arXiv:1606.07786v1 [cs.NE] 23 Jun 2016

Precise deep neural network computation on imprecise low-power analog hardware

Jonathan Binas*, Daniel Neil, Giacomo Indiveri, Shih-Chii Liu, Michael Pfeiffer

Institute of Neuroinformatics, University of Zurich and ETH Zurich,
Winterthurerstr. 190, 8057 Zurich, Switzerland

*Email: jbinas@ini.ethz.ch.

Abstract

There is an urgent need for compact, fast, and power-efficient hardware implementations of state-of-the-art artificial intelligence. Here we propose a power-efficient approach for real-time inference, in which deep neural networks (DNNs) are implemented through low-power analog circuits. Although analog implementations can be extremely compact, they have been largely supplanted by digital designs, partly because of device mismatch effects due to fabrication. We propose a framework that exploits the power of Deep Learning to compensate for this mismatch by incorporating the measured variations of the devices as constraints in the DNN training process. This eliminates the use of mismatch minimization strategies such as the use of very large transistors, and allows circuit complexity and power-consumption to be reduced to a minimum. Our results, based on large-scale simulations as well as a prototype VLSI chip implementation indicate at least a 3-fold improvement of processing efficiency over current digital implementations.

Modern information technology requires increasing computational power to process massive amounts of data in real time. This rapidly growing need for computing power has led to the exploration of computing technologies beyond the predominant von Neumann architecture. In particular, due to the separation of memory and processing elements, traditional computing systems experience a bottleneck when dealing with problems involving great amounts of high-dimensional data [4, 25], such as image processing, object recognition, probabilistic inference, or speech recognition. These problems can often best be tackled by conceptually simple but powerful and highly parallel methods, such as deep neural networks (DNNs), which in recent years have delivered state-of-the-art performance on exactly those applications [29, 47]. DNNs are characterized by stereotypical and simple operations at each unit, of which many can be performed in parallel. For this reason they map favorably e.g. onto the processing style of graphics

processing units (GPUs) [46]. The large computational demands of DNNs have simultaneously sparked interest in methods that make neural network inference faster and more power efficient, whether through new algorithmic inventions [19, 21, 11], dedicated digital hardware implementations [6, 17, 8], or by taking inspiration from real nervous systems [14, 37, 33, 23, 34].

With synchronous digital logic being the established standard of the electronics industry, first attempts towards hardware deep network accelerators have focused on this approach [6, 18, 7, 38]. However, the massively parallel style of computation of neural networks is not reflected in the mostly serial and time-multiplexed nature of digital systems. An arguably more natural way of developing a hardware neural network emulator is to implement its computational primitives as multiple physical and parallel instances of analog computing nodes, where memory and processing elements are co-localized, and state variables are directly represented by analog currents or voltages, rather than being encoded digitally [43, 1, 49, 5, 3, 45]. By directly representing neural network operations in the physical properties of silicon transistors, such analog implementations can outshine their digital counterparts in terms of simplicity, allowing for significant advances in speed, size, and power consumption [20, 32]. The main reason why engineers have been discouraged from following this approach is that the properties of analog circuits are affected by the physical imperfections inherent to any chip fabrication process, which can lead to significant functional differences between individual devices [40].

In this work we propose a new approach, whereby rather than brute-force engineering more homogeneous circuits (e.g. by increasing transistor sizes and burning more power), we employ neural network training methods as an effective optimization framework to automatically compensate for the device mismatch effects of analog VLSI circuits. We use the diverse measured characteristics of individual VLSI devices as constraints in an off-line training process, to yield network configurations that are tailored to the particular analog device used, thereby compensating the inherent variability of chip fabrication. Finally, the network parameters, in particular the synaptic weights found during the training phase can be programmed in the network, and the analog circuits can be operated at run-time in the sub-threshold region for significantly lower power consumption.

