Conversion of analog to spiking neural networks using sparse temporal coding

Bodo Rueckauer
Institute of Neuroinformatics
University of Zurich and ETH Zurich
Zurich, Switzerland
Email: rbodo@ini.uzh.ch

Shih-Chii Liu
Institute of Neuroinformatics
University of Zurich and ETH Zurich
Zurich, Switzerland
Email: shih@ini.uzh.ch

***Abstract*—The activations of an analog neural network (ANN) are usually treated as representing an analog firing rate. When mapping the ANN onto an equivalent spiking neural network (SNN), this rate-based conversion can lead to undesired increases in computation cost and memory access, if firing rates are high. This work presents an efficient temporal encoding scheme, where the analog activation of a neuron in the ANN is treated as the instantaneous firing rate given by the time-to-first-spike (TTFS) in the converted SNN. By making use of temporal information carried by a single spike, we show a new spiking network model that uses 7-10X fewer operations than the original rate-based analog model on the MNIST handwritten dataset, with an accuracy loss of $< 1\%$.**

I. Introduction

Deep Learning networks achieve state-of-the-art in numerous machine learning applications, for instance object detection and classification, scene parsing, and video captioning [1], [2]. Highly accurate classification during inference comes at a cost: Typical Analog Neural Networks (ANNs) require floating-point multiply-accumulate operations (MACs) to compute the activation values of all the neurons in the network. Graphical Processing Units (GPUs) process these operations efficiently in parallel, but their power consumption can prohibit their use in embedded applications. Considerable effort is being devoted to the development of hardware accelerators as well as algorithmic improvements which enable the efficient execution of deep neural networks on these hardware platforms. Notable directions include reducing the precision of weights and / or activations [3], skipping computations when activation values are zero [4], and minimizing data movement [5].

Unmatched in power efficiency, the biological brain employs all-or-none pulses (spikes) to transmit information. Motivated by this paradigm, artificial *Spiking* Neural Networks (SNNs) are being developed to solve similar tasks as ANNs, but making use of cheaper "addition" operations instead of MACs, and leveraging sparsity in neuron activations, as neurons will only be active if driven by a strictly positive input. In tasks with a continuous stream of input data, SNNs combined with event-based sensors [6] allow for data-driven computation, making SNNs ideal for always-on, low-power applications. Neuromorphic hardware [7], [8] optimizes routing of spikes across the network and implements power-efficient computation.

Training deep SNNs directly can be difficult. Recent spike-based learning algorithms demonstrated promising results on the MNIST dataset ([9], [10]), and even natural image datasets [11], but the scalability of the learning algorithm to larger architectures and more challenging data sets has yet to be shown. Mapping a pre-trained ANN to an SNN of similar architecture has been applied successfully to natural image datasets like CIFAR-10 [12], and ImageNet [13], [14]. The encoding scheme underlying this SNN conversion approach is rate-based: The SNN neurons generate a sequence of discrete spikes, which, when averaged over the simulation duration, approximates the analog firing rate of the corresponding ANN neuron. This rate-based encoding becomes more accurate as the simulation duration of the SNNs is increased and more spikes are generated. The computational cost of the SNN also scales with the average firing rate of its neurons: Each spike entails fetching the weight values of the synaptic connections to neurons in the following layer, and updating the state variables (e.g., membrane potential) of those post-synaptic neurons. In the ANN, *all* neurons are updated *once*, using MAC operations; in the SNN, an *active subset* of neurons is updated *repeatedly*, using ADD operations. Because the energy cost of memory transfer can exceed the energy cost of computations by two orders of magnitude [15], the SNN may lose a potential performance advantage over the ANN as the rates increase.

The energy cost disparity of memory and computation motivates the search for a spike code that is more efficient than the rate-based code. Coupled with evidence of temporal codes in the brain [16], we propose in this paper an ANN-to-SNN conversion mechanism, where the analog activation values of the ANN neurons are represented by the inverse time-to-first-spike (TTFS) in the SNN neurons [17]. Thus, neurons in the SNN spike at most once during inference of one sample, combining the spatial (feature map) sparsity of SNNs with the temporal sparsity of ANNs.

