Neuromorphic Accelerators: A Comparison Between Neuroscience and Machine-Learning Approaches

Zidong Du[†‡] **Daniel D Ben-Dayan Rubin**[‡] **Yunji Chen**[†] **Liqiang He**[¶]
Tianshi Chen[†] **Lei Zhang**[†‡] **Chengyong Wu**[†] **Olivier Temam**[§]
[†]State Key Laboratory of Computer Architecture, Institute of Computing Technology (ICT), CAS, China
[‡]University of CAS, China [‡]Intel, Israel [¶]College of Computer Science, Inner Mongolia Univ., China [§]Inria, France
{duzidong,cyj,chentianshi,cwu}@ict.ac.cn daniel.ben-dayan.rubin@intel.com
liqiang.he@gmail.com zhanglei515@mails.ucas.ac.cn olivier.temam@inria.fr

ABSTRACT

A vast array of devices, ranging from industrial robots to self-driven cars or smartphones, require increasingly sophisticated processing of real-world input data (image, voice, radio, ...). Interestingly, hardware neural network accelerators are emerging again as attractive candidate architectures for such tasks. The neural network algorithms considered come from two, largely separate, domains: machine-learning and neuroscience. These neural networks have very different characteristics, so it is unclear which approach should be favored for hardware implementation. Yet, few studies compare them from a hardware perspective. We implement both types of networks down to the layout, and we compare the relative merit of each approach in terms of energy, speed, area cost, accuracy and functionality.

Within the limit of our study (current SNN and machine-learning NN algorithms, current best effort at hardware implementation efforts, and workloads used in this study), our analysis helps dispel the notion that hardware neural network accelerators inspired from neuroscience, such as SNN+STDP, are currently a competitive alternative to hardware neural networks accelerators inspired from machine-learning, such as MLP+BP: not only in terms of accuracy, but also in terms of hardware cost for realistic implementations, which is less expected. However, we also outline that SNN+STDP carry potential for reduced hardware cost compared to machine-learning networks at very large scales, if accuracy issues can be controlled (or for applications where they are less important). We also identify the key sources of inaccuracy of SNN+STDP which are less related to the loss of information due to spike coding than to the nature of the STDP learning algorithm. Finally, we outline that for the category of applications which require permanent online learning and moderate accuracy, SNN+STDP hardware accelerators could be a very cost-efficient solution.

MICRO-48, December 05-09, 2015, Waikiki, HI USA
© 2015 ACM. ISBN 978-1-4503-4034-2/15/12$15.00
DOI: http://dx.doi.org/10.1145/2830772.2830789.

Categories and Subject Descriptors

C.1.3 [**PROCESSOR ARCHITECTURES**]: Other Architecture Styles

Keywords

Neuromorphic, Accelerator, Comparison

1. INTRODUCTION

Due to stringent energy constraints, both embedded and high-performance systems are turning to accelerators, i.e., custom circuits capable of efficiently implementing a range of tasks, at the cost of less flexibility than processors. While FPGAs or GPUs are popular forms of accelerators, Kuon et al. [1] still report an energy ratio of 12x on average between FPGAs and ASICs, and GPUs are even worse [2]. At the same time, single-algorithm ASICs are not an option for most heterogeneous multi-cores or SoCs except for a few specific but frequently used algorithms, such as video decoding. Finding accelerators which can achieve high energy efficiency yet retain a broad application scope is a significant challenge.

In that context, *machine-learning* is a particularly interesting application domain: a very broad set of applications rely on machine-learning algorithms, especially on the client for processing real-world inputs such as visual and audio information, e.g., typically image and audio/voice recognition tasks. And interestingly, at the same time, a few key algorithms, especially *Deep Neural Networks* (DNNs), have been shown to be state-of-the-art across a broad range of applications [3, 4]. This, in turn, provides a unique opportunity to design accelerators which can be very efficient, because they only need to implement a few variations of the same algorithm, yet achieve a very broad application scope. A number of researchers have identified that recent trend and proposed hardware neural network accelerators [5, 6]. However, DNNs are far too large to be fully mapped in hardware with tens of millions of synapses and hundreds of thousands of neurons [4]. So the most recent research works propose only to implement a few of the neurons of Convolutional Neural Networks (CNNs, a variant of DNNs) [5] or a full but small-scale Multi-Layer Perceptron (MLP) [6].

However, next to this trend, another type of hardware neural networks is emerging: those inspired by *neuroscience*. Unlike machine-learning neural networks which only have a remote relationship with biological neurons, these neural networks are more directly inspired from neuroscience models [7]. The primary goal of these hardware neural networks is the emulation of large-scale biological neural networks, though, increasingly, they are also considered for application purposes. Two of the most prominent efforts are driven by companies, i.e., TrueNorth [8] by IBM and the recent Zeroth processor [9] by Qualcomm, so it is safe to assume that such efforts will ultimately target commercial applications. Such neuroscience-inspired models have several significant assets: (1) they have been successfully applied to complex machine-learning tasks, such as face detection or handwritten digit recognition [10, 11], so they should not be ruled out as potential candidate accelerators, (2) the fact the information is coded using spikes (Spiking Neural Networks or SNNs) can help achieve significant area and energy savings [12], (3) the bio-inspired learning rule called Spike-Timing Dependent Plasticity (STDP) which allows to learn continuously while the SNN is being used, providing high adaptivity.

As a result, it is no longer clear which type of model is the most compatible with the area, speed, energy, accuracy and functionality requirements of embedded systems. Unfortunately, since the two types of models are investigated by two fairly different communities, i.e., machine-learning and neuroscience, there are very few *quantitative* comparisons of their accuracy and even less so of their hardware cost. In order to help fill this gap, we make a quantitative comparison between two of the most well-known models of each type: the Multi-Layer Perceptron with Back-Propagation (BP) [13] and the Spiking Neural Network with Spike-Timing Dependent Plasticity learning [10]. Using the MNIST benchmark for handwritten digit recognition [13] as our driving example, we make a best effort at exploring their design spaces,[1] and implementing them in hardware, down to the layout at 65nm,[2] in order to evaluate their area, energy, speed and accuracy. We also validate our conclusions on two other benchmarks respectively tackling image recognition and speech processing: the MPEG-7 benchmark for object recognition [14], and the Spoken Arabic Digits for speech recognition [15].

We attempt to provide some answers to the following questions about hardware neural network accelerators:

1. Can neural networks inspired from neuroscience (such as SNN+STDP) perform as well as (or better than) neural networks inspired from machine-learning (such as MLP+BP) on recognition tasks?
2. Is the cost of hardware SNN significantly lower than the cost of hardware MLP?
3. In which cases shall the designer consider using hardware SNN or hardware MLP accelerators?

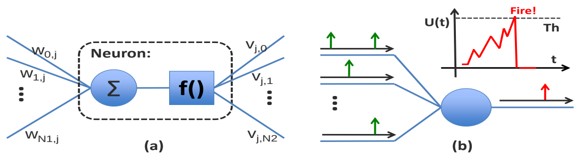

Figure 1: (a) MLP vs. (b) SNN.

Since both SNN and machine-learning NN models evolve constantly, as well as the options for implementing them in hardware, we cannot provide definite answers to these questions. However, for current models, using best efforts at hardware implementations and current technologies, and using a limited set of workloads (especially MNIST, one of the few workloads used in both SNN and machine-learning NN studies), we can provide a snapshot of what the current balance between the two approaches is.

Within these restrictions of the scope of our study, we provide the following conclusions:
(1) When considering the exact same recognition task (e.g., MNIST handwritten digit recognition), the accuracy of models inspired from neuroscience, SNN+STDP, is significantly lower than even simple models inspired from machine-learning, such as MLP+BP. We also find that the cause of the accuracy gap is non-trivial, less related to spike coding than to the nature of the STDP learning algorithm; we were able to fully bridge the accuracy gap between SNN+STDP and MLP+BP.
(2) While hardware SNNs are significantly cheaper (area and power) than MLPs when they are fully expanded in hardware (each logical neuron and synapse mapped to a hardware neuron and a hardware synapse respectively), MLPs are actually cheaper than SNNs when reasonable area costs constraints, compatible with embedded system design, are considered. We could reduce the gap between the two models, especially energy-wise, by using a simplified SNN model, at a moderate loss of accuracy; however, the MLP model remains significantly more area/energy efficient than the SNN one.
(3) Even though the range of applications which require permanent online learning (as opposed to offline learning) may be limited, we observe that the hardware overhead (area and power) of implementing STDP is very low. So applications requiring permanent online learning and tolerant to moderate accuracy are excellent candidates for SNN+STDP accelerators. SNN+STDP should also be the design of choice for fast and large-scale implementations (spatially expanded), especially if accuracy issues can be mitigated by changing the learning algorithm as explored in this article.

In Section 2 we present the two target models (MLP+BP and SNN+STDP), in Section 3 we compare their ac-

[1] We have explored the topologies and operator types for both MLP and SNN models. For SNN, we have additionally explored different coding schemes (including four rate coding and two temporal coding schemes, see Section 5) as well as neuron parameters. we also make a best effort at reimplementing an IBM TrueNorth core [8] in our exploration of hardware SNNs, see Section 5.

[2] In accordance with prior art [16, 11, 8], we focus the hardware implementation on the feed-forward (testing or inference) path of neural networks as most applications are compatible with offline learning [6, 16]. However, we also investigate the hardware cost of implementing STDP as it allows for online learning, unlike BP.

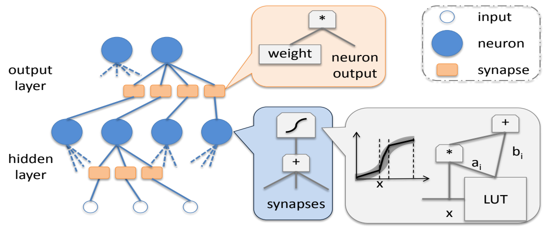

Figure 2: Operators within a 2-layer MLP.

