Event-driven implementation of deep spiking convolutional neural networks for supervised classification using the SpiNNaker neuromorphic platform

Alberto Patiño-Saucedo [a,*], Horacio Rostro-Gonzalez [a], Teresa Serrano-Gotarredona [b], Bernabé Linares-Barranco [b]

[a] Department of Electronics Engineering, University of Guanajuato, Salamanca, Mexico
[b] Instituto de Microelectrónica de Sevilla (IMSE-CNM), CSIC, Seville, Spain

Keywords:
Neuromorphic hardware
Artificial neural networks
Spiking neural networks
MNIST
SpiNNaker
Event processing

A B S T R A C T

Neural networks have enabled great advances in recent times due mainly to improved parallel computing capabilities in accordance to Moore's Law, which allowed reducing the time needed for the parameter learning of complex, multi-layered neural architectures. However, with silicon technology reaching its physical limits, new types of computing paradigms are needed to increase the power efficiency of learning algorithms, especially for dealing with deep spatio-temporal knowledge on embedded applications. With the goal of mimicking the brain's power efficiency, new hardware architectures such as the SpiNNaker board have been built. Furthermore, recent works have shown that networks using spiking neurons as learning units can match classical neural networks in supervised tasks. In this paper, we show that the implementation of state-of-the-art models on both the MNIST and the event-based NMNIST digit recognition datasets is possible on neuromorphic hardware. We use two approaches, by directly converting a classical neural network to its spiking version and by training a spiking network from scratch. For both cases, software simulations and implementations into a SpiNNaker 103 machine were performed. Numerical results approaching the state of the art on digit recognition are presented, and a new method to decrease the spike rate needed for the task is proposed, which allows a significant reduction of the spikes (up to 34 times for a fully connected architecture) while preserving the accuracy of the system. With this method, we provide new insights on the capabilities offered by networks of spiking neurons to efficiently encode spatio-temporal information.

1. Introduction

In the last few years, progress in the field of Artificial Neural Networks (ANNs) has led it to take a central role in solving Artificial Intelligence problems, outperforming other machine learning approaches such as kernel machines in highly complex tasks of computer vision, speech recognition, natural language processing, among others (Schmidhuber, 2015). Even though ANNs have been studied for decades, their widespread use and development was restricted by their high computational cost. Their late success came on a par with the sustained exponential growth in computing capacity as predicted by Moore (1998). This allowed the cornerstone learning algorithm of ANNs, backpropagation (Rumelhart, Hinton, & Williams, 1985), to solve the weight assignment problem across multiple computational stages, or layers, giving rise to Deep Learning. The success of Deep ANNs lies in their ability to discover increasingly optimal representations of data, encoded in hierarchical structures, with no need for humans to specify all the knowledge needed by the system (Goodfellow, Bengio, & Courville, 2016).

Many commercial, medical and scientific applications of deep ANNs can be found nowadays. One clear example is the face, fingerprint and voice recognition performed by smart-phones through the inference of deep ANN models, generally trained on external servers. However, with the upcoming end of Moore's Law, and to push the capabilities of deep learning forward, more power-efficient ways to train and deploy deep ANNs need to be achieved. Currently, training relies on massively parallel computing. The number of connections, power consumption and the required memory and computation time have limited the use of resource-intensive deep learning algorithms directly in embedded systems (LeCun, Bengio, & Hinton, 2015). With this in

* Corresponding author.
E-mail addresses: alberto.patino@ugto.mx (A. Patiño-Saucedo), hrostrog@ugto.mx (H. Rostro-Gonzalez), terese@imse-cnm.csic.es (T. Serrano-Gotarredona), bernabe@imse-cnm.csic.es (B. Linares-Barranco).

mind, new ways to learn with more efficiency are being discussed and new paradigms of computation such as quantum and neuromorphic computing have emerged.

Neuromorphic computing is a novel technology that seeks to emulate the wiring and processes of the human brain in hardware. It has been boosted by two major projects: the Human Brain Project (Europe) (Markram, 2012), which uses the SpiNNaker (Furber, Galluppi, Temple, & Plana, 2014) and the BrainScaleS (Schmitt et al., 2017) neuromorphic chips, and the BRAIN initiative (USA) (Jorgenson et al., 2015). Likewise, major hardware companies conduct active research on neuromorphic chips, with IBM (TrueNorth) (Akopyan et al., 2015) and Intel (Loihi) (Davies et al., 2018) standing out. One trend is to emulate cortical cell structures as accurately as possible and expect for emergent properties of intelligence to arise. Other more straightforward approach is to seek a convergence between neuromorphic and deep learning technologies. Both views agree on the use of Spiking Neurons, models for neural simulations that capture a fundamental property of biological neurons missing in ANNs: the use of spikes, or binary events, which enable an efficient way of modeling spatio-temporal data (Kasabov, 2018).

Spiking Neural Networks (SNNs) are being studied with the hope to get energy-efficient representations of the world, inspired in the brain's high memory capacity, noise robustness, and task complexity on low power consumption. Methods for training Deep Spiking Neural Networks (DSNNs) have appeared as a natural bridge between neuromorphic computing and deep learning, and several algorithms have been proposed for implementing spiking versions of Fully Connected and Convolutional Neural Networks (CNNs) (Cao, Chen, & Khosla, 2015; Rueckauer, Lungu, Hu, & Pfeiffer, 2016), Restricted Boltzmann Machines (Neftci, Das, Pedroni, Kreutz-Delgado, & Cauwenberghs, 2014), Deep Belief Networks (O'Connor, Neil, Liu, Delbruck, & Pfeiffer, 2013; Stromatias et al., 2015) and Recurrent Neural Networks (Shrestha et al., 2017). Focusing on the most widely used learning algorithm of Deep Learning, backpropagation, it has been successfully applied to train spiking CNNs, with approximations such as SpikeProp (Bohte, Kok, & La Poutre, 2002), ReSuMe (Ponulak & Kasiński, 2010), SLAYER (Shrestha & Orchard, 2018) and Spatio-Temporal Backpropagation (STBP) (Wu, Deng, Li, Zhu, & Shi, 2018). Furthermore, methods for direct conversion from pre-trained non-spiking CNNs to SNNs have been proposed (Diehl et al., 2015; Rueckauer et al., 2016), with results matching the state of the art in supervised classification on benchmark datasets.

Most authors of the aforementioned frameworks for training or converting DSNNs hinted on the convenience and feasibility to deploy their networks in neuromorphic hardware. Cao et al. (2015), Rueckauer et al. (2016), Stromatias et al. (2015) and Wu et al. (2018) let the deployment of their proposed frameworks as future work. Shrestha and Orchard (2018) pointed out the difficulty of performing the training phase of SNNs on current neuromorphic chips, leaving only room for just performing inference of their method. Cao advocates for demonstrating the power efficiency of neuromorphic implementations of DSNNs. The implicit consensus is that the current state of development of neuromorphic chips would only allow the implementation of DSNN systems with two separate stages: one for offline training/conversion and other for online neuromorphic inference.