In this article, in addition to introducing a novel training method for both device and network, we also propose compact and low-power candidate VLSI circuits. A closed-loop demonstration of the framework is shown, based on a fabricated prototype chip, as well as detailed, large-scale simulations. The resulting analog electronic neural network performs as well as an ideal network, while offering at least a threefold lower power consumption over its digital counterpart.

1 Results

A deep neural network processes input signals in a number of successive layers of neurons, where each neuron computes a weighted sum of its inputs followed by a non-linearity, such as a sigmoid or rectification. Specifically, the output of a neuron i is given by $x_i = f\left(\sum_j w_{ij} x_j\right)$,

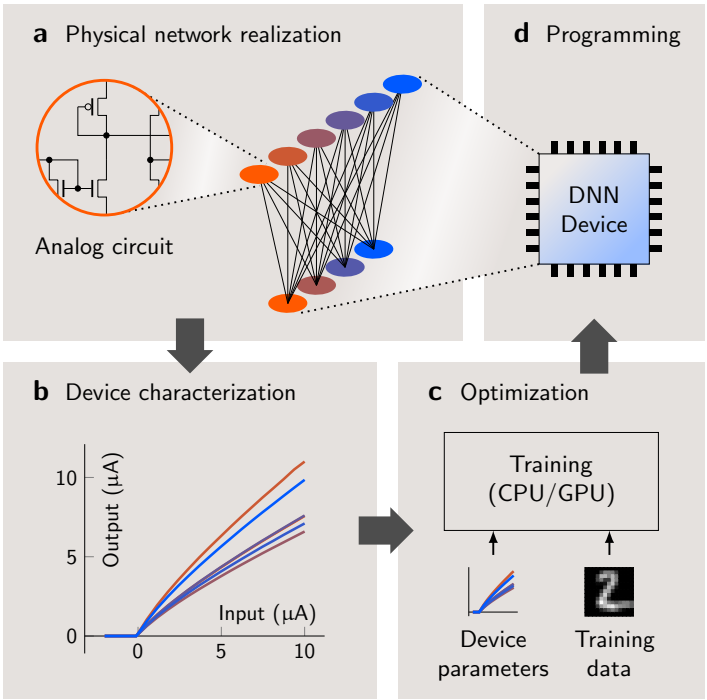

Figure 1: Implementing and training analog electronic neural networks. a) The configurable network is realized on a physical substrate by means of analog circuits, together with local memory elements that store the weight configuration. b) The transfer characteristics of individual neurons are measured by applying specific stimuli to the input layer and simultaneously recording the output of the network. Repeating these measurements for different weight configurations and input patterns allows to reconstruct the individual transfer curves and fit them by a model to be used for training. c) Including the measured transfer characteristics in the training process allows optimization of the network for the particular device that has been measured. d) Mapping the parameters found by the training algorithm back to the device implements a neural network, whose computation is comparable to the theoretically ideal network. Arrows indicate the sequence of steps taken as well as the flow of measurement/programming data.

where f is the non-linearity, and w_{ij} is the weight of the connection from neuron j to neuron i. Thus, the basic operations comprising a neural network are summation, multiplication by scalars, and simple non-linear transformations. All of these operations can be implemented in analog electronic circuitry very efficiently, that is with very few transistors, whereby numeric values are represented by actual voltage or current values, rather than a digital code. Analog circuits are affected by fabrication mismatch, i.e. small fluctuations in the fabrication process that lead to fixed distortions of functional properties of elements on the same device, as well as multiple sources of noise. As a consequence, the response of an analog hardware neuron is slightly different for every instance of the circuit, such that $x_i = \hat{f}_i\left(\sum_j w_{ij}x_j\right)$, where $\hat{f}_i$ approximately corresponds to f, but is slightly different for every neuron i.