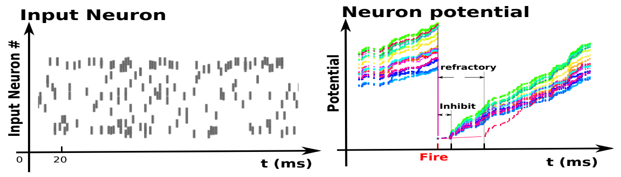

Figure 3: Spike coding in SNN: *(left) all spikes of 300 neurons (one neuron per line) for one MNIST presentation image, (right) the potential of the different neurons increases as they receive input spikes (one color line per neuron), until one fires; the firing neuron enters a refractory period for 20 ms, and the other neurons are inhibited for 5 ms by the firing neuron; note that all neurons have different firing thresholds due to homeostasis.*

curacy and attempt to bridge them, in Section 4 we consider two variants of hardware implementation (spatially expanded at any cost, and cost-constrained spatially folded), including a simplified (and more energy-efficient) variant of SNNs, and we also explore the hardware cost of implementing online learning (STDP); in Section 6 we present the related work.

2. MODELS

In this section, we briefly recap how both models (MLP+BP and SNN+STDP) operate as an introduction to their hardware implementation in Section 4, both in feed-forward mode, and in learning mode, then we compare their accuracy on the MNIST benchmark.

2.1 MLP

Topology. MLPs contain input layer, one or multiple hidden layers, and an output layer; the input layer does not contain neurons, and the inputs are usually n-bit values (8-bit values in our case for the pixel luminance). The output neurons are connected to all neurons in the hidden layer, and all neurons in the hidden layer are connected to all inputs.

Neurons and synapses. A neuron j in layer l performs the following computations: $y_j^l = f\left(s_j^l\right)$ where $s_j^l = \sum_{i=0}^{N_{l-1}} w_{ji}^l y_i^{l-1}$, w_{ji} is the synaptic weight between neuron i in layer $l-1$ and neuron j in layer l, N_l is the number of neurons in layer l, and f is the activation function; we used the typical sigmoid function $f(x) = \frac{1}{1+e^{-x}}$.

Learning (Back-Propagation). Back-Propagation is a supervised learning method where an input is first presented to the network which produces an output using the feed-forward path. The error between this network output and the known output is then propagated back through the network to update the weights. The method is iterative: this process is repeated multiple times until the target error is achieved or the allocated learning time has elapsed. The weights are updated as follows: $w_{ji}^l(t+1) = w_{ji}^l(t) + \eta\delta_j^l(t)y_i^{l-1}(t)$, where t is the learning iteration, η is the learning rate, and δ_j^l is the error gradient, i.e., the direction of the error. At the output layer, the gradient expression is $\delta_j^l(t) = f'(s_j^l(t)) \times e_j^l(t)$, where e_j^l is the error (the difference between the network output and the expected output), and in the hidden layer, the gradient expression is $\delta_j^l(t) = f'(s_j^l(t)) \times \sum_{k=0}^{N_{l+1}} \delta_k^{l+1}(t)w_{kj}(t)$, where f' is the derivative of f.

2.2 SNN

Topology. The SNN layer is not only connected to all its inputs through excitatory connections, but all neurons are connected among themselves using inhibitory connections. We consider a single-layer SNN. While this choice may initially come across as unfair with respect to MLPs, the best reported results so far on MNIST using SNNs have been achieved with a single layer [11]. In fact, the lateral inhibitory connections produce the so-called winner-takes-all (WTA) dynamics and create a form of recurrent network. This type of connectivity effectively ensures both compactness and, theoretically, a number of non-linear mappings of the input data, leading to competitive recognition accuracy compared to multi-layer networks [17].

Neurons and synapses. The inputs and outputs of the neuron are *spikes*, see Figure 3. Each of the neuron output can be reinterpreted as a spike train and the spike frequency converted by counting the number of spikes over a fixed time interval [12]. However, a more efficient readout method is to consider the first neuron which spikes as the winner, i.e., a form of spike-based winner-takes-all. This approach lends well to both a fast and dense hardware implementation and it has achieved some of the best machine-learning capabilities so far with SNN [10].

We consider the standard Leaky Integrated and Fire (LIF) neuron model [18], where the neuron potential v_j of neuron j is the solution of the following differential equation:

$$\dot{v}_j(t) + \frac{v_j(t)}{T_{leak}} = \sum_{i=1}^{n} w_{ji} \times I_i(t)$$

where T_{leak} is the leakage time constant, and I_i is the value of input i. Normally, the solution is obtained via discrete simulation (multiple time steps), which, in hardware, would translate into repeated computations at short time intervals. This would be inefficient from both a time and energy perspective. We take advantage of the fact that the potential $v_j(t)$ is the solution of the following differential equation between two input spikes: $\dot{v}_j(t) + \frac{v_j(t)}{T_{leak}} = 0$. Unlike the aforementioned differential question, it is possible to derive an analytical solution of this equation and deduce that the potential, between two consecutive spikes occurring at time T_1 and T_2, is given by the following expression: $v_j(T_2) = v_j(T_1) \times e^{-\frac{T_2-T_1}{T_{leak}}}$. Such an expression lends to a more efficient hardware implementation.

Finally, note that when a neuron fires, it inhibits all other neurons, emulating the presence of inhibitory connections, in line with the practice of other efficient SNN

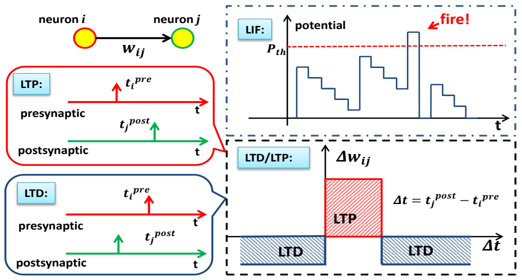

Figure 4: LTP (Potentiation) and LTD (Depression).

models [10]; it also enters a refractory period where incoming spikes have no impact.

Learning (STDP). The learning principles of STDP are very different from BP: the learning is *unsupervised*, each neuron progressively converging to one of the most salient information. The learning principle is to detect causality between input and output spikes: if a neuron fires soon after receiving an input spike from a given synapse, it suggests that synapse plays a role in the firing, and thus it should be reinforced; this notion is called Long-Term Potentiation (LTP), see Figure 4. Conversely, if a neuron fires a long time after receiving an input spike, or shortly before receiving it, the corresponding synapse has most likely no impact, and it should thus be decreased; this is called Long-Term Depression (LTD), see Figure 4. Note that, in our case, STDP is only applied to the input excitatory connections, not the lateral inhibitory connections.

Homeostasis. Finally, in order to balance information among neurons, it is critical to adjust their firing threshold, by punishing neurons which fire too frequently (by increasing their firing threshold), and promoting neurons which fire infrequently (by conversely decreasing their firing threshold). Such dynamic adjustment exists in biological neurons through a process called homeostasis [19], which we have also implemented. While this ensures that all output neurons are learning to be specialized, we observed that it improves the SNN accuracy around 5%. Overall, only one neuron can fire for a given input image, making the readout both trivial and fast.

Homeostasis works on a large time scale compared to spike events; we define a *homeostasis epoch*, and we update all neurons thresholds at the end of each such epoch. The update is done by increasing (respectively decreasing) the neuron threshold if it has fired more (respectively less) than a pre-set homeostasis threshold, according to the following expression: $firing\_threshold += sign(activity - homeostasis\_threshold) * firing\_threshold * r$, where *activity* is the number of times the neuron has fired during the homeostasis epoch, and r is a multiplicative positive constant [20]. Note that the whole process is entirely local to each neuron (so the wiring overhead at the hardware level will be small), except for the counter for signaling the end of a homeostasis epoch.

Labeling. Because STDP is an unsupervised learning method, there is a need for a complementary step to label the neurons with the corresponding output information. We use a self-labeling method which proceeds as follows. We use the training images, of which the labels are known, to label the neurons. Each neuron has

Table 1: MLP and SNN characteristics.

MLP Para.	Range	Our choice	Description
# N_{hidden}	10-1000	100	hidden neurons
# N_{output}	10	10	output neurons
η	0.1-1	0.3	Learning rate
# epochs	10-200	50	Training epochs
SNN Para.	Range	Our choice	Description
# N	10-800	300	Single layer, neurons
T_{period}	100-800	500ms	Image presentation duration
T_{leak}	10-800	500ms	Leakage time constant
$T_{inhibit}$	1-20	5ms	Inhibitory period
T_{refrac}	5-50	20ms	Refractory period
T_{LTP}	1-50	45ms	LTP threshold
T_{init}	w_{max}*70	17850	Initial firing threshold
$Homeo_T$	$10 * T_{period} * (\#N)$	1,500,000	Homeostasis epoch in ms
$Homeo_{th}$	$\frac{3 * Homeo_T}{T_{period} * (\#N)}$	30	Homeostasis threshold

as many counters as the different possible labels; when an output neuron fires for a given input image label, the corresponding neuron label counter is incremented. After all training images have been processed, each neuron gets tagged with the label with the highest score (the score is deduced from the label counter value by dividing by the number of input images with that label; this accounts for possible discrepancies in the number of times each label is used as input).

3. ACCURACY COMPARISON

3.1 SNN+STDP vs. MLP+BP

In order to compare the accuracy of both models, we use the well-known MNIST [13] machine-learning benchmark corresponding to handwritten digits. The winning neural network of the ImageNet 2012 competition (a CNN) achieves an accuracy of 99.21% on the MNIST benchmark using 60 million synapses and 650,000 neurons [4], well beyond reasonable hardware capacity, and the best reported results (99.77%) have been achieved with a multi-column DNN [21].