Among the works proposing implementations of DSNNs on neuromorphic chips, Esser, Appuswamy, Merolla, Arthur, and Modha (2015) implemented a sparsely connected neural network on the TrueNorth chip achieving a maximum 99.42% classification accuracy on the MNIST dataset, being the best reported score in this and any neuromorphic platform. This work also reported a tradeoff between energy efficiency and classification accuracy. Schmitt et al. (2017) implemented a DSNN on the BrainScales system, reaching a maximum accuracy of 95% on the MNIST. Regarding the SpiNNaker platform, in an early attempt, Jin et al. (2010) deployed a non-spiking Multilayer Perceptron Network, without testing on benchmark datasets. Serrano-Gotarredona, Linares-Barranco, Galluppi, Plana, and Furber (2015) implemented a CNN for symbol recognition, with events as inputs, achieving 80% accuracy. Stromatias et al. (2015) deployed a spiking Deep Belief Network, reaching 95% on the MNIST dataset, and Liu et al. (2018) deployed an energy efficient non-spiking Deep Neural Network with online training, achieving 96% on the MNIST.

In this work, we show a SpiNNaker implementation of the popular LeNet architecture, including approximated pooling layers and Relu activations, by the method of direct conversion as suggested by Rueckauer et al. (2016). This network reaches 98.20% on the MNIST dataset, beating the best reported accuracy on the SpiNNaker platform. Additionally, we show the first neuromorphic implementation of an event-based digit classifier, by deploying a network trained with the STBP algorithm for the N-MNIST dataset, reaching 97.92% accuracy. Both networks were simulated using PyNN, a neural simulation platform, and a comparison between ANN implementation, SNN simulation and final hardware deploying is provided. Finally, we propose a modification of the cost function of the STBP algorithm in order to reduce the average spike rate necessary for classification, achieving a 19 times reduction on the LeNet architecture and a 34 times reduction for a densely connected SNN with drops in the classification accuracy of less than 2% and 1%, respectively. The reduction on the number of spikes needed to perform the classification task is important to achieve more energy-efficient inference, as is shown by a further experiment on the SpiNNaker where the inference time per input sample is reduced by up to 6%.

Furthermore, this work presents the first implementation on neuromorphic hardware of a SNN trained with the widely used PyTorch (Paszke et al., 2017) framework, with a performance comparison between software and hardware implementations, broadening the scope of the deep SNN architectures that can be tested on the SpiNNaker platform.

2. Materials and methods

2.1. SNN model and simulation

The spiking neuron model used in this work is an instance of the commonly used Leaky Integrate-and-Fire (LIF), suitable for very efficient implementations. The dynamics of the membrane potential $u(t)$ of a single neuron is given by:

$$\frac{du(t)}{dt} = \frac{u_{rest} - u(t)}{\tau_m} + \frac{I(t)}{c_m} \tag{1}$$

where u_{rest} is the resting potential, τ_m is the membrane's time constant, c_m is the membrane capacitance and $I(t)$ is the neuron's input current.

By injecting a small input current pulse of duration Δt, starting at $t = 0$, with initial membrane potential at rest state equal to zero, $u(0) = u_{rest} = 0$, the neuron's membrane is 'charged' during the stimulus and 'discharged' when it ends. For discrete simulation purposes, this Δt is taken as the sampling period of the discrete time. The response of the neuron to a small pulse or spike in the time k corresponds to the discharge of the neuron, yielding the discrete update equation of the membrane potential:

$$u[k+1] = u[k]e^{-\frac{\Delta t}{\tau_m}} \tag{2}$$

Table 1
Summary of the main features of the toolbox.

Supported features	Keras [K] Lasagne [L] Caffe [C] (input)	Brian2 [B] pyNN [P] MegaSim [M] INIsim [I] (output)
Fully connected	All	All
Convolutional	All	All
Max-Pooling	All	I
Average-Pooling	All	All
Batch-Normalization	All	All
Dropout	All	All
Flatten	All	All
Merge/Concatenate (Inception modules)	K, L	I
Linear activation	All	Replaced by ReLU
ReLU activation	All	All
Softmax activation	All	I
Binary activation {-1, 1} or {0, 1}	L	I
Binary weights {-1, 1}	L	All
Non-zero biases	All	I

In the case of multiple pre-synaptic connections to the neuron, its input current is computed as the cumulative effect of pre-synaptic spikes:

$$I[k] = \sum_{j=1}^{M} w_j \theta_j[k] + I_{bias} \tag{3}$$

where M is the number of pre-synaptic neurons (with membrane potentials u_j), w_j is the synaptic strength from the jth pre-synaptic neuron (positive if the synapse is excitatory and negative if inhibitory), I_{bias} is an offset current and $\theta_j[k]$ denotes the occurrence of a spike on the jth pre-synaptic neuron in the current timestep k. Each neuron fires whenever the membrane potential surpasses a threshold u_{th}. The spike is computed by all post-synaptic neurons in the next time-step, and its membrane potential is reset to u_{reset}. The spike function is thus given by:

$$\theta_j[k] = \begin{cases} 1 & u_j[k-1] \geq u_{th} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

One important metric of a spiking neuron's state used in this work is its firing rate. By definition, the firing rate $r[k]$ of a given neuron whose simulation started at time $k = 0$ is:

$$r[k] = \frac{\sum_t \theta[k]}{k} \tag{5}$$

For running experiments with the spiking neural model used in this work, we used PyNN (Davison, Brüderle, Eppler, Kremkow, Muller, Pecevski, et al., 2009), a high level spiking neuron interface supporting experiments across multiple simulators (e.g. BRIAN, NEST, NEURON) making their scripts highly portable. Most importantly, PyNN can be used as an interface between high level modeling and hardware implementation into the SpiNNaker platform. More details on the PC-based and SpiNNaker-based PyNN implementations are given in Section 2.5.

2.2. ANN to SNN conversion

Given a fully connected ANN with L layers, each with M^l neurons or units let W^l, $l \in \{1, \ldots, L\}$ denote the weight matrix connecting units between layer $l-1$ and layer l. The ReLU activation ${a_i}^l$ for each i unit of layer l with bias ${b_i}^l$ is given by:

$${a_i}^l = max(\{0, {z_i}^l\}) \tag{6}$$

$${z_i}^l = \sum_{j=1}^{M^{l-1}} W_{ij}^l a_j^{l-1} + b_i^l \tag{7}$$

The main objective of the ANN to SNN conversion is to take a pre-trained ANN and create an analogous SNN with the same connectivity (i.e. one-to-one correspondence among ANN and SNN units) where the firing rate r_i^l of every spiking neuron is proportional to the value of the activation a_i^l of the corresponding artificial neuron. This is performed by exploiting the fact that ReLU activations in ANNs, such as spiking rates in SNNs, are always positive. The conversion is performed by computing a scaled version of W^l and taking it as the synaptic weight matrix of the corresponding layers of the SNN. The scaling is needed to ensure that for every layer of the ANN, two conditions are satisfied: (1) if ${z_i}^l < 0$, the effect of presynaptic spikes to the membrane potential of neuron i does not make it fire, and (2) $\max(\{a_1^l, \ldots, a_{M^l}^l\})$ do not surpass the maximum firing rate of the spiking neuron simulation, set to 1 kHz.