1.1 Training with heterogeneous transfer functions

The weights of multi-layered networks are typically learned from labeled training data using the backpropagation algorithm [44], which minimizes the training error by computing error gradients and passing them backwards through the layers. In order for this to work in practice, the transfer function f needs to be at least piece-wise differentiable, as is the case for the commonly used rectified linear unit (ReLU) [16]. Although it is common practice in neural network training, it is not necessary for all neurons to have identical activation functions f. In fact, having different activation functions makes no difference to backpropagation as long as their derivatives can be computed. Here this principle is exploited by inserting the heterogeneous but measured transfer curves $\hat{f}_i$ from a physical analog neural network implementation into the training algorithm, with the goal of finding weight parameters that are tailored for a particular heterogeneous system given by $\hat{f}_1, \ldots, \hat{f}_N$.

The process of implementing a target functionality in such a heterogeneous system is illustrated in Fig. 1. Once a neural network architecture with modifiable weights is implemented in silicon, the transfer characteristics of the different neuron instances can be measured by controlling the inputs specific cells receive and recording their output at the same time (see Methods). If the transfer curves are sufficiently simple (depending on the actual implemented analog neuron circuit), a small number of discrete measurements yield sufficient information to fit a continuous, (piece-wise) differentiable model to the hardware response. For instance, the rectified linear neuron $f(r) = \max\{0, a \cdot r\}$ is fully described by a single parameter a, which is simply the ratio of output to input, and therefore can easily be measured. The continuous, parameterized description is then used by the training algorithm, which is run on traditional computing hardware, such as CPUs or GPUs, to generate a network configuration that is tailored to the particular task and the physical device that has been characterized.

1.2 Analog circuit implementation

To achieve a compact and low-power solution, we construct a multilayer network using the circuits shown in Fig. 2 and operate them in the subthreshold region. The subthreshold current

of a transistor is exponential in the gate voltage, rather than polynomial as is the case for above threshold operation, and can span many orders of magnitude. Thus, a system based on this technology can be operated at orders of magnitude lower currents than a digital one. In turn, this means that the device mismatch arising due to imperfections in the fabrication process can have an exponentially larger impact. Fortunately, as our method neither depends on the specific form nor the magnitude of the mismatch, it can handle a wide variety of mismatch conditions.