We trained a much smaller MLP (100 neurons in hidden layer, see Table 1) on this problem and we achieved an accuracy of 97.65%, slightly lower than the best reported results for MLPs [22] (98.4% using 800 hidden neurons). We selected 100 hidden neurons after exploring the number of hidden neurons from to 10 to 1000 (and simultaneously exploring the hyper-parameters, such as the learning rate), and we concluded that beyond 100 hidden neurons, the accuracy gains become marginal with respect to the area overhead. We used the full 60,000 non-distorted MNIST images, and the full 10,000 testing images.

We then trained the SNN as follows. Each pixel luminance is converted into a Poisson spike train of rate proportional to the pixel luminance, for example, a maximum luminance of 255 corresponds to a mean period of 50ms (20Hz). The λ value of the Poisson process corresponds to the following expression: $\frac{1}{3*U-2*\frac{p(i,j)}{255}}$, where U is a pre-defined constant indicating the maximum spike frequency and $p(i,j)$ is the luminance value of input pixel. We did a fine-grained exploration to decide the rest parameters of SNN, including #neurons, duration of image presentation, leakage time constant, etc (see Table 1 for the final parameter setting selected

Table 2: Best accuracy reported on MNIST (no distortion).

Type	Accuracy (%)
MLP+BP [22]	98.40
SNN+STDP [11]	93.50
SNN+STDP [23]	95.00
ImageNet [4]	99.21
MCDNN [21]	99.77

Table 3: Accuracy of MLP and SNN on MNIST.

Type	Accuracy (%)
SNN+STDP - LIF (SNNwt)	91.82
SNN+STDP - Simplified (SNNwot)	90.85
SNN+BP	95.40
MLP+BP	97.65

out of 1000 evaluated settings). Note that, even though SNN is a model inspired from neuroscience, our main goal here is computing efficiency and accuracy. So when it comes to calibrating the model hyper-parameters, we do not impose ourselves to strictly abide by values compatible with neuroscience observations, we simply select the values yielding the best results, e.g., our empirical exploration showed that the best accuracy was achieved with a leakage time constant of 500ms, while that constant is known to be around 50ms in the neuroscience literature.

We report the accuracy results in Table 2 and Table 3. We can observe that the SNN+STDP accuracy is 5.83% less than for the MLP. This result is within 1.68% of the best reported result using SNN+STDP and that number of neurons for MNIST (93.50% by Querlioz et al. [11]). Note that we attempted to exactly match that result by carefully reproducing their approach, and even interacted with the authors, but since we could not access their software, it was impossible to fully bridge this gap. Anyway, whichever is the effective best possible result using SNN+STDP on MNIST (our 91.82% or their 93.50%), it is 5.83% to 4.15% away from the MLP result. While this difference is significant for certain applications, e.g., life or death decisions for radiography analysis of cancer patients, it is acceptable for benign applications, e.g., identifying surrounding objects using a smartphone app such as Google Goggles. In a nutshell, SNN+STDP performs significantly less well than MLP+BP, but its accuracy remains high enough for many applications. Note that Diehl et al. [23] reports a better MNIST accuracy of 95% using an SNN model similar to [11], albeit with 6400 neurons (and 82.9% with 400 neurons vs. 91.82% with 300 neurons in our case).

3.2 Bridging the gap between SNN+STDP and MLP+BP

SNN+BP. We want to further analyze the source of (and as much as possible bridge) the accuracy discrepancy between SNN+STDP and MLP+BP. Here, we focus on the learning algorithms which differ in two major ways. First, STDP is an unsupervised learning algorithm while BP is a supervised learning algorithm. The principle behind STDP [24] relies solely on the timing of the occurring spikes in the pre- and post-synaptic neurons, i.e., local information, there is no notion of a global error signal as in BP; there is also no notion of back-propagation of error along the input-output pathway. Second, STDP is an online algorithm, while BP is an offline algorithm. Online learning rules like STDP raise the problem of retention of earlier memories when new ones are presented. In single-neuron models, the learning rule was shown to have a strong effect on the memory retention time [25]. In particular, it was shown that sufficient lateral inhibition stabilizes receptive fields, the stability of which is a measure of memory retention time span [25].

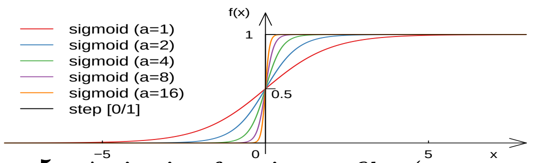

Figure 5: Activation function profiles (parameterized sigmoid and step function).

In order to assess the impact of the learning algorithm, we train the SNN using back-propagation. We proceed as follows: in the feed-forward mode, we use the SNN exactly as before (spikes, leakage, threshold for firing, etc), but after each image presentation, we compute the output error, and propagate it to the synaptic weights using the Back-Propagation algorithm (gradient descent and weights updates). We report the results in Table 3, and we can see that the accuracy has jumped from 91.82% to 95.40%, i.e., only 2.25% of accuracy difference between SNN+BP and MLP+BP. This highlights that spike coding only carries a small responsibility in the loss of accuracy, most of it is due to the learning algorithm (STDP).

Threshold function vs. Sigmoid. We try to further bridge the gap between SNN+BP and MLP+BP. At this point, the main difference between the two models lays in spike coding and in the activation function used. It is as if SNNs use a threshold function which is a simple [0/1] step function (no spike/spike), while MLPs use a sigmoid, see Figure 5. By parameterizing the sigmoid function $f_a(x) = \frac{1}{1+e^{-a\times x}}$, instead of $f(x) = \frac{1}{1+e^{-x}}$, it is possible to gradually alter the profile of the sigmoid in order to bring it closer to the profile of a step function; a is a slope parameter, and the higher a, the closer to a step function. see Figure 5. We train and test the MLP+BP for different values of a, and we show the corresponding error in Figure 6. One can see that when a increases, the error becomes increasingly close to that of the step function, closing the gap between SNN+BP and MLP+BP.

Overall, we observe that spike coding is only partially at fault in the accuracy discrepancy between SNN+STDP and MLP+BP: the nature of the learning algorithm (STDP) has far more impact, and the only spike-related aspect is the use of a threshold function to generate spikes, instead of the sigmoid function used by MLPs. While our hardware implementation of SNN described thereafter uses a step function, this analysis suggests a research direction for further bridging the accuracy gap between SNNs and MLPs.

4. HARDWARE COST COMPARISON

In this section, we investigate the design tradeoffs for implementing MLPs and SNNs in hardware. As explained in the introduction, we focus the hardware implementation on the feed-forward path because most applications are compatible with offline learning. We

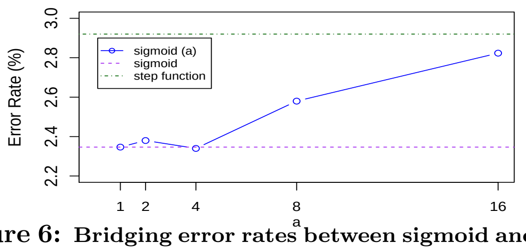

Figure 6: Bridging error rates between sigmoid and step functions.

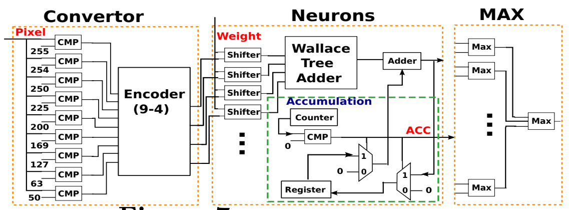

Figure 7: Hardware SNN.

distinguish two types of design: the most straightforward, but also costliest, approach where the hardware design corresponds to the conceptual representation of the NN, and where all components (neurons, synapses) are mapped to *individual* hardware components, i.e., it is *spatially expanded* in hardware. Due to the cost of such designs, they can only correspond to small-scale NNs and thus they are not appropriate for meaningful applications. We then explore *spatially folded* designs where hardware components (especially neuron inputs) are time-shared by several logical components of the original NN. For the sake of fairness, we try to make similar choices for the implementation of both MLPs and SNNs.

4.1 Methodology

We have implemented the different circuits described in this article at the RTL level, synthesized them and placed and routed them; in some cases (fully expanded version), we estimate the area of the circuits based on the placed and routed versions of individual neurons, when the circuits are too large. The synthesis was done using the Synopsys Design Compiler (TSMC 65nm GPlus high VT library), and the layout using the Synopsys IC compiler. With the layout version, we simulated the design using Synopsys VCS and PrimeTime PX to get accurate power and energy numbers. All reported area, delay, power and energy numbers are obtained with these tools.

For the accuracy comparisons, we implemented a C++ version of both the SNN+STDP and MLP+BP due to the long training times, incompatible with RTL-level simulations. We validated both simulators against their RTL counterpart.

4.2 Spatially Expanded

4.2.1 *MLP.*

For MLPs, the spatially expanded hardware design is similar to the conceptual representation of Figure 2. The hardware neuron is implemented through a set of multipliers, one per input synapse, followed by an adder tree to cumulate all such products. The sigmoid is implemented using 16-point piecewise linear interpolation, requiring only a small SRAM table to store the interpolation points (2 coefficients per point), an adder and a multiplier [6], i.e., the interpolated function f value at point x in segment i is $f(x) = a_i \times x + b_i$. We evaluated the optimal size of the operators and synaptic storage (through repeated training/testing experiments using the MNIST benchmark), and we found that the results achieved with 8-bit fixed-point operators (multipliers, adders, SRAM width) were on par with the ones obtained with floating-point operators: respectively 96.65% vs. 97.65%.