In Rueckauer, Lungu, Hu, Pfeiffer, and Liu (2017), authors proposed a method to convert ANNs into SNNs for image classification, and released a toolbox supporting the conversion from ANN models defined in different platforms such as Keras, Lasagne and Caffe, returning the synaptic weights to be used on SNN simulators such as PyNN. It supports a number of commonly used features of ANNs, such as Convolutional and Batch-Normalization layers, and ReLU, Softmax and binary activations, among others. A summary of the main features of the conversion toolbox is provided by the authors in the website[1] and presented in Table 1.

This toolbox has been used here to convert a modified LeNet CNN for digit recognition into its corresponding SNN to evaluate its performance in PC simulation and hardware implementation by running the equivalent PyNN codes into the SpiNNaker platform. Numerical results are presented in Section 3.

The MNIST dataset (Lecun, Bottou, Bengio, & Haffner, 1998) for digit recognition was used to test the ANN to SNN conversion approach, consisting of 60,000 training and 10,000 test images of handwritten digits. This dataset is a standard benchmark to test the performance of machine learning algorithms, and it has also been used for classification tasks on neuromorphic chips. See Table 4 for a survey of the neuromorphic chip implementations of SNNs for the MNIST dataset.

In this work, the ANN to SNN conversion process was made by taking a pre-trained LeNet CNN for MNIST and converting it to a Convolutional SNN. The architecture of the LeNet network consists of one input layer, three convolutional layers and two fully connected layers (including the output layer), with pooling operations in between. A scheme of the architecture is shown in Fig. 3. The converted equivalent SNN network was simulated in PyNN and implemented on the SpiNNaker chip following the process shown in Fig. 1.

[1] http://snntoolbox.readthedocs.io/en/latest/guide/intro.html

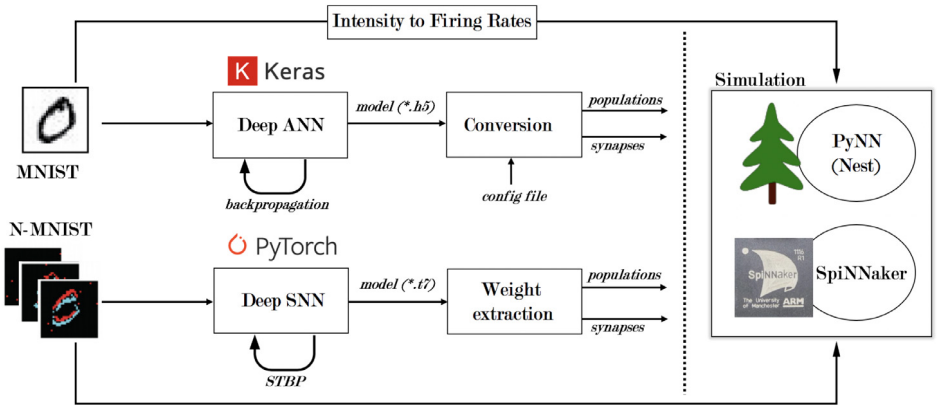

Fig. 1. Scheme of the different stages of the neuromorphic digit classification systems proposed in this work. The upper part depicts the conversion system and the lower part the STBP-trained system.

Table 2
Simulation parameters for the converted and STBP trained SNN model.

Parameter	Conversion	STBP Training
u_{reset} (mV)	0.0	0.0
u_{rest} (mV)	0.0	0.0
u_{th} (mV)	1.0	0.3
τ_m (ms)	1000	0.8325
c_m (nF)	0.09	0.001
Δ_t (ms)	1.0	1.0

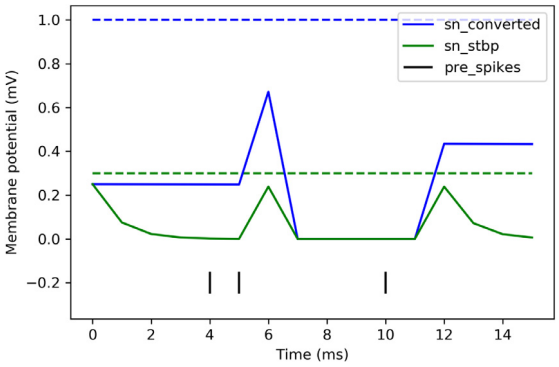

Fig. 2. Responses to pre-synaptic spikes for both kind of neurons considered in this work. Dashed horizontal lines represent the thresholds. Note that both neurons emit a spike between 6 and 7 ms.

2.3. SNN training with spatio temporal back propagation

Conversion methods such as the one discussed above force the SNN model to focus its attention on the spatial domain information. Spiking neuron parameters of the simulator for the standard conversion method as proposed by Rueckauer et al. (2017), neglect the temporal or "memory" effect of the membrane by setting a high value on its time constant τ_m (refer to Table 2). In contrast, the method proposed by Wu et al. (2018), denominated Spatio Temporal Back Propagation (STBP) allows a more complete treatment of the temporal domain by training the SNN with a time-dependent generalization of the ANN's backpropagation algorithm. Here, as in the conversion method, the neuron's activity is determined by its firing rate. The algorithm uses a loss function ℓ across S training samples and a time window T:

$$\ell = \frac{1}{S}\sum_{s=1}^{S}\left\| \boldsymbol{y_s} - \frac{1}{T}\sum_{t=1}^{T}\boldsymbol{\theta_{s,L}} \right\|_2^2 \tag{8}$$

where $\boldsymbol{y_s}$ and $\boldsymbol{\theta_{s,L}}$ are the label vector of the sth training sample and its corresponding spike activity vector in the output layer (last layer L) after forward propagation, respectively.

We used an implementation of this algorithm provided by the authors. The neural network model is described in PyTorch (Paszke et al., 2017), an open source deep learning platform. After training any deep SNN model, the platform allows the extraction of the final weights and biases. These are used for reproducing the results from PyTorch with PyNN and the subsequent implementation on SpiNNaker, provided the connectivity, i.e. populations and projections in PyNN are equivalent to the PyTorch tensor operations pertaining the whole model. This way, the extracted weights are used as synaptic weights in PyNN, and the extracted biases are used as offsets for the threshold u_{th}, with exact same values. A comparison of the neuron parameters for the PyNN implementations of both methods, conversion and training, is given in Table 2. As seen in Fig. 2, the decay of the membrane potential in the model used by the STBP method is more realistic than that used by the conversion method.