Earlier work in this direction includes the HFirst [18] network which uses spike timing in object recognition: The max-pool operation is implemented by performing a temporal winner-take-all among neurons in a pooling region. The authors in [19] developed a method to convert ANNs to SNNs based on rate-coding, but with adapting thresholds that reduce the firing rates significantly compared to other rate-based methods. Several groups have trained SNNs directly, either in an unsupervised [11], [20] or supervised manner [21], [9]. Closely related to our coding scheme, Mostafa and colleagues [22] proposed an algorithm to train an SNN using an analytic

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

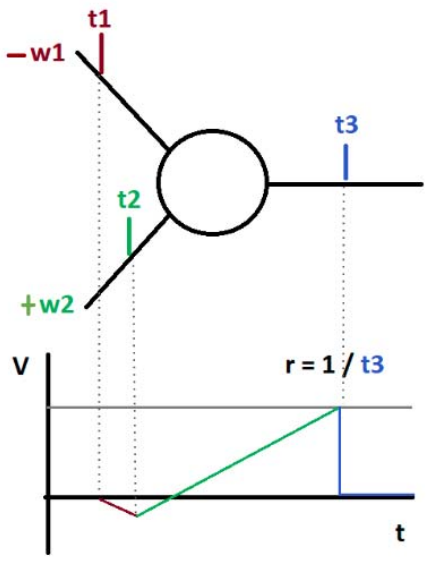

Figure 1. Spike generation mechanism with time-to-first-spike coding.

expression for the time-to-first-spike, which can be optimized via back-propagation. Where applicable, we compare their results with ours in Sec. III.

The three variants of our proposed TTFS encoding are described in Sec. II. The evaluation of these methods on the MNIST dataset is presented in Sec. III, and the results discussed in Sec. IV.

II. Methods

A. Time-to-first-spike base-line model ("TTFS base")

In general, the dynamics of the membrane potential $u_i(t)$ of a neuron i can be modeled with the following equation:

$$u_i(t) = \sum_{t_i^{(f)} \in \mathcal{F}_i} \eta_i(t - t_i^{(f)}) + \sum_{j \in \Gamma_i} \sum_{t_j^{(f)} \in \mathcal{F}_j} w_{ij} \epsilon_{ij}(t - t_j^{(f)}) + I_{\text{ext}}(t), \tag{1}$$

where η_i models the shape of an action potential (including a possible refractory period), the scalar values w_{ij} are the synaptic weights, the kernel ϵ_{ij} describes the post-synaptic potential (PSP) caused by an input spike from neuron j, and the external input current $I_{\text{ext}}(t)$ represents the bias. The set $\mathcal{F}_i = \{t_i^{(f)} \mid 1 \leq f \leq n\} = \{t \mid u_i(t) = \theta\}$ contains the output spike times, and Γ_i denotes the set of pre-synaptic neurons.

In this work, we require only the first spike of each neuron and can prevent additional spikes e.g., by making the refractory period very long. Equation (1) for the membrane potential up to the first spike of neuron i then simplifies to

$$u_i(t) = \sum_{j \in \Gamma_i} w_{ij} \epsilon_{ij}(t - t_j^{(0)}) + b_i t. \tag{2}$$

We choose a simple piecewise-linear form for the PSP kernel $\epsilon_{ij}(t - t_j^{(0)}) = \left[t - t_j^{(0)}\right] \mathcal{H}(t - t_j^{(0)})$, where $\mathcal{H}(x)$ is the Heaviside step function. The Heaviside function can be removed by introducing $\Gamma_i^< := \{j \mid t_j^{(0)} < t_i^{(0)}\}$ as the set of "causal neurons", i.e., pre-synaptic neurons whose spikes arrive at neuron i before its output spike is generated. Then

$$u_i(t) = \sum_{j \in \Gamma_i^<} w_{ij} \left[t - t_j^{(0)}\right] + b_i t. \tag{3}$$

In a simulation with time step dt, this explicit equation translates to a rate of increase

$$\frac{u_i(t + dt) - u_i(t)}{dt} = \sum_{j \in \Gamma_i^<} w_{ij} + b_i =: \mu_i. \tag{4}$$

This mechanism is illustrated in Figure 1. In order to determine the first instance in time when neuron i spikes, we set the membrane potential equal to the threshold, $u_i(t_i^{(0)}) = \theta$, and solve for $t_i^{(0)}$:

$$t_i^{(0)} = \frac{1}{\mu_i} \left(\theta + \sum_{j \in \Gamma_i^<} w_{ij} t_j^{(0)} \right) \tag{5}$$