4.2.2 *SNN.*

Spike *timing* has a fundamental impact on the bio-realistic STDP learning strategy typically used with SNNs [26]. However, in the feed-forward path, it is less obvious that spike timing information is as critical. Being able to ignore the timing information would bring significant speedups by avoiding the need to carefully model timing information and treating spikes simultaneously.

Therefore, we start by evaluating the importance and impact of timing information in the forward path. We consider two SNN versions: one is a traditional Leaky Integrated and Fire model, as presented in Section 2.2 (which we call SNNwt for "SNN with time") and the second one is a simplified model where all timing information has been removed (which we call SNNwot for "SNN without time"). For SNNwot, each pixel is converted into a set of spikes, along the same principles as for the LIF model, except only the number of spikes is obtained, not the time between spikes; similarly, the role of the leak is ignored. We report the corresponding accuracy results in Table 3. We observe that the accuracy difference between the two is 1.03%, which is not negligible, but small enough that it can be accepted provided it comes with significant speed benefits. We now consider the hardware implementation of each version.

SNN without Timing Information (SNNwot).

Spike generation. We first need to convert pixel information (here, 8-bit greyscale) into spikes; note that some sensors can directly generate spikes [27], but we make the conservative assumption that traditional CMOS sensors are used.

Ignoring the time interval between spikes allows to significantly reduce the hardware complexity and improve speed: we only need to count the number of spikes generated by a given pixel value. Still, we must generate the same number of spikes as for the STDP learning process in order to obtain consistent forward-phase results. The pixel-to-spikes conversion is the following: an 8-bit pixel can generate up to 10 spikes (the delay between image presentation is 500ms and the minimum spike interval is 50ms), though this number of spikes is directly generated as a 4-bit value, see Figure 7, instead of the sequence of unary spikes (spike train) used by STDP.

Neuron. The neuron potential increases with each spike, by an amount equal to the synaptic weight on the corresponding neuron input, see Figure 1. For a given input with synaptic weight W, the accumulation of weights

Table 4: Spatially expanded SNN vs. MLP.

Network	Operator	Area /operator (μm^2)	# operators	Total cost /operator (mm^2)	Total cost /wo SRAM (mm^2)	SRAM (mm^2)	Total cost (mm^2)
SNNwot (28x28-300)	adder tree	89006	300	26.70	26.79	19.27	46.06
	max	6081	16	0.10			
SNNwt (28x28-300)	adder tree	60820	300	18.25	19.62	19.27	38.89
	rand	1749	784	1.37			
MLP (28x28-100-10)	adder tree	45436	100	4.54	73.14	6.49	79.63
	adder tree	5657	10	0.06			
	multiplier	862	79510	68.54			
MLP (28x28-15-10)	adder tree	45436	15	0.68	10.98	1.35	12.33
	adder tree	1131	10	0.01			
	multiplier	862	11935	10.29			

is equal to $N \times W$ where N is the number of spikes. Due to the limited value of N ($N \leq 10$, i.e., less than 10 spikes), an efficient hardware implementation of the multiplier consists of 4 shifters and 4 adders, to compute $n_3 * 2^3 * W + n_2 * 2^2 * W + n_1 * 2^1 * W + n_0 * 2^0 * W$ where $N = n_3 n_2 n_1 n_0$, see Figure 7. The results from all inputs are accumulated through a Wallace tree adder.

Readout. After reaching a threshold potential value, a neuron will fire an output spike. The most typical readout method for SNNs is to count the number of output spikes and the one fires most wins. An alternative is to detect the first neuron which spikes [28]. Both methods rely on timing information which have been voluntarily removed for hardware efficiency purposes. However, the *neuron potential* is highly correlated to the number of output spikes, so we determine the winning neuron as the one with the highest potential.

Overall, the hardware implementation is a 3-stage pipeline, one for the spike generation, one for the adder tree, and one for the max operation of the readout.

SNN with Timing Information (SNNwt).

In order to use the spike timing information, we must randomly generate the number of spikes and the time between spikes for each pixel value, as explained in Section 3.1, see Figure 3. In order to both speed up the execution and minimize the number of random number generators per neuron, we only generate the spike time intervals, and since a fixed amount of time is assigned to each image presentation (500 ms in our case, emulating one millisecond with one clock cycle), we simply stop the spike generation when that threshold is reached; so we do not generate the number of spikes per se, it is done implicitly.

We normally need a Poisson random number generator, see Section 3.1, however, such a generator is usually costly [29, 30]. We have experimented with various alternative methods for generating the spike timings, and we have observed that the accuracy does not change noticeably with a Gaussian instead of a Poisson distribution, even though it is slightly less bio-realistic. Fortunately, a Gaussian pseudo-random number generator can be efficiently implemented [31] using the central limit theorem. The principle is to sum random uniform numbers generated from four Linear Feedback Shift Registers (LFSRs). Using 31-bit as the length and $x^{31} + x^3 + 1$ as the primitive polynomial avoids obtaining cycling over numbers

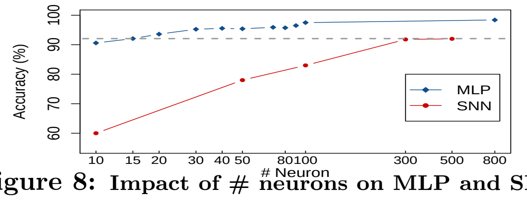

Figure 8: Impact of # neurons on MLP and SNN.

using LFSRs. A single Gaussian random number generator costs 1,749 μm^2, and we use 784 such generators (as many as the number of inputs). These generators provide the time between spikes in milliseconds (with one clock cycle modeling one millisecond), the counters are decremented every cycle, and a new spike is generated when the counter reaches 0.

4.2.3 Comparison.

We now compare the area of the spatially expanded version of both networks. Each network has 28x28 8-bit greyscale inputs corresponding to the MNIST images. For each network, we adopted the structure most commonly found in the literature: 2 layers for MLP (one hidden layer) [13], 1 large layer for SNNs albeit with lateral inhibition [32]. We then explored the configuration of both networks to find the minimal number of neurons in each case for the MNIST benchmark. Consider Figure 8 where we vary the number of neurons for both MLPs and SNNs; we can observe that the accuracy of the MLP plateaus around 100 hidden neurons, and the accuracy of the SNN plateaus around 300 neurons.

We also explored the neurons and synapses bit width, with the goal of finding the most compact size which is within 1% of the best accuracy on MNIST obtained by floating-point operators. We ended up using 8-bit operators and weights for the MLP. For the SNN, SNNwt uses 8-bit weights, and SNNwot uses 12-bit weights (8-bit weights $\times$ number of spikes). It is one of the assets of the learning algorithm of neural networks to be able to compensate for such low precision.

The MLP network is too large to lay out for a normal server (estimated footprint is about 73 mm^2), so we proceeded as follows: we estimated the area cost of both networks based on the individual size and number of all operators and registers, and we also laid out two small-scale versions of the network. The approximate area cost of the full networks is shown in Table 4, where the cost of individual operators and registers are estimated using the TSMC 65nm GPlus high-VT library. We can observe that the area cost of the MLP version is far larger (2.72x) than that of the SNN version in spite of the lower number of neurons, because of the high cost of the multipliers used in MLP neurons, while SNN neurons only use adders.

Comparison of the layout of two small-scale designs. These results are further confirmed by fully laying out two small-scale designs, see Table 5, where the number of entries is restricted to 4x4 inputs, the MLP uses 10 hidden neurons and 10 output neurons, while the SNN uses 20 neurons. Note that delay and energy ratios are on par with the area ratio, only the power costs of both networks are similar, in part because of the clock power

Table 5: Hardware Characteristics of SNN (4x4-20) and MLP (4x4-10-10).

Type	Area (mm^2)	Delay (ns)	Power (W)	Energy (nJ)
SNN	0.08	1.18	0.52	0.63
MLP	0.21	1.96	0.64	1.28

accounts for a larger share of the total power in the SNN version (60% vs. 20% in the MLP).

Area comparison at same accuracy. Finally, we also empirically compare the area costs of MLP and SNN when they achieve a similar accuracy. Since the SNN has a lower accuracy than the MLP, we reduce the number of neurons in the MLP until their accuracy are almost the same, i.e., with 15 neurons, the MLP accuracy is 92.07% vs. 91.82% for the best SNN MNIST accuracy we achieved, see Figure 8. However, the area of the MLP is then respectively 68.30% and 73.23% smaller than for SNNwt and SNNwot, further confirming that when factoring both cost and accuracy, MLP significantly outperforms SNN.

4.3 Spatially Folded

The estimated area cost of the spatially expanded versions of the networks described in Section 4.2 shows that such networks are too large for accelerators compatible with the low footprint expected for embedded systems. There exist prior MLP designs [6] with a footprint of a few mm^2, but they were targeting tasks of the UC Irvine machine-learning repository [15], which typically have few inputs and outputs; as a result, an MLP with 90 inputs, 10 hidden neurons and 10 output neurons was sufficient in this case. For embedded systems applications, an input image of 28x28 greyscale pixels is already on the low-end of the application spectrum, so it is fair to state that spatially expanded designs would be too costly for most devices.

While spatially expanding NN designs provide for maximum speed, it is possible to spatially *fold* NN designs: the principle is to time-share a few hardware neurons between the many logical neurons of the target network; spatial folding has been adopted in early hardware neural networks when the number of transistors was low [33], and in recent convolutional/deep neural network accelerators [16]. We now consider and compare spatially folded versions of both the MLP and SNN designs in order to achieve NN accelerators with more realistic footprints.