The N-MNIST dataset (Orchard, Jayawant, Cohen, & Thakor, 2015) is an event-based version of the MNIST dataset, where each sample was displayed in a monitor and recorded with a Dynamic Vision Sensor (DVS) mounted in a motorized pan–tilt unit performing a saccade movement. The sensor records a spike whenever a change of illumination is detected. The spatial dimension is the same as that of the MNIST dataset, 28 × 28 pixels. According to the work by Wu et al. (2018), for every sample we take both, positive and negative change of illumination, as two different channels and feed them to a 400-400-10 fully connected SNN trained with the STBP algorithm. Afterward, the trained weights are used for a simulation of the SNN on PyNN and implementation on SpiNNaker. Results for both experiments are presented in Section 3.

2.4. SNN training with spike regularization

One of the premises that has boosted research on SNNs is the hope to make computations more energy-efficient when implemented on event-driven neuromorphic hardware in comparison with their frame-based counterparts. This would be possible due to the characteristics of neuromorphic devices, which allow to keep and update the state of every neuron independently, without the need for a general clock, i.e, computing spikes asynchronously. It is known that the conditional multiply-accumulate operation in each synapse is the driver of neural computations in neuromorphic hardware (Merolla et al., 2011). This means

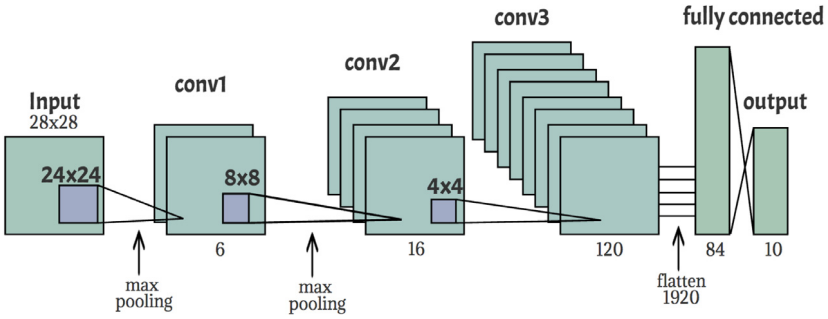

Fig. 3. LeNet CNN Architecture used for direct conversion into Deep Convolutional SNN.

that the spike rates and the number of active synapses can be used to estimate the energy consumption of such devices. By taking this into account, Cao et al. (2015) conducted an analysis of power consumption for its spiking CNN module, assuming a direct relation between the spike count and the consumed power. In this work, we adopt this approach and propose a modification of the cost function of the STBP algorithm to decrease the number of spikes. This modification acts as an spike activity regularization, analog to the weight regularization that is commonly used when training classical neural networks. With the goal of not only to achieve better generalization in the classification task, but a reduced spike activity, we introduce the new loss function:

$$\ell = \frac{1}{S}\sum_{s=1}^{S}\left(\left\|\boldsymbol{y_s} - \frac{1}{T}\sum_{t=1}^{T}\boldsymbol{\theta_{s,L}}\right\|_2^2 + \frac{\lambda_{sr}}{NT}\sum_{t=1}^{T}\sum_{l=1}^{L-1}\boldsymbol{\theta_{s,l}}\right) \tag{9}$$

This cost function computes for every sample the amount of spikes elicited in a time window T of all the neurons, except those in the output layer. The scaling factor N, equal to the total number of hidden units, is used to ensure both terms in the equation are in the same scale. The spike regularization factor $\lambda_{sr} \in [0, 1]$ is added for control. λ can be interpreted as a compromise between network's accuracy and spike economy.

We conduct experiments with six different values of spike regularization on both the fully connected and the convolutional network and report the results in Section 3.2.

2.5. Spiking neural network architecture (SpiNNaker)

SpiNNaker is a massively parallel multicore computing system designed for modeling very large spiking neural networks in real time. Both the system architecture and the design of the SpiNNaker chip were developed by the Advanced Processor Technologies Research Group (APT) at the University of Manchester. Each SpiNNaker chip consists of 18 fully programmable ARM cores.

In this work, a SpiNNaker 103 machine (Fig. 4) was used. This board comprises 48 SpiNNaker chips, totaling 864 ARM processor cores deployed as 48 monitor processors, 768 application cores and 48 spare cores. Each application core has two types of RAM: a 32 kB ITCM (instruction tightly coupled memory) for storing instructions and a 64 kB DTCM (data tightly coupled memory) for storing neuron states and parameters. Additionally, each SpiNNaker chip contains a 128 MB SDRAM shared by the 18 cores for storing the synaptic weights. The communication between cores is done through a multicast packet-routing mechanism that mimics the high connectivity found in biological brains (Cuevas-Arteaga et al., 2017). A 100 Mbps Ethernet connection is used for controlling an I/O interface between the computer and the SpiNNaker board. The neurons and synapses are modeled with sPyNNaker (Rhodes et al., 2018), a software package for simulating PyNN-defined spiking neural networks on the SpiNNaker platform. Two SpiNNaker implementations were performed for each proposed approach, as shown in the diagram of Fig. 1.

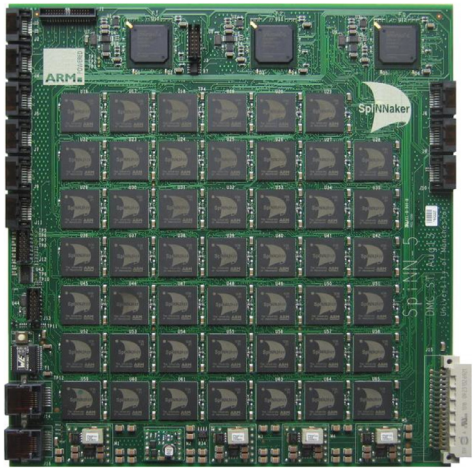

Fig. 4. SpiNNaker 103 machine.

3. Results

3.1. SpiNNaker implementation

We have implemented two Deep SNNs on the SpiNNaker Neuromorphic platform for a handwritten digit classification task. The first is previously trained on Keras as a classical CNN (LeNet) with a static dataset (MNIST), then converted into Deep SNN with the snntoolbox. The second is trained as a Deep SNN on PyTorch using the novel STBP algorithm. The input of the second network is an event-based equivalent of the MNIST dataset, recorded with a DVS camera. For measuring the performance of the implementation, the whole test set (10,000 samples) was propagated in both software and hardware, measuring the classification accuracy as the percentage of correctly detected digits. In the case of the SpiNNaker implementation, 15 ms of activity were recorded. An additional neural simulation on the PyNN (Nest) software was performed for both SNNs, using 1000 samples. The real-time implementation takes approximately 0.4 s per sample in the neuromorphic hardware and 10 s per sample in the neural simulation software.

An example of the activity during inference of the STBP trained network, both in software (PyTorch, PyNN) and hardware (SpiNNaker) is shown in Fig. 5. The figure shows the spike times of all neurons in the network for ten presented samples. An image is considered to have been classified correctly if the neuron associated with the input digit displays the highest activity of all the output layer neurons.