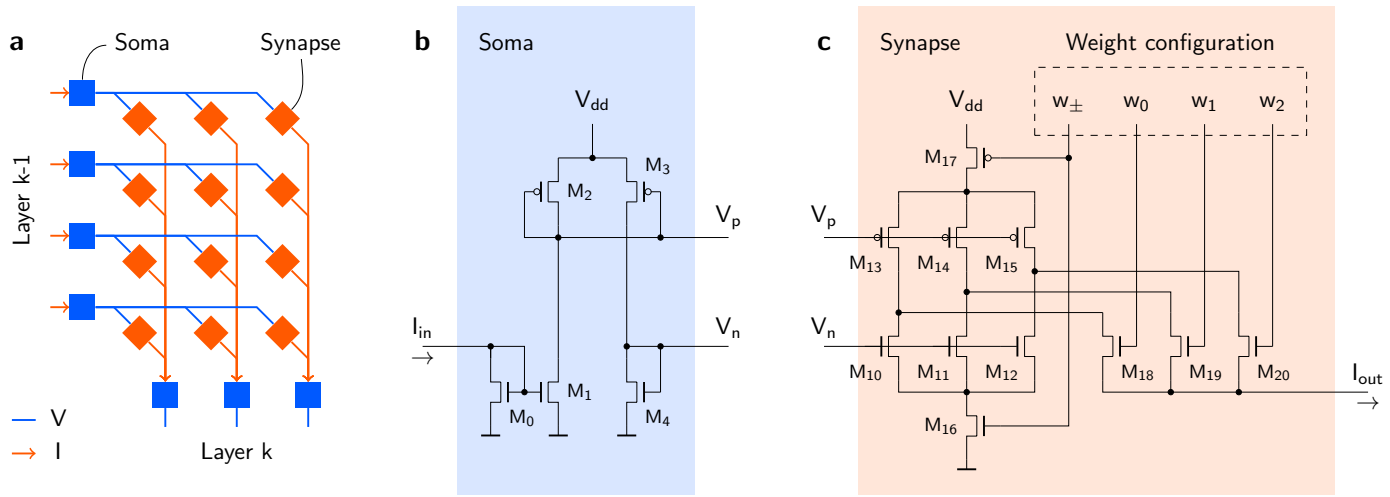

Figure 2: A multi-layer neural network implemented with current-mode analog circuits. a) A network is constructed by connecting layers of soma circuits through matrices of synapse circuits. The output of a soma circuit is communicated as a voltage (blue) and passed to a row of synapse circuits, implementing multiplications by scalars. The output of a synapse is a current (orange), such that the outputs of a column of synapses can be summed up by simply connecting them through wires. The summed current is then passed as input to a soma of the next layer, which implements the non-linearity. b) Proposed soma circuit, taking a current as input and providing two output voltages V_n and V_p, which in the subthreshold region are proportional to the log-transformed, rectified input current. c) Proposed programmable synapse circuit with 3 bit precision, taking voltages V_n and V_p as inputs and providing an output current corresponding to an amplified version of the rectified soma input current, where the gain is set by the digital signals $w_\pm$, w_i.

As a demonstration of our framework, a feed-forward network is implemented in which every neuron consists of one soma and multiple synapse circuits, then train it for different classification tasks. As illustrated in Fig. 2a, multiple layers of soma circuits are connected through matrices of synapse circuits. A soma circuit (Fig. 2b) takes a current as input and communicates its output in terms of voltages, which are passed as input signals to a row of synapse circuits. A synapse circuit (Fig. 2c), in turn, provides a current as output, such that the outputs of a column of synapses can be summed up simply by connecting them together. The resulting current is then fed as an input current to the somata of the next layer. The first transistor

of the soma circuit rectifies the input current. The remaining elements of the soma circuit, together with a connected synapse circuit, form a set of scaling current mirrors, i.e. rudimentary amplifiers, a subset of which can be switched on or off to achieve a particular weight value by setting the respective synapse configuration bits. Thus, the output of a synapse corresponds to a scaled version of the rectified input current of the soma, similar to the ReLU transfer function. In our proposed example implementation we use signed 3-bit synapses, which are based on 2×3 current mirrors of different dimensions (3 for positive and 3 for negative values). One of 2^4 possible weight values is then selected by switching the respective current mirrors on or off. The scaling factor of a particular current mirror, and thus its contribution to the total weight value, is proportional to the ratio of the widths of the two transistors forming it. The weight configuration of an individual synapse can be stored digitally in memory elements that are part of the actual synapse circuit. Thus, in contrast to digital processing systems, our circuit computes *in memory* and thereby avoids the bottleneck of expensive data transfer between memory and processing elements.

Although this is just one out of many possible analog circuits implementations, the simple circuits chosen offer several advantages besides the fact that they can be implemented in small areas: First, numeric values are conveyed only through current mirrors, and therefore are temperature-independent. Second, most of the fabrication-induced variability is due to the devices in the soma with five consecutive transistors, whereas only one layer of transistors affects the signal in the synapse. This means that the synapse-induced mismatch can be neglected in a first order approximation.