The corresponding instantaneous firing rate r_i equals $1/t_i^{(0)}$. Successful ANN-to-SNN conversion implies that the SNN spike rate r_i is approximately equal to the corresponding activation a_i of the original ANN.[1] Then the output spike time in the SNN is inversely proportional to the ANN activation: $t_i^{(0)} = 1/a_i$. The update mechanism (3) constitutes the base model of the present work, labeled "TTFS base" in the remainder of the paper. A potential problem with the TTFS expression (5) derived from the base model can be seen by considering two neurons A, B which drive neuron C with connection strengths $w_A = 1$ and $w_B = -2$. If the two input neurons are activated with $a_A = 2$ and $a_B = 1$ respectively, the net effect on neuron C in the ANN would cancel out. In the SNN however, neuron A will fire twice as fast as B, potentially driving C above threshold before the inhibitory input arrives. Raising the threshold to delay generation of output spikes in the post-synaptic neuron is not a viable solution because it will equally delay the arrival of input spikes.

B. TTFS with dynamic threshold ("TTFS dyn thresh")

In order to remedy the situation where neurons fire their first spike prematurely, i.e., before having received all input spikes, we replaced the constant threshold by a threshold that dynamically adapts to the observed input. In particular, the threshold of each neuron is increased by an amount equal to the input that is still missing. When an input spike arrives, the threshold is reduced by the amount equal to the corresponding synaptic strength. Thus, the likelihood of producing an output spike is inversely proportional to the information that would be lost if the spike were generated prematurely.

To determine the missing input, the pre-synaptic neuron j signals to its target neuron i whether the sign of its present

[1] We assume the ANN uses the common rectifying linear unit activation function $\text{relu}(x) = \max(0, x)$.

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

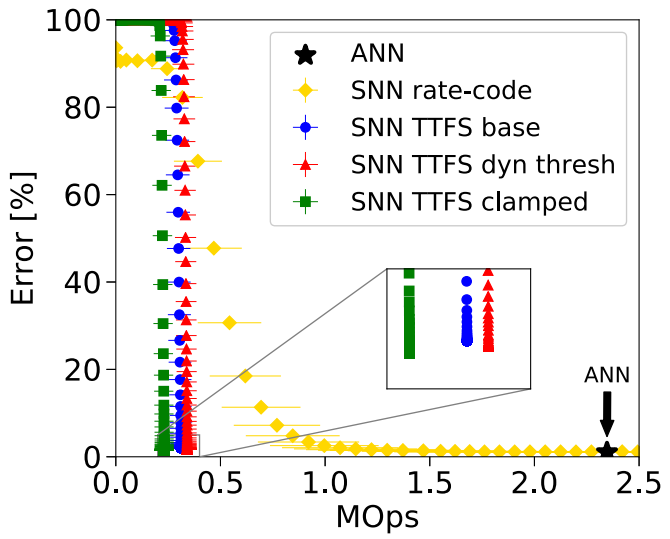

Figure 2. Classification error versus operation count of the original LeNet-5 ANN and the converted SNNs tested on MNIST. Error bars are removed from inset for clarity.

PSP $x_j(t)$ is positive. If the net PSP of a neuron is excitatory (it expects to fire a spike in the future), then the post-synaptic neuron i updates its threshold according to the following rule:

$$\theta_i(t) = \theta_i(\infty) + \sum_{j \in \Gamma_i} |w_{ij}| \mathcal{H}(x_j(t)). \tag{6}$$

In essence, each neuron fires two types of spikes: The first type signals whether to expect a spike in the future, and the second type is the actual output spike. We label this method "TTFS dyn thresh" in the remainder of the paper.

C. TTFS with clamped ReLU ("TTFS clamped")

Because the spike count in "TTFS dyn thresh" is at least twice as high as in "TTFS base", we propose a second approach to mitigate the effect of long-latency spikes. The original ANN is refined with a modified ReLU activation function, where the activation values below some threshold $\beta > 0$ are clamped to zero:

$$\mathrm{relu}_{\mathrm{clamp}}(x) = \begin{cases} 0 & \text{if } x \leq \beta \\ x & \text{else.} \end{cases} \tag{7}$$

This way, the network learns to perform well without relying on low activations, and neurons in the converted SNN do not have to wait for long-latency spikes. A neat side effect is that sparsity in feature maps is increased, further reducing the spike count in the SNN. An obvious limitation of this method is the need to retrain with a constraint in the form of the clamped ReLU (7). This refinement is trivial in case of small problems like MNIST, and may even show additional benefits accompanying regularizations, but can become cumbersome for larger networks and datasets. This method is labeled "TTFS clamped" in the remainder of the paper.