Numerical results are shown in Table 3. A comparison with other neuromorphic MNIST (Table 4) implementations on neuromorphic hardware shows that ours achieve the second overall

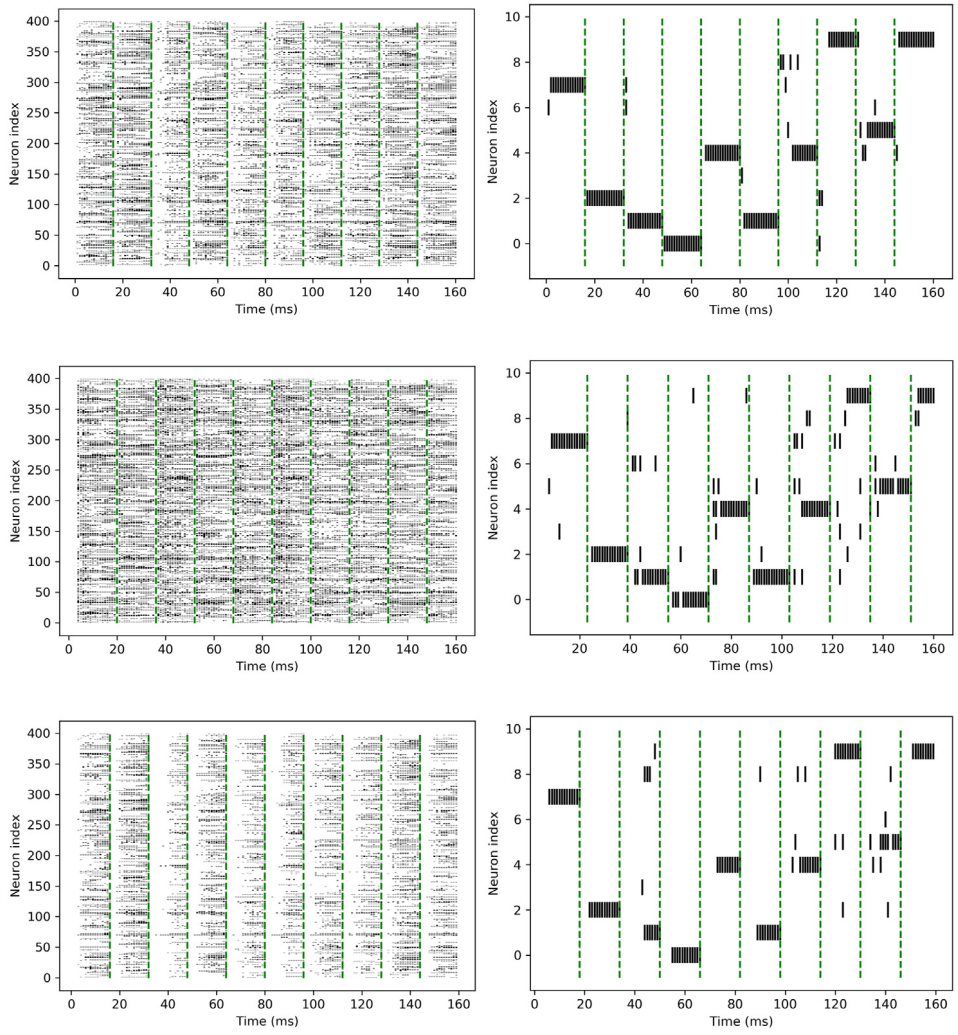

Fig. 5. Raster plots for the Deep SNN simulation of ten NMNIST samples using PyTorch (top), PyNN (center) and SpiNNaker (bottom). Each sample was propagated 16 ms, as indicated by the vertical lines. The corresponding labels are, from left to right: 7, 2, 1, 0, 4, 1, 4, 9, 5, 9.

Table 3
Comparison of classification accuracy for the proposed methods and datasets.

	MNIST/Conversion	NMNIST/STBP
Keras/Pytorch	98.96	98.50
PyNN (Nest)	97.98	97.04
SpiNNaker	98.2	97.92

Table 4
Summary of spiking deep learning models implemented on neuromorphic hardware and their accuracy on MNIST. The column *T* is true if the hardware performs online training. The column *S* is true if the model uses spikes internally.

Model	Hardware	Network Arch.	S	T	Acc
Merolla et al. (2011)	Custom core	Spiking RBM	Yes	No	94.00
Neil and Liu (2014)	FPGA	Spiking DBN	Yes	No	92.00
Garbin et al. (2014)	OxRAM device	DSNN (ConvNet)	Yes	No	94.00
Stromatias et al. (2015)	SpiNNaker	Spiking DBN	Yes	No	95.00
Esser et al. (2015)	TrueNorth	DSNN (Sparse)	Yes	No	**99.42**
Schmitt et al. (2017)	BrainScales	DSNN (Dense)	Yes	Yes	95.00
Liu et al. (2018)	SpiNNaker	Deep Rewiring	No	Yes	96.00
Ours	SpiNNaker	DSNN (ConvNet)	Yes	No	**98.20**

result, with 98.2% correct classification, and the best on the SpiNNaker platform. Furthermore, to the best of our knowledge, this work presents the first neuromorphic hardware implementation of the event-based NMNIST benchmark, with 97.92% correct classification.

3.2. Spike regularization

We report the effect of the proposed spike regularization of the STBP's cost function on the fully connected 400-400-10 (labeled Dense400) and the LeNet convolutional network for six different values of spike regularization (λ_{sr} in Eq. (9)) ranging from $\lambda_{sr} = 0$ (no regularization) to $\lambda_{sr} = 1$. After training both network architectures with PyTorch, the entire test set was propagated and measurements of the average spike rate per neuron were recorded. Additionally, the fully connected network was loaded into the SpiNNaker platform and measurements of the simulation time for 100 samples per experiment were performed. The simulation parameters of STBP training from Table 2 remained unchanged. Numerical results are given in Table 5. Graphical results are shown in Figs. 6 and 7. It is observed that as spike regularization increases, the average spike rate per neuron decreases following a logarithmic rule that gets close to rates observed in biological neurons as λ_{sr} approaches 1. The amount of

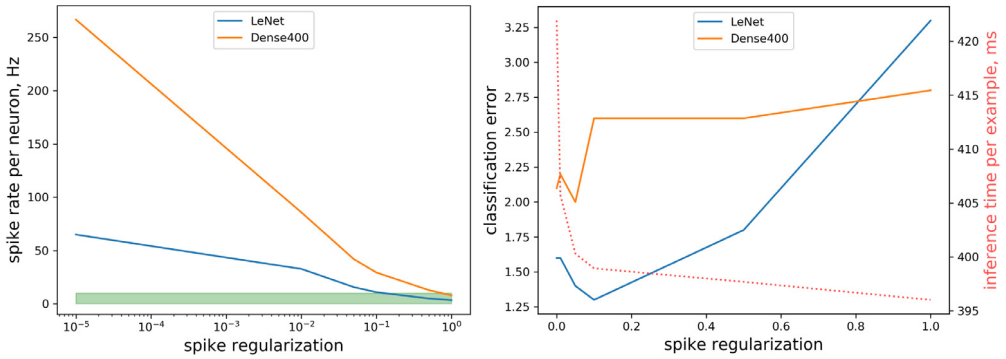

Fig. 6. Effect of spike regularization on both densely connected and convolutional neural network for the NMNIST digit recognition. Left: Effect on spike rate, with spike regularization on log scale for better visualization. The green area indicates the spike rates commonly reported in biological neurons. Right: Effect on classification error and inference time (dotted line).