Once an analog electronic neural network has been implemented physically as a VLSI device, the transfer characteristics of the individual soma circuits are obtained through measurements. The transfer function implemented by our circuits can be well described by a rectified linear curve, where the only free parameter is the slope, and thus can be determined from a single measurement per neuron. Specifically, the transfer curves of all neurons in a layer k can be measured through a simple procedure: A single neuron in layer $k-1$ is connected, potentially through some intermediate neurons, to the input layer and is defined to be the 'source'. Similarly, a neuron in layer $k+1$ is connected, potentially through intermediate neurons, to the output layer and is called the 'monitor'. All neurons of layer k can now be probed individually using the source and monitor neurons, whereby the signal to the input layer is held fixed and the signal recorded at the output layer is proportional to the slope of the measured neuron. Note that the absolute scale of the responses is not relevant, i.e. only the relative scale within one layer matters, as the output of individual layers can be scaled arbitrarily without altering the network function. The same procedure can be applied to all layers to obtain a complete characterization of the network. The measurements can be parallelized by defining multiple source and monitor neurons per measurement to probe several neurons in one layer simultaneously, or by introducing additional readout circuitry between layers to measure multiple layers simultaneously.

1.3 Handwritten and spoken digit classification

Large-scale SPICE simulations of systems consisting of hundreds of thousands of transistors are employed to assess power consumption, processing speed, and the accuracy of such an analog implementation.

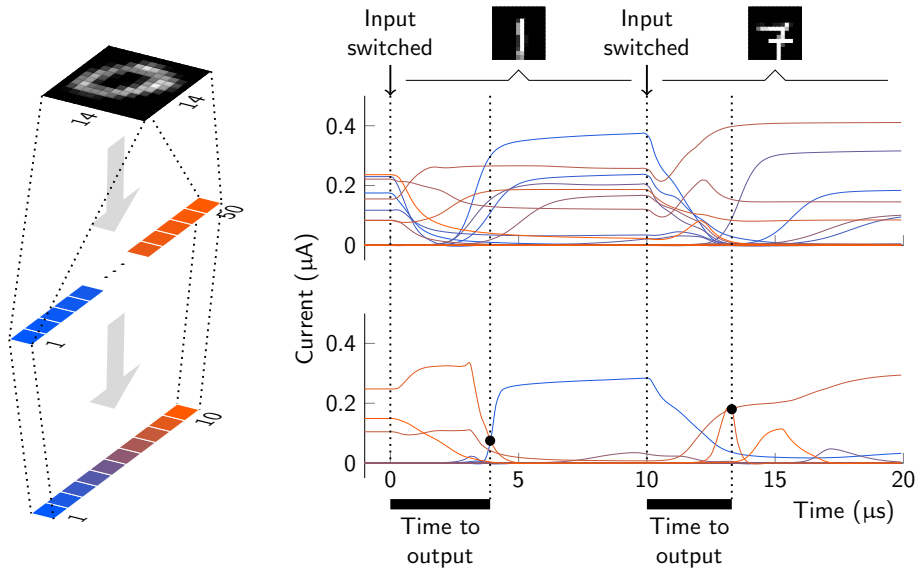

Figure 3: Analog circuit dynamics allow classification within microseconds. The curves represent the activities (currents) of all hidden (top) and output (bottom) units of the $196 - 50 - 10$ network shown on the left. When a new input symbol is presented (top), the circuit converges to its new state within microseconds. Only a few units remain active, while many tend to zero, such that their soma circuits and connected synapses dissipate very little power.

After simulating measurements and parameterizing the transfer characteristics of the circuits as described previously, software networks were trained on the MNIST dataset of handwritten digits [30] and the TIDIGITS dataset of spoken digits [31] by means of the ADAM training method [27]. In order to optimize the network for the use of discrete weights in the synaptic circuits dual-copy rounding [48, 10] was used (see Methods). By evaluating the responses of the simulated circuit on subsets of the respective test sets, its classification accuracy was found to be comparable to the abstract software neural network (see Tab. 1 for comparison). Fig. 3 shows how inputs are processed by a small example circuit implementing a $196 - 50 - 10$ network, containing around 10k synapses and over 100k transistors. Starting with the presentation of an input pattern in the top layer, where currents are proportional to input stimulus intensity, the higher layers react almost instantaneously and provide the correct classification, i.e. the index of the maximally active output unit, within a few microseconds. After a switch of input patterns, the signals quickly propagate through the network and the outputs of different nodes converge to their asymptotic values. The time it takes the circuit to converge to its final output defines the

'time to output', constraining the maximum frequency at which input patterns can be presented and evaluated correctly. Measured convergence times are summarized in Fig. 4 for different patterns from the MNIST test set, and are found to be in the range of microseconds for a trained $196 - 100 - 50 - 10$ network, containing over 25k synapses and around 280k transistors. Note that observed timescale is not fixed as the network can be run faster or slower by changing the input current, while the average energy dissipated per operation remains roughly constant.