4.3.1 MLP.

Hardware neuron. The high cost of the spatially expanded design is essentially due to the high connectivity (high number of inputs and synaptic multipliers) of the neurons. So a spatially folded design consists in putting a bound on the maximum number of inputs each hardware neuron can have; let us call n_i that number. As a result, a folded hardware neuron must accumulate the inputs $\times$ weights products by chunks of size n_i, and it must contain a register to store the partial product, see Figure 10 for an example when $n_i = 2$. Since the weights are unique to each input, they must be brought in for each chunk; so the neuron also contains n_i registers for the inputs, and an SRAM table stores all the weights

Figure 9: Layout of MLP (left, $n_i = 16$), SNN (right, $n_i = 16$).

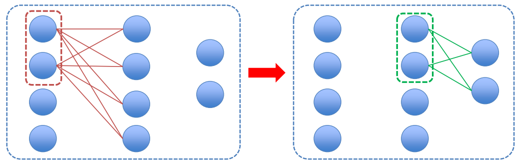
Figure 10: MLP scheduling on spatially folded design.

needed by the neuron (the SRAM table has a single port, of width $n_i \times 8$ bits; since it must contain all N_i inputs, its depth is $\frac{N_i}{n_i}$). The corresponding hardware MLP neuron design is shown in Figure 11.

Scheduling. Due to spatial folding, a hardware neuron now needs $\lceil \frac{N_i}{n_i} \rceil + 1$ cycles to compute its output (the "+1" term is for the multiplier and adder of the activation function). While this multi-cycle computation forbids a fully pipelined execution (processing a new image every cycle) as for the expanded design, it is still possible to implement a staggered pipeline where each stage requires multiple execution cycles (as for most floating-point operations in processors). On the other hand, the smaller number of inputs allows to achieve a higher clock time than for a spatially expanded design, see Table 7, where the critical path delay of the hardware neuron naturally decreases as the number of inputs n_i decreases. Note that the output of each neuron of the hidden layer is computed in parallel; it is then buffered in the output register of the neuron while the neurons of the output layer use them as inputs, by chunks of n_i again (same hardware neuron characteristics).

Measurements. In Table 7, we report the area, delay, power and energy for the MLP used for MNIST using a spatially folded neuron design (same number of hardware neurons as in the expanded design though). We observe that a hardware neuron with $n_i = 16$ inputs (vs. 784 inputs for the expanded neuron design) leads to an overall design which is 38.84x smaller than the expanded design, far more compatible with embedded designs, and processing one MNIST input only requires 57 cycles at 435MHz. A hardware neuron with $n_i = 4$ inputs leads to a design which is 117.76x smaller and requires 223 cycles per input at 435MHz. On the other hand, the energy of the folded designs is about 5x that of the expanded design; in fact the power of the expanded design is naturally high due to the high number of operators, but one image is treated in 15.16 ns (4 cycles of 3.79ns) vs. 128.25ns (57 cycles of 2.25 ns) to 1975.68 ns (882 cycles of 2.24ns) in the folded design.

4.3.2 SNN.

Spatial folding of the SNN neuron follows several of the same principles as for the MLP neuron. The number of inputs is similarly limited to n_i, with weights stored in

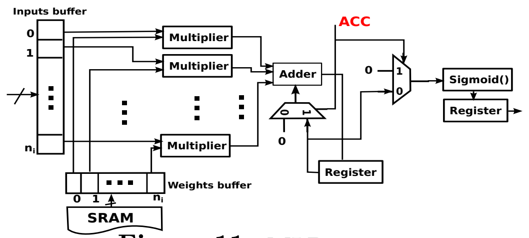

Figure 11: MLP neuron.

Table 6: SRAM characteristics for synaptic storage.

Design	$ni = 1$		$ni = 4$		$ni = 8$		$ni = 16$	
	SNN	MLP	SNN	MLP	SNN	MLP	SNN	MLP
SRAM width	128		128		128		128	
SRAM depth	784		200		128		128	
Read Energy (pJ)	44.41		33.05		32.46		32.46	
Area (μm^2)	108351		46002		40772		40772	
# Banks	19	8	75	28	150	55	300	110
Total Energy (nJ)	0.84	0.31	2.48	0.93	4.87	1.79	9.74	3.56
Total Area (mm^2)	2.06	0.76	3.45	1.29	6.12	2.24	12.23	4.48

an associated SRAM table (width of $n_i \times 8$ bits) and fed into n_i registers every cycle. Naturally, unlike the MLP folded neuron, the SNN folded neuron contains an adder dealing with n_i inputs instead of a multiplier; upon every group of n_i spikes, the potential is incremented. After all inputs have been processed, the potentials of all neurons are passed to a max operator. This max operator could also be folded, just like neurons, by limiting its number of inputs, but we found that a two-level max tree (15 max with 20 inputs, followed by one max with 15 inputs) only accounts for 5.6% (178,448 μm^2) of the area of the smallest SNN folded neuron with $n_i = 1$, so we decided to keep a fully expanded max tree.

We report the area, delay, energy and number of cycles for both the folded SNNwt and SNNwot in Table 7. SNNwt has a significantly lower area than SNNwot because SNNwot can treat all spikes of a single synapse simultaneously and thus requires operators which can accommodate more simultaneous inputs than SNNwt (n_i times the maximum number of spikes which can be received for one pixel). On the other hand, SNNwt must emulate the whole sequence of the image presentation and the associated leakage process, corresponding to 500ms; by decomposing the sequence into steps of 1 ms (an already coarse granularity), SNNwt requires 500 steps (i.e., 500 clock cycles) to complete the computation. With folding, this number of steps is further multiplied by the ratio of the number of inputs by n_i, i.e., approximately $\frac{784}{n_i} \times 500$. As a result, while SNNwt is competitive cost-wise, it is not competitive time-wise with SNNwot. Consequently, we will focus on the comparison between SNNwot and MLP.

4.3.3 *Comparison.*

MLP vs. SNN. While for the fully expanded version, the cost of an SNN accelerator is several times lower than that of an MLP, this ratio no longer holds for folded versions. Even with a fairly large degree of parallelism in the processing of inputs, e.g., $n_i = 16$, the area of a folded MLP is 2.57x lower than that of a folded SNNwot (and even slightly lower than the cost of a folded SNNwt). The main cause of the discrepancy is related to the synaptic storage required for the SNN. More neurons are needed in the SNN to achieve the best possible accuracy (though still significantly lower than MLP), and as a result, the total number of synaptic weights is significantly larger ($784 \times 100 + 100 \times 10 = 79,400$ weights for the MLP, i.e., 784×100 for the hidden layer and 100×10 for the output layer vs. $784 \times 300 = 235,200$ weights for the SNN); as a reference, we indicate in Table 6, the SRAM storage cost for different values of n_i for both MLP and SNN, using 128-bit SRAM banks. Energy-wise, the advantage of MLP is a bit less significant because the higher cost and complexity of the multipliers both increase the delay and the power cost of individual operations; still, the MLP is 2.41x more energy efficient than SNNwot at $n_i = 16$, and even 2.71x at $n_i = 1$ because of the lower number of multipliers.

Table 7: Hardware characteristics of spatially folded SNN and MLP.

Type	# Inputs per operator	Area (no SRAM) (mm^2)	Total Area (mm^2)	Delay (ns)	Energy (uJ)	# Cycles (per image)
SNNwot (28x28-300)	$ni = 1$	1.11	3.17	1.24	1.03	791
	$ni = 4$	1.89	5.34	1.48	0.68	203
	$ni = 8$	2.79	8.91	1.76	0.67	105
	$ni = 16$	4.10	16.33	1.84	0.70	56
	expanded[3]	26.79	46.06	3.17	0.03	3
SNNwt (28x28-300)	$ni = 1$	0.48	2.56	1.15	471.58	791*500
	$ni = 4$	0.84	4.36	1.11	315.33	203*500
	$ni = 8$	1.19	7.45	1.18	307.09	105*500
	$ni = 16$	1.74	14.25	1.84	325.69	56*500
	expanded[3]	19.62	38.89	2.61	214.70	500
MLP (28x28-100 -10)	$ni = 1$	0.29	1.05	2.24	0.38	882
	$ni = 4$	0.62	1.91	2.24	0.29	223
	$ni = 8$	1.02	3.26	2.25	0.30	113
	$ni = 16$	1.88	6.36	2.25	0.29	57
	expanded[3]	73.14	79.63	3.79	0.06	4

[3]Estimated numbers, based on synthesized version of operators

Overall, one can only observe that folded MLPs appear to be attractive accelerators with respect to SNNs in terms of cost, energy and accuracy. SNN+BP can significantly bridge the accuracy gap, as shown in Section 3.2, though not entirely, still making MLP+BP a better alternative. Nevertheless, there is one domain where SNN+STDP still shines with respect to MLP+BP: STDP is an *online* learning process, allowing the network to learn and be used at the same time, a characteristic which may turn out to be useful for certain applications. As a result, in the next section, we evaluate the cost of implementing STDP in hardware.

MLP & SNN accelerators vs. GPU. While the focus of this study is to understand which neural network accelerator approach one should pursue, we still provide a comparison with the corresponding software models implemented on a GPU, as a reference. We implemented CUDA codes which correspond as closely as possible to the two most efficient accelerator versions

Table 8: Speedups and energy benefits over GPU.