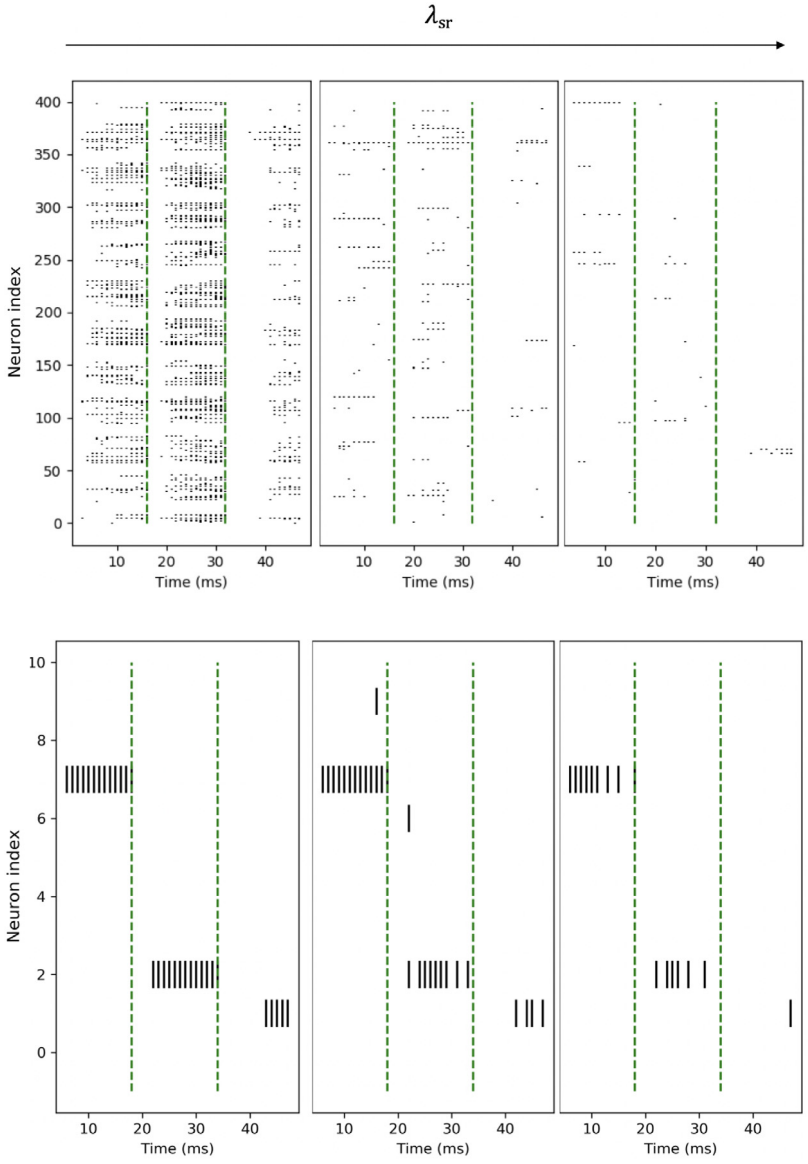

Fig. 7. Raster plots for the Deep Fully Connected simulation of three NMNIST examples: 7,2,1; for three increasing values of λ_{sr}: 0, 0.1 and 1. Each sample was propagated for 16 ms, as indicated by the vertical lines. Top plot is for the first hidden layer. Bottom plot is for the output layer.

elicited spikes is reduced almost 34 times for the fully connected and 19 times for the convolutional network, with a small drop in the classification accuracy: less than 1% and less than 2% for the Dense400 and LeNet respectively.

4. Discussion

This paper constitutes an attempt to consistently port deep Spiking Neural Networks simulations into neuromorphic

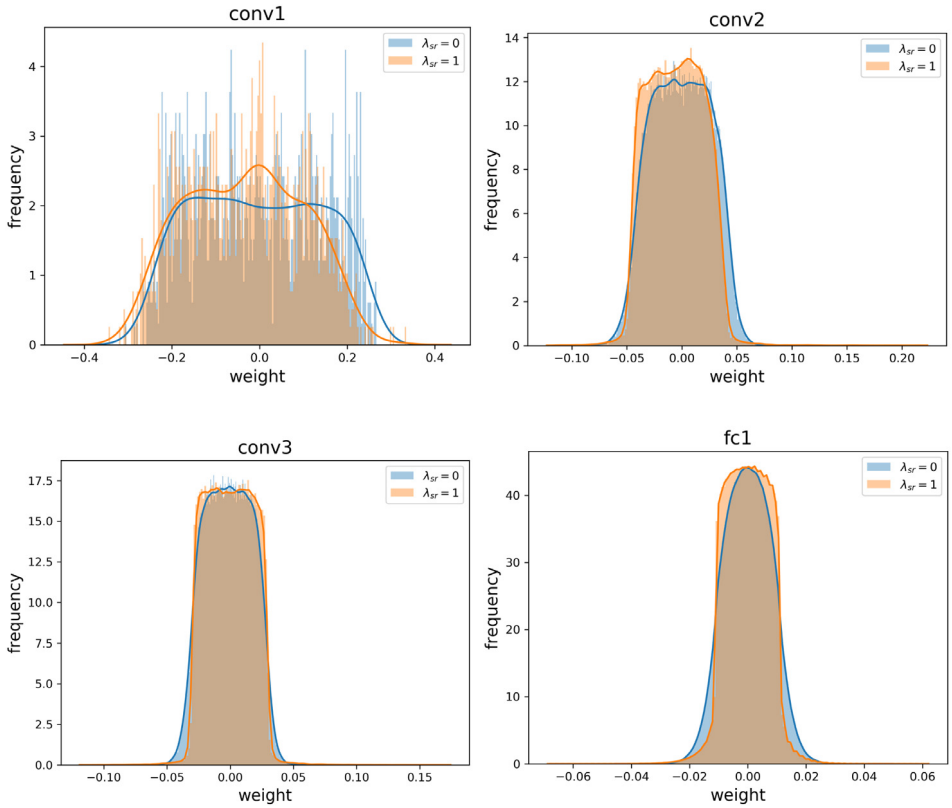

Fig. 8. Weight distribution shift of four layers of the spiking CNN for two extreme values of λ_{sr}. The number of bins for the histogram visualization is fixed to 200.

hardware. Fig. 5 shows that in general the desired spiking activity is preserved in our hardware implementation even if multiple layers are used. Also, the possibility to use SNN versions of commonly used techniques of Deep Learning such as Convolutional layers, Average Pooling, Batch Normalization and Dropout, added to the aforementioned spiking reliability of the hardware implementation in SpiNNaker opens the door to deploying complex deep spiking architectures for image and video classification, natural language processing, robot navigation, etc. Currently, the main limitation seems to be the number of simulated neurons, which in the SpiNNaker 103 board is about 200 thousand, considering 255 neurons per core, the maximum recommended value. For reference, the bigger implementation reported here, the converted CNN, used only 8 thousand neurons.

