro-temporal data modelling and pattern recognition. In J. Liu, et al. (Eds.), *LNCS*: *vol. 7311*. *IEEE WCCI 2012* (pp. 234–260). Berlin, Heidelberg: Springer-Verlag.

Kasabov, N. (2012b). NeuCube EvoSpike architecture for spatio-temporal modelling and pattern recognition of brain signals. In Mana, Schwenker, & Trentin (Eds.), *LNAI*: *vol. 7477*. *ANNPR* (pp. 225–243). Springer.

Kasabov, N. (Ed.) (2013). *The springer handbook of bio- and neuroinformatics*. Springer.

Kasabov, N., Benuskova, L., & Wysoski, S. (2005). A computational neurogenetic model of a spiking neuron. In *IJCNN 2005 conf. proc. vol. 1* (pp. 446–451). IEEE Press.

Kasabov, N., & Hu, Y. (2010). Integrated optimisation method for personalised modelling and case study applications. *International Journal of Functional Informatics and Personalised Medicine*, *3*(3), 236–256.

Kasabov, N., Schliebs, R., & Kojima, H. (2011). Probabilistic computational neurogenetic framework: from modelling cognitive systems to alzheimer's disease. *IEEE Transactions on Autonomous Mental Development*, *3*(4), 300–311.

Kasabov, N., & Song, Q. (2002). DENFIS: dynamic, evolving neural-fuzzy inference systems and its application for time-series prediction. *IEEE Transactions on Fuzzy Systems*, *10*, 144–154.

Kistler, G., & Gerstner, W. (2002). *Spiking neuron models—single neurons, populations, plasticity*. Cambridge Univ. Press.

Lalor, E., Kelly, S., Finucane, C., Burke, R., Smith, R., Reilly, R., et al. (2005). Steady-state vep-based brain–computer interface control in an immersive 3D gaming environment. *EURASIP Journal on Applied Signal Processing*, *2005*, 3156–3164.

Lichtsteiner, P., & Delbruck, T. (2005). A 64×64 AER logarithmic temporal derivative silicon retina. *Research in Microelectronics and Electronics*, *2*, 202–205.

Maass, W., & Markram, H. (2002). Synapses as dynamic memory buffers. *Neural Networks*, *15*(2), 155–161.

Maass, W., Natschlaeger, T., & Markram, H. (2002). Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural Computation*, *14*(11), 2531–2560.

Maass, W., & Sontag, E. D. (2000). Neural systems as nonlinear filters. *Neural Computation*, *12*(8), 1743–1772.

Maass, W., & Zador, A. M. (1999). Computing and learning with dynamic synapses. In *Pulsed neural networks* (pp. 321–336). MIT Press.

Masquelier, T., Guyonneau, R., & Thorpe, S. J. (2008). Spike timing dependent plasticity finds the start of repeating patterns in continuous spike trains. *PLoS ONE*, *3*(1), e1377.

Masquelier, T., Guyonneau, R., & Thorpe, S. J. (2009). Competitive STDP-based spike pattern learning. *Neural Computation*, *21*(5), 1259–1276.

Mehrtash, N., Jung, D., & Klar, H. (2003). Image pre-processing with dynamic synapses. *Neural Computing & Applications*, *12*, 33–41 http://dx.doi.org/10.1007/s00521-030-0371-2.

Mitra, S., Fusi, S., & Indiveri, G. (2009). Real-time classification of complex patterns using spike-based learning in neuromorphic VLSI. *IEEE Transactions on Biomedical Circuits and Systems*, *3*(1), 32–42.

Moradi, S., & Indiveri, G. (2011). A VLSI network of spiking neurons with an asynchronous static random access memory. In *Biomedical circuits and systems conference BIOCAS 2011*.

Morrison, A., Diesmann, M., & Gerstner, W. (2008). Phenomenological models of synaptic plasticity based on spike timing. *Biological Cybernetics*, *98*, 459–478.

Namarvar, H., Liaw, J.-S., & Berger, T. (2001). A new dynamic synapse neural network for speech recognition. In *Proc. IJCNN'01. int. joint conf. on neural networks, 2001*.

Natschläger, T., & Maass, W. (2002). Spiking neurons and the induction of finite state machines. *Theoretical Computer Science—Natural Computing*, *287*(1), 251–265.

Neuper, C., Muller, G., Kubler, A., Birbaumer, N., & Pfurtscheller, G. (2003). Clinical application of an EEG -based brain–computer interface: a case study in a patient with severe motor impairment. *Clinical Neurophysiology*, *114*(3), 399–409.

Nuntalid, N., Dhoble, K., & Kasabov, N. (2011). EEG classification with BSA spike encoding algorithm and evolving probabilistic spiking neural network. In *LNCS*: *vol. 7062*. *Proc. 18th int. conf. on neural information processing, ICONIP, 2011, Shanghai, China* (pp. 451–460). Heidelberg: Springer.