		GPU	$ni = 1$	$ni = 16$	expanded
Speedup	SNNwot	1	59.10	543.43	6086.46
	SNNwt	1	0.12	1.14	44.60
	MLP	1	40.44	626.03.04	5409.63
Energy Benefit	SNNwot	1	2799.72	4132.53	31542.31
	SNNwt	1	6.15	8.90	13.51
	MLP	1	12743.14	16365.61	79151.75

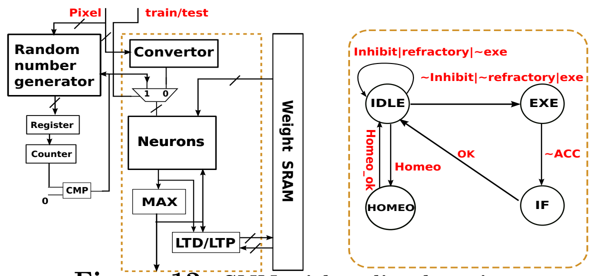

Figure 12: SNN with online learning.

(MLP and SNNwot), and in order to ensure good quality code we have resorted to the CUDA BLAS library (CUBLAS) [34], especially the `sgmv` routine [35, 36]. We use a recent NVIDIA K20M GPU, and we report the speedups for different values of n_i over the same GPU implementation (corresponding to $n_i = 1$) in Table 8. Even for $n_i = 1$ the accelerators speedups are significant for at least three reasons: the time to fetch data from global memory to the computational operators, the lack of reuse for the target operations, and the small size of the data structures (100 to 300 neurons, 784 inputs) [35, 36].

4.4 Online Learning

The neuron-level STDP circuit manages several information through a simple finite-state machine, see Figure 12.(b). First, it records the time elapsed since the last output spike for each neuron; this information will be used to later implement LTP and LTD, see Section 2.2. It also manages a refractory counter and an inhibitory counter. The refractory counter is reset once the neuron fires; the inhibitory counter is reset when any other neuron fires (and then sends an inhibitory signal); when either counter is active, the neuron ignores the input spikes and does not modify its potential. In order to implement LTP and LTD, a neuron also keeps an internal counter which is reset every time it fires. At the next firing event, it compares this counter to the LTP delay, and if it is smaller or equal, the corresponding weight is increased (LTP) by a constant increment (1), otherwise it is similarly decreased (LTD).

As a result, the neuron circuit determines which synaptic weights to increase or decrease; it applies constant increments/decrements of 1. As explained in Section 2.2, we use the following expression: $v_j(T_2) = v_j(T_1) \times e^{-\frac{T_2-T_1}{T_{leak}}}$ to model the leak. We implement this expression in hardware using piecewise linear interpolation.

Homeostasis. Finally, the neuron also records the number of times it fired within a homeostasis epoch (a constant period of time after which the threshold is adjusted in all neurons; we use 1,500,000 ms, i.e., every 3,000 images). Once the homeostasis epoch is finished (it is managed by a single external counter, common to all neurons), the number of firings of each neuron is compared to a preset *homeostasis_threshold*, and the neuron firing threshold is adjusted along the expression provided in Section 2.2.

4.4.1 *Overhead of STDP.*

In Table 9, we present the characteristics of the layout

Table 9: Hardware features (SNN with online learning).

Type		Area (no SRAM) (mm^2)	Total Area (mm^2)	Delay (ns)	Energy (mJ)
SNN	ni=1	2.55	4.92	1.23	0.71
	ni=4	3.33	7.10	1.48	0.37
	ni=8	4.26	10.70	1.81	0.32
	ni=16	6.44	19.06	1.88	0.33

of the SNNwt neuron with STDP for different values of n_i. We can observe that the total area (with SRAM) is about 1.34x ($n_i = 16$) to 1.93x ($n_i = 1$) times larger than the SNNwt neuron alone, described in Table 7. The cycle time increases by 7% at most, and the energy is about 1.02x ($ni = 16$) to 1.50x ($ni = 1$) times larger than the SNNwt neuron. Consequently, one can observe that the hardware overhead of implementing STDP is quite small. So the real value of hardware accelerators implementing SNN+STDP seems to be in the tasks specifically requiring permanent (online) learning.

4.5 Validation on Additional Workloads

We used the MNIST benchmark as a driving example throughout our study because it is one of the very few benchmarks used by both researchers from the machine-learning *and* the neuroscience domain. Nevertheless, we also want to validate our observations on other types of workloads. We picked two of the main important domains: object and speech recognition, namely the MPEG-7 CE Shape-1 Part-B [14] benchmark, and the Spoken Arabic Digits (SAD) dataset [15].

For these two workloads, we conducted exactly the same exploration as for the MNIST workload for both SNN and MLP, i.e., defining the optimal set of hyper-parameters of each model, especially the number of neurons, and then determining their hardware cost.

For MPEG-7, the optimized MLP (28x28-15-10) and SNN (28x28-90) achieve an accuracy of 99.7% and 92%, respectively. When n_i varies from 1 to 16, the spatially folded SNNwot consumes 3.81x-5.57x more area, and 3.20x-5.08x more energy than the spatially folded MLP.

For SAD, the optimized MLP (13x13-60-10) and SNN (13x13-90) achieve an accuracy of 91.35% and 74.7%, respectively. When n_i varies from 1 to 16, the spatially folded SNNwot consumes 1.27x-1.31x more area, and 1.24x-1.26x more energy than the spatially folded MLP.

In a nutshell, our results on these benchmarks are consistent with the results obtained on MNIST: SNN achieves lower accuracy and requires a higher cost than MLP.

5. DISCUSSION

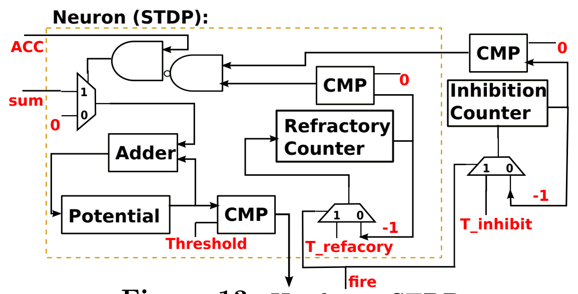

Figure 13: Hardware STDP.

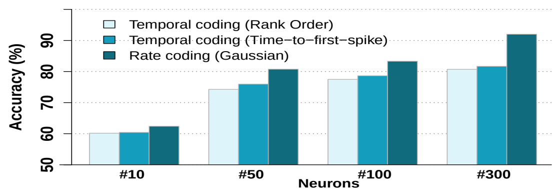

Figure 14: SNN models with different coding schemes.

Coding Scheme. In addition to rate coding schemes studied in previous sections, we had also explored two common temporal coding schemes, rank order coding [28] and time-to-first-spike coding [37, 38] for SNN. While both rate coding [39, 40] and temporal coding [41] have been shown to be biologically plausible, we observed on MNIST that the SNN with temporal coding schemes is significantly less accurate than the SNN with rate coding schemes (82.14% vs. 91.82%), see Figure 14, where both SNN models use the same network topology. For the sake of brevity, we only discuss rate coding in previous sections.

TrueNorth. TrueNorth is a digital many-core spiking-neuron chip developed by IBM [42]. While the most recent article [42] did not report TrueNorth's accuracy on standard machine learning benchmark datasets such as MNIST, MPEG-7 and SAD, we were able to extract related information from previous articles published by the same group. For example, according to Merolla et al. [8], a TrueNorth core (having the same architecture but 4 times larger [42]) that contains 1024 inputs, 256 output neurons, and 1024×256 synapses (with 9-bit precision) within $4.2mm^2$ area under IBM 45nm process and runs at 1MHz,[4] can achieve the accuracy of 89% on MNIST [43, 42]. Here we compare TrueNorth with the SNNwot folded with $ni = 1$, an SNN accelerator sharing similar characteristics with the TrueNorth core (e.g., both process one input for all output neurons at a time, see Section 4.3.2). As IBM has not released the resources of TrueNorth, we made a best effort to reimplement the TrueNorth core down to the layout (using TSMC 65nm GPlus high VT standard library) according to the descriptions in [8]. Using our reimplementation, we observe that SNNwot outperforms TrueNorth in terms of area ($3.17mm^2$ vs. $3.30mm^2$), speed (0.98us vs. 1024us), energy (1.03uJ vs. 2.48uJ) and accuracy (90.85% vs. 89%); however, it is likely that our reimplementation, and this comparison, does not make justice to TrueNorth design optimizations that were not described in the article.

6. RELATED WORK

Neural Network applications. Thanks to recent progress in machine-learning, certain types of neural networks, especially Deep Neural Networks [3] and Convolutional Neural Networks [13], have become state-of-the-art machine-learning techniques [44, 4] across a broad range of applications such as web search [45], image analysis [46] or speech recognition [47].

[4]TrueNorth adopts a physical frequency of 1MHz so that the largest possible spiking frequency can become lower than 1KHz, a level consistent with neuroscience findings [39, 40].