On the other hand, the use of neuromorphic hardware significantly reduces the time of simulations of biologically realistic DSNNs. Here we show that training DSNN prototypes in e.g. PyTorch and deploying on SpiNNaker using the user-friendly PyNN module is possible in few steps. This can greatly contribute to research on new training algorithms and architectures for efficient machine learning systems deployed on brain-like hardware.

It is important to point out that the ultimate goal of neuromorphic systems for machine learning is to achieve better energy efficiency compared to conventional hardware, rather than perfect accuracy. Following the approach of Cao et al. (2015), we sought to reduce the number of spikes while preserving a high classification accuracy. The proposed modification of the cost function of the STBP algorithm achieves this goal, yielding other interesting results worth further exploration:

- Spike rates similar to biological neurons are achievable with Deep SNNs. Although exact average spike rate of human brain neurons is still a matter of discussion, works like Mizuseki and Buzsáki (2013) and Roxin, Brunel, Hansel, Mongillo, and van Vreeswijk (2011) allow an estimate between 0.1 and 10 Hz for hippocampal and cortical neurons. Our work shows that with enough spike regularization, average spike rates in a DSNN can go below 10 Hz for a digit classification task, being the lower reported average 3.35 Hz.
- Forcing the spike regularization factor to yield lower spike rates than 0.1 Hz leads to a significant drop in accuracy, especially for the spiking CNN. Minimum spike rate achievable for this task is left to future work.
- The best accuracy is achieved with a small spike regularization. By observing the right-hand plot in Fig. 6, for both tested architectures the best accuracy was, surprisingly, with a small regularization factor λ_{sr}, between 0.05 and 0.1. A monotonic increase in the error was expected. One possible explanation for this behavior is that both networks learn the best balance between generalization of the data and expressive power (given by the spike rate) in this range.
- The spikes in Spiking CNNs are more sparse than in their densely connected counterparts, hence the observed lower spike regularization effect on the Spiking LeNet was expected. Nonetheless, the observed spike reduction on Spiking CNNs is not as uniform as in Fully Connected networks, but expressed in the fact that fewer feature maps are allowed to learn. Refer to the provided animations (links in caption of Fig. 9) to observe this effect.
- In Fig. 8, the synaptic weight distribution for different layers of Spiking LeNet is displayed, with and without regularization. As expected, regularization decreases the weights of excitatory (positive) synapses and increases those of inhibitory (negative) synapses. Interestingly, the effect is more visible in the first layer, responsible for processing lower level, faster spatio-temporal features.

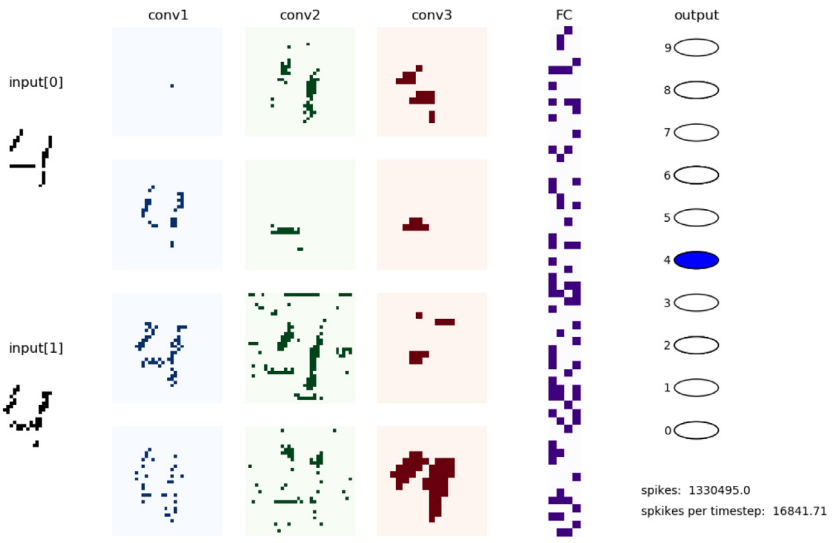

Fig. 9. Animated plots of the firing patterns in both architectures are provided. Spiking LeNet: https://photos.app.goo.gl/h2DytAutZ2pFK5rd8. Dense400: https://photos.app.goo.gl/uMq7dJbA7pmSwYyt9. Fully connected layers are arranged in 2D for better visualization.

Table 5
Effect of spike regularization in the spike rate and accuracy for both densely connected and convolutional spiking neural networks.

λ_{sr}	Spike rate [Hz]		Accuracy	
	LeNet	Dense400	LeNet	Dense400
0.00	64.88	266.57	98.4	97.9
0.01	32.70	85.82	98.4	97.8
0.05	15.63	41.89	98.6	98.0
0.10	10.86	29.28	98.7	97.4
0.50	4.92	12.80	98.2	97.4
1.00	3.35	7.86	96.7	97.2

One limiting factor of using the SpiNNaker platform is that it is difficult to assess its energy consumption. The reported time (seen in Fig. 6) can constitute an indication that processing less spikes do indeed require less energy in this neuromorphic platform, but this time difference is mostly due to the memory footprint left while recording the spikes and membrane potentials.

While an energy efficiency analysis was not in the scope of this paper, we believe that SNNs are naturally more suited for energy-efficient event-based processing than traditional ANNs. In this regard, we consider that the use in this work of a DVS recorded dataset such as NMNIST and the introduction of spike regularization mechanisms in the training phase are steps in the right direction. Spatial event processing in neuromorphic hardware such as that performed in this and other recent works form part of a promising alternative to represent and harness spatio-temporal data: without the need for a global time axis. Traditional temporal processing relies on recording and processing snapshots of the data at a given rate. Instead, we advocate for the use interconnected units that locally react to changes in the stimuli as they occur and are able to keep track of previous states. This way knowledge is represented in the connectivity, internal states and firing activity of such deeply connected units, augmenting the memory capacity and energy efficiency of the systems as background information is not represented. In particular, this work presents a way to extract knowledge from spatio-temporal data with SNNs by penalizing the excessive firing activity observed in previous systems. In future work we aim to deploy more complex architectures and investigate in how to efficiently perform online neuromorphic hardware learning.