Table I. COMPARISON WITH OTHER SNNS ON MNIST.

	err [%]			
Method	ANN	SNN	# spikes	# neurons
Rate-based conversion [23]	1.40	1.52	10^4	3194
	0.90	0.88	10^5	5194
Rate-based conversion [19]	1.16	1.16	200	3194
	0.86	0.86	100	5194
STDP, TTFS [11]	NA	1.60	600	9144
STDP [20]	NA	5.0	17	6400
Supervised TTFS learning [22]	NA	3.02	135	1384
TTFS fully-conn. [this work]	1.50	1.65	100	1394
TTFS convol. [this work]	1.04	1.43	1000	7614

III. RESULTS

We tested the three versions of the time-to-first-spike approach ("TTFS base", "TTFS dyn thresh", "TTFS clamped") on the MNIST handwritten digit recognition data set and using the classic Lenet-5 [24] model. The dataset consists of 70000 28x28 gray-scale images, of which 10000 were used for testing, the rest for training. The Lenet-5 model contains three convolution layers, two max-pooling layers and two fully-connected layers, amounting to a total of 7614 neurons and 1.2 million synapses. The network conversion and simulations were performed using the "SNN toolbox"[2]. The top-1 error of our LeNet-5 ANN is 1.04%; the classification of a single sample in one forward pass requires 2.35 million operations[3]. Inference in the SNN is not done in a single forward pass where all kernel operations and neuron updates are performed once, but instead consists of simulating the SNN until one of the 10 output neurons has spiked. Then the simulation is stopped. At each time step, we measure both the classification error and the cumulative number of spikes in the network. From the spike count we infer the total number of operations in the SNN: Each spike causes a number of synaptic operations equal to the number of post-synaptic neurons connected to it.

The curves of the mean classification error versus the number of operations for all 3 methods are shown in Fig. 2. We also display the curve obtained when converting the LeNet-5 ANN using a rate-code as in [13]. In all four SNNs, classification error is high initially while spikes propagate through the network and the membrane potential of output neurons is still below threshold. Once an output spike is produced, the classification error drops. This drop is remarkably steep in the TTFS encoded networks, indicating that such a network needs approximately the same number of operations to classify each sample. The error-curve of the rate-coded SNN drops more slowly over time because neurons get to fire multiple times, gradually improving the network output.

All three versions of the TTFS encoding scheme produce very similar results: Their error rates are all within 1% of that from the original ANN, while the number of operations is reduced by 7-10X. The "TTFS base" baseline version reduces operational cost by a factor of 7, but the classification error rate is almost double due to missing long-latency spikes as described in Sec. II. The advantage of this method is that we do not have to retrain the network just as for the "TTFS clamped" version, and we do not have to compute the threshold

[2]http://snntoolbox.readthedocs.io
[3]MACs are counted twice, for multiplication and addition.

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

Table II. LENET-5 PERFORMANCE ON MNIST.

Model	MOps	Err [%]
ANN	2.35	1.04
SNN rate code (44 sim. steps)	3.03	1.07
SNN rate code (18 sim. steps)	1.07	2.07
SNN TTFS base	0.31	2.00
SNN TTFS dyn thresh	0.34	1.80
SNN TTFS clamped	0.23	1.47

updates as in "TTFS dyn thresh". As explained in Sec. II-B, threshold updates require transmission of a flag, indicating whether the post-synaptic neuron should expect a spike from the pre-synaptic neuron. The spike count of "TTFS dyn thresh" shown in Fig. 2 (not counting threshold updates) is higher than "TTFS base" because output spike generation is delayed to take into account long-latency spikes. When including threshold updates, the operation count of "TTFS dyn thresh" is 3X larger than the base-line, but still 2X lower than the ANN. The "TTFS clamped" approach achieves the lowest error rate at the lowest operational cost during inference, because the clamped ReLU increases sparsity in activations. This favorable number of operations does not take into account the one-time computational overhead when refining the ANN.