Neural Network accelerators.
Machine-Learning accelerators. Because these models need costly computing-intensive architectures such as GPUs to be run [48, 49], there is a growing interest in developing custom accelerators in order to broaden their usage in embedded devices and data centers. For instance, Farabet et al. [5] proposed the Neuflow architecture which can run a CNN in real-time, Chen's group proposed DianNao [16], a small-footprint accelerator to implement CNNs and DNNs, DaDianNao [50], a custom multi-chip architecture for CNNs and DNNs, and ShiDianNao [51], a super efficient CNN accelerator for embeded system, mobile platform. Beyond cost, Esmaeilzadeh et al. [52] has proposed to use an MLP accelerator to speed up (by approximating) certain functions within a program.
Neuroscience accelerators. Together with these accelerators for machine-learning tasks, a number of hardware neural networks have been proposed to accelerate (often large-scale) biological neural network models. For instance Vogelstein et al. [53] have investigated several neuron models in hardware, Smith [54] has studied the capabilities and hardware implementations of LIF neuron models, Schemmel et al. [55] investigate wafer-scale spiking neural networks as part of the BrainScale project, SpiNNaker proposes to implement a billion (spiking) neurons [56], IBM has proposed several digital neuromorphic implementations [8, 57] with the goal of implementing as many neurons as in a cat brain as part of the SyNAPSE project. Slightly more application than neuroscience-oriented, Joubert et al. [58] has investigated a network of analog spiking neurons for signal processing tasks, Roclin et al. [59] have analyzed how to reduce the cost of a SNN+STDP model without effectively proposing a hardware design, and Qualcomm recently introduced the Zeroth [9] project for a spiking neural network accelerator.
Neuroscience vs. Machine-Learning Neural Network models. While there are innumerable studies on respectively neural network models derived from neuroscience and neural networks used as machine-learning algorithms, there are very few studies which effectively compare both types of neural networks. MNIST is one of the rare elements of comparison between the two domains as one of the few benchmarks used in some neuroscience and some machine-learning studies. Unfortunately, neuroscience studies rarely use MNIST to its full extent, e.g., all ten digits, all training inputs, all testing inputs, etc, thus still making a fair comparison difficult. For example [60] reported an accuracy of 91.90% (down to 89.4% for a 5-bit resolution synapses) when a RBM composed of LIF neurons (824 visible and 500 hidden neurons) with STDP synapses using a newly proposed event-driven contrastive divergence sampling. When standard contrastive divergence is used for the above architecture with stochastic units the performance is 93.6%. Arthur et al. [61] achieved 94% accuracy (down to 89% when implemented with finite resolution synapses of their neurosynaptic core) using LIF neurons with an RBM layout (256 visible and 484 hidden neu-

rons). Other results are in the range of 96.8% [62], 91.64% [63], but they are not directly comparable because of a partial and different usage of the MNIST dataset, synaptic update rule, synaptic model, and spiking neuron model. Querlioz et al. [11] perform one of the few thorough evaluations of MNIST on SNN in the context of a spiking neural network architecture based on memristive devices. We also mentioned the work of Diehl et al. [23] which achieved a slightly better MNIST accuracy (95% vs. 93.50%) than Querlioz et al., albeit using 6400 neurons vs. 300 neurons; with 400 neurons, their accuracy is only 82.9%.

A small number of works specifically focus on the comparison between neuroscience and machine-learning neural networks. For instance, Serre et al. [64] propose HMAX, a neural network inspired from the visual cortex (albeit using rate-based, not spiking, neurons) and shown to be competitive with established machine-learning models (not neural networks though). More closely, Masquelier et al. [10] compare a SNN-based neural network (similar to SNNwot) against HMAX and demonstrate competitive results on object recognition tasks; Nere et al. [65] use a more biologically precise SNN+STDP model (closer to SNNwt) and build an HMAX-like neural network, unfortunately, their goal is not to compare against HMAX. Farabet et al. [66] compare SNN and CNN models mapped to FPGAs; they outline the lower computational requirements of SNNs with respect to multi-layer CNNs, however, they perform limited exploration of the possible hardware designs (for both SNNs and CNNs), considering only fully expanded (non time-multiplexed) designs (with time information for SNNs), similar to our fully expanded SNNwt. Rast et al.[67] propose a different approach, and they map an MLP on an architecture designed for SNNs (SpiNNaker [56]), but it does not help clarify the debate between MLP and SNN, it just provides an alternative approach for implementing MLPs in hardware.

7. CONCLUSIONS AND FUTURE WORK

The motivation for this work is the growing interest in cognitive tasks at large, and the existence of two, largely distinct, candidate neural models for these accelerators: one from neuroscience and the other from machine-learning.

In order to help decide whether industrial applications should rather target machine-learning or neuroscience inspired models, we compare two the best known models in each approach on one of the only benchmarks (MNIST) used in both neuroscience and machine-learning studies, and we further validate our results on an object recognition and a speech recognition benchmark.

Our study has inherent limitations: current SNN and machine-learning NN algorithms, current best effort at hardware implementation, and a restricted set of target workloads. But within these limitations, we can make the following observations. While the neuroscience-inspired model (SNN+STDP) is intellectually attractive due to its closer relationship to the brain, we observe that a corresponding hardware accelerator is currently not competitive with a hardware accelerator based on a classic machine-learning model, across all characteristics (accuracy, area, delay, power, energy) for area footprints of a few mm^2.

Only for very large-scale implementations, SNNs could become more attractive (area, delay, energy and power, but still not accuracy) than machine-learning models.

We also identify the cause of the accuracy discrepancy between SNN+STDP and MLP+BP, i.e., the nature of the STDP learning algorithm, and spike coding.

Acknowledgements

This work is partially supported by the NSF of China (under Grants 61133004, 61222204, 61221062, 61303158, 61432016, 61472396, 61473275, 61522211 61532016), the 973 Program of China (under Grant 2015CB358800), the Strategic Priority Research Program of the CAS (under Grants XDA06010403, XDB02040009), the International Collaboration Key Program of the CAS (under Grant 171111KYSB20130002), and the 10000 talent program. And this work is supported by French ANR MHANN. This research is supported by the Intel Collaborative Research institute for Computational Intelligence (ICRI-CI).

8. REFERENCES

[1] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," in *International Symposium on Field Programmable Gate Arrays*, FPGA '06, (New York, NY, USA), pp. 21–30, ACM, Feb. 2006.

[2] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach, "Accelerating compute-intensive applications with gpus and fpgas," in *Symposium on Application Specific Processors*, pp. 101–107, IEEE, June 2008.

[3] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *International Conference on Machine Learning*, (New York, New York, USA), pp. 473–480, ACM Press, 2007.

[4] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, pp. 1–9, 2012.

[5] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "NeuFlow: A runtime reconfigurable dataflow processor for vision," in *CVPR Workshop*, pp. 109–116, Ieee, June 2011.

[6] O. Temam, "A Defect-Tolerant Accelerator for Emerging High-Performance Applications," in *International Symposium on Computer Architecture*, (Portland, Oregon), 2012.

[7] Y. Dan and M. M. Poo, "Hebbian depression of isolated neuromuscular synapses in vitro.," *Science (New York, N.Y.)*, vol. 256, pp. 1570–3, June 1992.

[8] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. Modha, "A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm," in *IEEE Custom Integrated Circuits Conference*, pp. 1–4, IEEE, Sept. 2011.

[9] S. Kumar, "Introducing Qualcomm Zeroth Processors: Brain-Inspired Computing," 2013.

[10] T. T. Masquelier and S. J. Thorpe, "Unsupervised learning of visual features through spike timing dependent plasticity.," *PLoS computational biology*, vol. 3, p. e31, Feb. 2007.

[11] D. Querlioz, O. Bichler, and C. Gamrat, "Simulation of a memristor-based spiking neural network immune to device variations," in *International Joint Conference on Neural Networks*, (San Jose, CA), pp. 1775–1781, IEEE, July 2011.

[12] B. Belhadj, A. Joubert, Z. Li, R. Heliot, and O. Temam, "Continuous Real-World Inputs Can Open Up Alternative Accelerator Designs," in *International Symposium on Computer Architecture*, 2013.

[13] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, 1998.

[14] "Shape data for the mpeg-7 core experiment ce-shape-1."

[15] A. Asuncion and D. J. Newman, "Uci machine learning repository."

[16] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2014.

[17] P. Pinheiro and R. Collobert, "Recurrent Convolutional Neural Networks for Scene Parsing," *arXiv preprint arXiv:1306.2795*, no. June, 2013.

[18] N. Brunel and S. Sergi, "Firing frequency of leaky intergrate-and-fire neurons with synaptic current dynamics.," *Journal of theoretical biology*, vol. 195, no. 1, pp. 87–95, 1998.

[19] E. Marder and J.-M. Goaillard, "Variability, compensation and homeostasis in neuron and network function.," *Nature reviews. Neuroscience*, vol. 7, pp. 563–74, July 2006.

[20] D. Querlioz, O. Bichler, P. Dollfus, and C. Gamrat, "Immunity to Device Variations in a Spiking Neural Network with Memristive Nanodevices," *IEEE Transactions on Nanotechnology*, vol. 12, pp. 288–295, May 2013.

[21] D. CireÅ§an, U. Meier, and J. Schmidhuber, "Multi-column Deep Neural Networks for Image Classification," in *International Conference of Pattern Recognition*, pp. 3642–3649, 2012.

[22] P. Simard, D. Steinkraus, and J. Platt, "Best practices for convolutional neural networks applied to visual document analysis," *International Conference on Document Analysis and Recognition*, vol. 1, no. Icdar, pp. 958–963, 2003.

[23] P. U. Diehl, S. Member, and M. Cook, "Unsupervised Learning of Digit Recognition Using Spike-Timing-Dependent Plasticity," *IEEE Transactions in Neural Networks and Learning Systems*, vol. 52538, pp. 1–6, 2014.

[24] H. Markram, J. Lübke, M. Frotscher, and B. Sakmann, "Regulation of Synaptic Efficacy by Coincidence of Postsynaptic APs and EPSPs," *Science*, vol. 275, pp. 213–215, Jan. 1997.

[25] G. Billings and M. C. W. van Rossum, "Memory retention and spike-timing-dependent plasticity.," *Journal of neurophysiology*, vol. 101, pp. 2775–88, June 2009.

[26] S. Song, K. D. Miller, and L. F. Abbott, "Competitive Hebbian learning through spike-timing-dependent synaptic plasticity.," *Nature neuroscience*, vol. 3, pp. 919–26, Sept. 2000.