Nuzly, H., Kasabov, N., & Shamsuddin, S. (2010). Probabilistic evolving spiking neural network optimization using dynamic quantum inspired particle swarm optimization. *Australian Journal of Intelligent Information Processing Systems*, *11*(1).

Schliebs, S., Defoin-Platel, M., Worner, S., & Kasabov, N. (2009). Integrated feature and parameter optimization for evolving spiking neural networks: exploring heterogeneous probabilistic models. *Neural Networks*, *22*, 623–632.

Schliebs, S., Fiasch'e, M., & Kasabov, N. (2012). Constructing robust liquid state machines to process highly variable data streams. In A. Vila, et al. (Eds.), *LNCS*: *vol. 7552*. *ICANN 2012* (pp. 604–611). Springer.

Schliebs, S., Hamed, H. N. A., & Kasabov, N. (2011). A reservoir-based evolving spiking neural network for on-line spatio-temporal pattern learning and recognition, Neural Information Processing. In *LNCS*: *vol. 7063*. *Proc. 18th int. conf. neural information processing*, ICONIP, (pp. 160–168). Shanghai, China, Heidelberg: Springer.

Schliebs, S., & Kasabov, N. (2012). *Evolving spiking neural networks: a survey, evolving systems*. Springer.

Schliebs, S., Kasabov, N., & Defoin-Platel, M. (2010). On the probabilistic optimization of spiking neural networks. *International Journal of Neural Systems*, *20*(6), 481–500.

Schrauwen, B., & Van Campenhout, J. (2003). BSA—a fast and accurate spike train encoding scheme. In *2003 Proc. int. joint conf. on neural networks. Vol. 4* (pp. 2825–2830). IEEE.

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.

Sona, D., Veeramachaneni, S., Olivetti, E., & Avesani, P. (2007). Inferring cognition from fMRI brain images. In J. M. de Sá, L. A. Alexandre, W. Duch, & D. P. Mandic (Eds.), *LNCS*: *vol. 4669*. *ICANN 2007* (pp. 869–878). Springer.

Song, S., Miller, K., Abbott, L., et al. (2000). Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nature Neuroscience*, *3*, 919–926.

Tanaka, K., Matsunaga, K., & Wang, H. (2005). Electroencephalogram-based control of an electric wheelchair. *IEEE Transactions on Robotics*, *21*(4), 762–766.

Thorpe, S. J. (2009). The speed of categorization in the human visual system. *Neuron*, *62*(2), 168–170.

Thorpe, S. J. (2012). Spike-based image processing: can we reproduce biological vision in hardware. *Lecture Notes in Computer Science*, *vol. 7583*, 516–521.

Thorpe, S.J., Brilhault, A., & Perez-Carrasco, J.A. (2010). Suggestions for a biologically inspired spiking retina using order-based coding. In *IEEE international symposium on circuits and syst.* (pp. 265–268).

Thorpe, S., & Gautrais, J. (1998). Rank order coding. *Computational Neuroscience*, *13*, 113–119.

Thorpe, S., Guyonneau, R., Guilbaud, N., Allegraud, J. M., & Vanrullen, R. (2004). SpikeNet: real-time visual processing with one spike per neuron. *Neurocomputing*, *58–60*, 857–864.

Tsodyks, M., Pawelzik, K., & Markram, H. (1998). Neural networks with dynamic synapses. *Neural Computation*, *10*(4), 821–835.

van Schaik, A., & Shih-Chii Liu, L. (2005). AER EAR: a matched silicon cochlea pair with address event representation interface. In *Proc. of ISCAS—IEEE int. symp. on circuits and systems, Vol. 5* (pp. 4213–4216). May.

Verstraeten, D., Schrauwen, B., D'Haene, M., & Stroobandt, D. (2007). An experimental unification of reservoir computing methods. *Neural Networks*, *20*(3), 391–403.

Wang, X., Hou, Z.-G., Tan, M., Wang, Y., Lv, F., & Kasabov, N. (2012). Mobile robots' target—reaching controller based on spiking neural networks. In *LNCS*, *Proc. ICONIP 2012*. Qatar: Springer.

Watts, M. (2009). A decade of Kasabov's evolving connectionist systems: a review. *IEEE Transactions on Systems, Man, and Cybernetics Part C: Applications and Reviews*, *39*(3), 253–269.

Wolpaw, J., McFarland, D., & Vaughan, T. (2000). Brain–computer interface research at the wadsworth center. *IEEE Transactions on Rehabilitation Engineering*, *8*(2), 222–226.

Wysoski, S., Benuskova, L., & Kasabov, N. (2010). Evolving spiking neural networks for audiovisual information processing. *Neural Networks*, *23*(7), 819–835.