In order to compare our temporal coding method with the one presented in [22], we trained an ANN using the same architecture as in [22], namely a fully connected network with 784 input neurons, 600 hidden neurons, and 10 output neurons. The error rate of this ANN on MNIST is 1.50% and the operational cost is 953 kOps. Using the "TTFS base" method, the converted SNN achieves 1.65% error rate at 0.59 kOps. This computational cost is roughly equivalent to a network average of 100 spikes per sample. The method proposed in [22] does not convert an existing ANN, but trains the SNN directly with back-propagation, using an expression for the time-to-first-spike similar to Eq. (5). The gray-scale images were binarized and the bit-precisions of the network parameters and activations were reduced so as to facilitate the implementation on an FPGA. The cost function contains a penalty term that encourages neurons to fire early. Their resulting SNN achieves a 3.02% error rate, at a reported average activity of 135 spikes per sample.

IV. DISCUSSION

This work presents methods to convert analog neural networks into spiking neural networks using a temporal coding scheme that significantly reduces the spike redundancy present in previous rate-based conversion methods. In our proposed implementation, the activation value of neurons in the ANN is encoded as time-to-first-spike in the converted SNN. Thus, every neuron fires at most once, if receiving a positive net input, but does not spike if receiving zero or negative input. This encoding is sparse in both time and feature space, making the model suitable for low-power embedded applications.

The baseline TTFS method does not require modification of the network architecture or model parameters, and achieves a reduction in operation cost of 7X using LeNet-5 on MNIST. The classification error rate however increases by 1 percentage point. This increase can be mitigated by two proposed variants of the baseline method. The first variant consists of the refinement of the original ANN before conversion, by using a modified ReLU activation function. The second variant employs dynamic thresholds in the SNN to take into account the outputs of low-activated neurons. Both variants produce error rates close to the ANN error rate. The cost associated with this improvement is the need for retraining before inference, or for threshold updates during inference.

The SNNs obtained with the proposed TTFS code compare favorably against previous SNN results on MNIST (Tab. I), obtained either by conversion or direct training of the SNN. The rate-based conversion of [23] is superior in terms of error rate but requires significantly more spikes. The rate-based conversion by [19] is likewise more accurate, and features low spike rates due to its threshold adaptation. However, the spike rate alone does not account for the hidden computations due to threshold updates and processing of the real-valued spikes. Methods for training SNNs directly score in terms of low spike rates, but either require a higher number of neurons or converge to a higher error rate. We wish to emphasize the limited value of comparing conversion and training methods: Converting a pretrained ANN gives the SNN a "head-start" in classification performance; training the SNN directly has the advantage that the model can be taught to cope with artifacts that would occur during conversion, for instance the long-latency spikes discussed in Sec. II.

SNNs potentially run more efficiently than standard ANNs for two reasons: (1) SNNs use additions instead of MACs, which consume about 14X less energy and occupy 21X less area[4]. (2) Zeros in feature maps are automatically skipped in the forward-pass of the SNN. On the other hand, SNN operation incurs additional memory traffic due to repeated updating of neuron states. The cost of a memory transfer can exceed the cost of a floating-point multiplication operation by two orders of magnitude [15]. TTFS-encoded SNNs fire at most one spike per neuron, thereby minimizing the energy cost in memory access.

Future research will be directed towards hardware implementation of this time-to-first-spike code, as well as the application on larger models and more challenging datasets. The massive reduction of spike counts in TTFS-encoded SNNs makes this model attractive for neuromorphic hardware platforms where energy costs are dominated by spike-induced memory fetches and spike routing.

ACKNOWLEDGMENT

This work has been supported by the Samsung Advanced Institute of Technology and the University of Zurich.

REFERENCES

[1] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," *Neural Networks*, vol. 61, pp. 85–117, 2014. [Online]. Available: http://arxiv.org/abs/1404.7828http://dx.doi.org/10.1016/j.neunet.2014.09.003

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84930630277&partnerID=40&md5=befeefa64ddca265c713cf81f4e2fc54

[4] Simulated for 32-bit floating-point in a Global Foundry 28 nm process.