[27] R. Serrano-Gotarredona, M. Oster, P. Lichtsteiner, A. Linares-Barranco, R. Paz-Vicente, F. Gomez-Rodriguez, H. K. Riis, T. Delbruck, S. C. Liu, S. Zahnd, A. M. Whatley, P. Hafliger, G. Jimenez-Moreno, A. Civit, T. Serrano-Gotarredona, A. Acosta-Jimenez, and B. Linares-Barranco, "AER Building Blocks for Multi-Layer Multi-Chip Neuromorphic Vision Systems," in *In Y. Weiss, B. Sch olkopf, J. Platt (Eds.), Advances in neural information processing (NIPS)*, no. 1, pp. 1217–1224, 2006.

[28] S. Thorpe and J. Gautrais, "Rank order coding," *Computational neuroscience: Trends in Research*, vol. 13, pp. 113–119, 1998.

[29] D.-u. Lee, J. D. Villasenor, S. Member, W. Luk, and P. H. W. Leong, "A Hardware Gaussian Noise Generator Using the Box-Muller Method and Its Error Analysis," *IEEE Transactions on Computers*, vol. 55, no. 6, pp. 659–671, 2006.

[30] P. Leong, J. Villasenor, and R. Cheung, "Ziggurat-based hardware gaussian random number generator," *International Conference on Field Programmable Logic and Applications, 2005.*, pp. 275–280, 2005.

[31] J. S. Malik, J. N. Malik, A. Hemani, and N. Gohar, "Generating high tail accuracy Gaussian Random Numbers in hardware using central limit theorem," *2011 IEEE/IFIP 19th International Conference on VLSI and System-on-Chip*, pp. 60–65, Oct. 2011.

[32] D. Querlioz, P. Dollfus, O. Bichler, and C. Gamrat, "Learning with memristive devices: How should we model their behavior?," in *Nanoscale Architectures (NANOARCH)*, pp. 150–156, 2011.

[33] M. Holler, S. Tam, H. Castro, and R. Benson, "An electrically trainable artificial neural network (ETANN) with 10240 'floating gate' synapses," in *Artificial neural networks*, (Piscataway, NJ, USA), pp. 50–55, IEEE Press, 1990.

[34] NVIDIA, "Cuda cublas library 5.5."

[35] N. Fujimoto, "Faster matrix-vector multiplication on GeForce 8800GTX," *2008 IEEE International Symposium on Parallel and Distributed Processing*, pp. 1–8, Apr. 2008.

[36] R. Nath, S. Tomov, T. T. Dong, and J. Dongarra, "Optimizing symmetric dense matrix-vector multiplication on GPUs," *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '11*, p. 1, 2011.

[37] F. Rieke, D. Warland, R. De Ruyter Van Steveninck, and W. Bialek, "Spikes: Exploring the Neural Code," 1997.

[38] W. Gerstner and W. M. Kistler, *Spiking Neuron Models.* Cambridge University Press, 2002.

[39] R. C. Froemke and Y. Dan, "Spike-timing-dependent synaptic modification induced by natural spike trains," *Nature*, vol. 416, pp. 433–438, Mar. 2002.

[40] P. J. Sjöström, G. G. Turrigiano, and S. B. Nelson, "Rate, Timing, and Cooperativity Jointly Determine Cortical Synaptic Plasticity," *Neuron*, vol. 32, pp. 1149–1164, Dec. 2001.

[41] D. A. Butts, C. Weng, J. Jin, C.-I. Yeh, N. A. Lesica, J.-M. Alonso, and G. B. Stanley, "Temporal precision in the neural code and the timescales of natural vision," *Nature*, vol. 449, pp. 92–95, Sept. 2007.

[42] P. A. Merolla, J. V. Arthur, R. Alvarez-icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, "A million spiling-neuron interated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, 2014.

[43] S. K. Esser, A. Andreopoulos, R. Appuswamy, P. Datta, D. Barch, A. Amir, J. Arthur, A. Cassidy, M. Flickner, P. Merolla, S. Chandra, N. Basilico, S. Carpin, T. Zimmerman, F. Zee, R. Alvarez-Icaza, J. a. Kusnitz, T. M. Wong, W. P. Risk, E. McQuinn, T. K. Nayak, R. Singh, and D. S. Modha, "Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores," *Proceedings of the International Joint Conference on Neural Networks*, no. December, 2013.

[44] Q. V. Le, M. A. Ranzato, R. Monga, M. Devin, K. Chen, G. S. Corrado, J. Dean, and A. Y. Ng, "Building High-level Features Using Large Scale Unsupervised Learning," in *International Conference on Machine Learning*, June 2012.

[45] P. Huang, X. He, J. Gao, and L. Deng, "Learning deep structured semantic models for web search using clickthrough data," in *International Conference on Information and Knowledge Management*, 2013.

[46] V. Mnih and G. Hinton, "Learning to Label Aerial Images from Noisy Data," in *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pp. 567–574, 2012.

[47] G. Dahl, T. Sainath, and G. Hinton, “Improving Deep Neural Networks for LVCSR using Rectified Linear Units and Dropout,” in *International Conference on Acoustics, Speech and Signal Processing*, 2013.

[48] A. Coates, P. Baumstarck, Q. Le, and A. Y. Ng, “Scalable learning for object detection with GPU hardware,” *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4287–4293, Oct. 2009.

[49] K.-S. Oh and K. Jung, “GPU implementation of neural networks,” *Pattern Recognition*, vol. 37, pp. 1311–1314, June 2004.

[50] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, “DaDianNao: A Machine-Learning Supercomputer,” in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-47)*, pp. 609–622, 2014.

[51] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, X. Feng, Y. Chen, and O. Temam, “ShiDianNao: Shifting Vision Processing Closer to the Sensor,” in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pp. 92–104, 2015.

[52] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, “Neural Acceleration for General-Purpose Approximate Programs,” in *International Symposium on Microarchitecture*, no. 3, pp. 1–6, 2012.

[53] R. J. Vogelstein, U. Mallik, J. T. Vogelstein, and G. Cauwenberghs, “Dynamically reconfigurable silicon array of spiking neurons with conductance-based synapses,” *IEEE Transactions on Neural Networks*, vol. 18, no. 1, pp. 253–265, 2007.

[54] J. E. Smith, “Efficient Digital Neurons for Large Scale Cortical Architectures,” in *International Symposium on Computer Architecture*, 2014.

[55] J. Schemmel, D. Briiderle, A. Griibl, M. Hock, K. Meier, and S. Millner, “A wafer-scale neuromorphic hardware system for large-scale neural modeling,” in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pp. 1947–1950, IEEE, May 2010.

[56] M. M. Khan, D. R. Lester, L. A. Plana, A. Rast, X. Jin, E. Painkras, and S. B. Furber, “SpiNNaker: Mapping neural networks onto a massively-parallel chip multiprocessor,” in *IEEE International Joint Conference on Neural Networks (IJCNN)*, pp. 2849–2856, Ieee, 2008.

[57] J.-s. Seo, B. Brezzo, Y. Liu, B. D. Parker, S. K. Esser, R. K. Montoye, B. Rajendran, J. A. Tierno, L. Chang, D. S. Modha, and Others, “A 45nm CMOS neuromorphic chip with a scalable architecture for learning in networks of spiking neurons,” in *IEEE Custom Integrated Circuits Conference*, pp. 1–4, IEEE, Sept. 2011.

[58] A. Joubert, B. Belhadj, O. Temam, and R. Heliot, “Hardware Spiking Neurons Design: Analog or Digital?,” in *International Joint Conference on Neural Networks*, (Brisbane), 2012.

[59] D. Roclin, O. Bichler, C. Gamrat, S. J. Thorpe, and J.-o. Klein, “Design Study of Efficient Digital Order-Based STDP Neuron Implementations for Extracting Temporal Features,” in *International Joint Conference on Neural Networks*, 2013.

[60] E. Neftci, S. Das, B. Pedroni, K. Kreutz-Delgado, and G. Cauwenberghs, “Event-driven contrastive divergence for spiking neuromorphic systems,” *Frontiers in Neuroscience*, vol. 7, no. January, pp. 1–14, 2014.

[61] J. Arthur, P. Merolla, F. Akopyan, R. Alvarez, A. Cassidy, S. Chandra, S. Esser, N. Imam, W. Risk, D. Rubin, R. Manohar, and D. Modha, “Building block of a programmable neuromorphic substrate: A digital neurosynaptic core,” in *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pp. 1–8, June 2012.

[62] J. M. Brader, W. Senn, and S. Fusi, “Learning real-world stimuli in a neural network with spike-driven synaptic dynamics.,” *Neural computation*, vol. 19, pp. 2881–912, Nov. 2007.

[63] M. Beyeler, N. D. Dutt, and J. L. Krichmar, “Categorization and decision-making in a neurobiologically plausible spiking network using a stdp-like learning rule,” *Neural Networks*, vol. 48, no. 0, pp. 109 – 124, 2013.

[64] T. Serre, L. Wolf, and T. Poggio, “Object Recognition with Features Inspired by Visual Cortex,” in *Conference on Computer Vision and Pattern Recognition*, pp. 994–1000, Ieee, 2005.

[65] A. Nere, U. Olcese, D. Balduzzi, and G. Tononi, “A neuromorphic architecture for object recognition and motion anticipation using burst-STDP.,” *PloS one*, vol. 7, p. e36958, Jan. 2012.

[66] C. Farabet, R. Paz, J. Pérez-Carrasco, C. Zamarreño Ramos, A. Linares-Barranco, Y. Lecun, E. Culurciello, T. Serrano-Gotarredona, and B. Linares-Barranco, “Comparison between Frame-Constrained Fix-Pixel-Value and Frame-Free Spiking-Dynamic-Pixel ConvNets for Visual Processing.,” *Frontiers in neuroscience*, vol. 6, p. 32, Jan. 2012.

[67] a. D. Rast, L. a. Plana, S. R. Welbourne, and S. Furber, “Event-driven MLP implementation on neuromimetic hardware,” *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, June 2012.