Acknowledgments

This research has been supported by the CONACYT, Mexico project FC2016-1961 'Neurociencia Computacional: de la teoría al desarrollo de sistemas neuromórficos'. This work was also partially supported by the EU H2020 grant 824164 "HERMES", and by the Spanish grant TEC2015-63884-C2-1-P "COGNET" (with support from the European Regional Development Fund).

References

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., et al. (2015). Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *34*(10), 1537–1557.

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.

Cao, Y., Chen, Y., & Khosla, D. (2015). Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, *113*(1), 54–66.

Cuevas-Arteaga, B., Dominguez-Morales, J. P., Rostro-Gonzalez, H., Espinal, A., Jimenez-Fernandez, A. F., Gomez-Rodriguez, F., et al. (2017). A spiNNaker application: design, implementation and validation of SCPGs. In *International work-conference on artificial neural networks* (pp. 548–559). Springer.

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro*, *38*(1), 82–99.

Davison, A. P., Brüderle, D., Eppler, J. M., Kremkow, J., Muller, E., Pecevski, D., et al. (2009). Pynn: a common interface for neuronal network simulators. *Frontiers in Neuroinformatics*, *2*, 11.

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S.-C., & Pfeiffer, M. (2015). Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 international joint conference on neural networks (IJCNN)* (pp. 1–8). IEEE.

Esser, S. K., Appuswamy, R., Merolla, P., Arthur, J. V., & Modha, D. S. (2015). Backpropagation for energy-efficient neuromorphic computing. In *Advances in neural information processing systems* (pp. 1117–1125).

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

Garbin, D., Bichler, O., Vianello, E., Rafhay, Q., Gamrat, C., Perniola, L., et al. (2014). Variability-tolerant convolutional neural network for pattern recognition applications based on oxram synapses. In *2014 IEEE international electron devices meeting* (pp. 28–34). IEEE.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

Jin, X., Luján, M., Khan, M. M., Plana, L. A., Rast, A. D., Welbourne, S. R., et al. (2010). Algorithm for mapping multilayer bp networks onto the spinnaker neuromorphic hardware. In *2010 ninth international symposium on parallel and distributed computing* (pp. 9–16). IEEE.

Jorgenson, L. A., Newsome, W. T., Anderson, D. J., Bargmann, C. I., Brown, E. N., Deisseroth, K., et al. (2015). The brain initiative: developing technology to catalyse neuroscience discovery. *Philosophical Transactions of the Royal Society, Series B (Biological Sciences)*, *370*(1668), 20140164.

Kasabov, N. K. (2018). *Timespace, spiking neural networks and brain-inspired artificial intelligence, vol. 7*. Springer.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436.

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Liu, C., Bellec, G., Vogginger, B., Kappel, D., Partzsch, J., Neumärker, F., et al. (2018). Memory-efficient deep learning on a spinnaker 2 prototype. *Frontiers in Neuroscience*, *12*.

Markram, H. (2012). The human brain project. *Scientific American*, *306*(6), 50–55.

Merolla, P., Arthur, J., Akopyan, F., Imam, N., Manohar, R., & Modha, D. S. (2011). A digital neurosynaptic core using embedded crossbar memory with 45pj per spike in 45 nm. In *2011 IEEE custom integrated circuits conference (CICC)* (pp. 1–4). IEEE.

Mizuseki, K., & Buzsáki, G. (2013). Preconfigured, skewed distribution of firing rates in the hippocampus and entorhinal cortex. *Cell Reports*, *4*(5), 1010–1021.

Moore, G. E. (1998). Cramming more components onto integrated circuits. *Proceedings of the IEEE*, *86*(1), 82–85.

Neftci, E., Das, S., Pedroni, B., Kreutz-Delgado, K., & Cauwenberghs, G. (2014). Event-driven contrastive divergence for spiking neuromorphic systems. *Frontiers in Neuroscience*, *7*, 272.

Neil, D., & Liu, S.-C. (2014). Minitaur, An event-driven fpga-based spiking network accelerator. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *22*(12), 2621–2628.

O'Connor, P., Neil, D., Liu, S.-C., Delbruck, T., & Pfeiffer, M. (2013). Real-time classification and sensor fusion with a spiking deep belief network. *Frontiers in Neuroscience*, *7*, 178.

Orchard, G., Jayawant, A., Cohen, G. K., & Thakor, N. (2015). Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience*, *9*, 437.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., et al. (2017). Automatic differentiation in PyTorch. In *NIPS autodiff workshop*.

Ponulak, F., & Kasiński, A. (2010). Supervised learning in spiking neural networks with resume: sequence learning, classification, and spike shifting. *Neural Computation*, *22*(2), 467–510.

Rhodes, O., Bogdan, P. A., Brenninkmeijer, C., Davidson, S., Fellows, D., Gait, A., et al. (2018). Spynnaker: a software package for running pynn simulations on spinnaker. *Frontiers in Neuroscience*, *12*.

Roxin, A., Brunel, N., Hansel, D., Mongillo, G., & van Vreeswijk, C. (2011). On the distribution of firing rates in networks of cortical neurons. *Journal of Neuroscience*, *31*(45), 16217–16226.

Rueckauer, B., Lungu, I.-A., Hu, Y., & Pfeiffer, M. (2016). Theory and tools for the conversion of analog to spiking convolutional neural networks. arXiv preprint arXiv:1612.04052.

Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M., & Liu, S.-C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, *11*, 682.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning internal representations by error propagation*: *Tech. rep.*, California Univ San Diego La Jolla Inst for Cognitive Science.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, *61*, 85–117.

Schmitt, S., Klähn, J., Bellec, G., Grübl, A., Guettler, M., Hartel, A., et al. (2017). Neuromorphic hardware in the loop: Training a deep spiking network on the brainscales wafer-scale system. In *2017 international joint conference on neural networks (IJCNN)* (pp. 2227–2234). IEEE.

Serrano-Gotarredona, T., Linares-Barranco, B., Galluppi, F., Plana, L., & Furber, S. (2015). Convnets experiments on spinnaker. In *2015 IEEE international symposium on circuits and systems (ISCAS)* (pp. 2405–2408). IEEE.

Shrestha, A., Ahmed, K., Wang, Y., Widemann, D. P., Moody, A. T., Van Essen, B. C., et al. (2017). A spike-based long short-term memory on a neurosynaptic processor. In *2017 IEEE/ACM international conference on computer-aided design (ICCAD)* (pp. 631–637). IEEE.

Shrestha, S. B., & Orchard, G. (2018). Slayer: Spike layer error reassignment in time. In *Advances in neural information processing systems* (pp. 1419–1428).

Stromatias, E., Neil, D., Pfeiffer, M., Galluppi, F., Furber, S. B., & Liu, S.-C. (2015). Robustness of spiking deep belief networks to noise and reduced bit precision of neuro-inspired hardware platforms. *Frontiers in Neuroscience*, *9*, 222.

Wu, Y., Deng, L., Li, G., Zhu, J., & Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in Neuroscience*, *12*.