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

[3] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," *arXiv:1602.02830*, 2016. [Online]. Available: http://arxiv.org/abs/1602.02830

[4] A. Aimar, H. Mostafa, E. Calabrese, A. Rios-Navarro, R. Tapiador-Morales, I.-A. Lungu, M. B. Milde, F. Corradi, A. Linares-Barranco, G. Indiveri, S.-C. Liu, and T. Delbruck, "NullHop: A Flexible Convolutional Neural Network Accelerator Based on Sparse Representations of Feature Maps," *arXiv: 1706.01406*, 2017.

[5] Y. H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *Proceedings - 2016 43rd International Symposium on Computer Architecture, ISCA 2016*, 2016, pp. 367–379.

[6] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128 x 128 120dB 15us Latency Asynchronous Temporal Contrast Vision Sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, 2008.

[7] 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 spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014. [Online]. Available: http://www.sciencemag.org/cgi/doi/10.1126/science.1254642

[8] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.

[9] H. Mostafa, "Supervised learning based on temporal coding in spiking neural networks," *arXiv:1606.08165*, 2016. [Online]. Available: http://arxiv.org/abs/1606.08165

[10] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training Deep Spiking Neural Networks using Backpropagation," *Frontiers in Neuroscience*, vol. 10, no. NOV, pp. 1–10, 2016.

[11] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, "STDP-based spiking deep neural networks for object recognition," *arXiv*, 2016. [Online]. Available: http://arxiv.org/abs/1611.01421

[12] B. Rueckauer, I.-A. Lungu, Y. Hu, and M. Pfeiffer, "Theory and Tools for the Conversion of Analog to Spiking Convolutional Neural Networks," *arXiv:1612.04052*, 2016. [Online]. Available: http://arxiv.org/abs/1612.04052

[13] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification," *Frontiers in Neuroscience*, vol. 11, no. December, pp. 1–12, 2017. [Online]. Available: http://journal.frontiersin.org/article/10.3389/fnins.2017.00682/full

[14] E. Hunsberger and C. Eliasmith, "Training Spiking Deep Networks for Neuromorphic Hardware," *arXiv:1611.05141*, 2016. [Online]. Available: http://arxiv.org/abs/1611.05141%0Ahttp://dx.doi.org/10.13140/RG.2.2.10967.06566

[15] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *Digest of Technical Papers - IEEE International Solid-State Circuits Conference*, vol. 57, San Francisco, CA, 2014, pp. 10–14.

[16] R. VanRullen and S. J. Thorpe, "Rate coding versus temporal order coding: what the retinal ganglion cells tell the visual cortex." *Neural computation*, vol. 13, no. 6, pp. 1255–83, 2001. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/11387046

[17] S. Thorpe, A. Delorme, and R. Van Rullen, "Spike-based strategies for rapid processing," *Neural Networks*, vol. 14, no. 6-7, pp. 715–725, 2001.

[18] G. Orchard, C. Meyer, R. Etienne-Cummings, C. Posch, N. Thakor, and R. Benosman, "HFirst: A Temporal Approach to Object Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 10, pp. 2028–2040, 10 2015.

[19] D. Zambrano and S. M. Bohte, "Fast and Efficient Asynchronous Neural Computation with Adapting Spiking Neural Networks," *ArXiv:1609.02053*, 2016. [Online]. Available: http://arxiv.org/abs/1609.02053

[20] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in Computational Neuroscience*, vol. 9, no. August, pp. 1–9, 2015. [Online]. Available: http://journal.frontiersin.org/Article/10.3389/fncom.2015.00099/abstract

[21] B. Zhao, R. Ding, S. Chen, B. Linares-Barranco, and H. Tang, "Feedforward Categorization on AER Motion Events Using Cortex-Like Features in a Spiking Neural Network," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 9, pp. 1963–1978, 2015.

[22] H. Mostafa, B. U. Pedroni, S. Sheik, and G. Cauwenberghs, "Fast Classification Using Sparsely Active Spiking Networks," in *ISCAS*, 2017.

[23] P. U. Diehl, D. Neil, J. Binas, M. Cook, S. C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *Proceedings of the International Joint Conference on Neural Networks*, vol. 2015-Septe, Killarney, Ireland, 2015.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, no. 11. IEEE, 1998, pp. 2278–2323.

978-1-5386-4881-0/18/$31.00 ©2018 IEEE