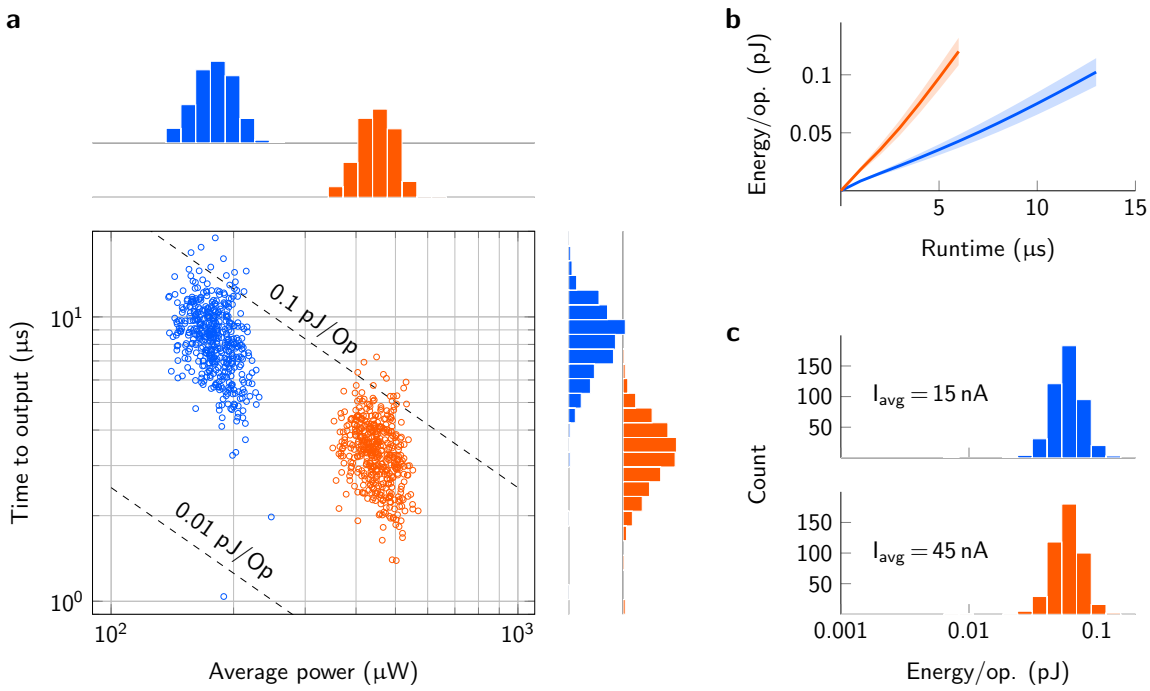

Figure 4: Processing performance of a network for handwritten digit classification. All data shown was generated by presenting 500 different input patterns from the MNIST test set to a trained $196 - 100 - 50 - 10$ network with the average input current per input neuron set to 15 nA (blue) or 45 nA (orange), respectively. a) The time to output is plotted against the average power dissipated over the duration of the transient (from start of the input pattern until time to output). The distributions of the data points are indicated by histograms on the sides. Changing the input current causes a shift along the equi-efficiency lines, that is, the network can be run slower or faster at the same efficiency (energy per operation). b) Energy dissipated per operation for different run times, corresponding to different fixed rates at which inputs are presented (mean over 500 samples; standard deviation indicated by shaded areas). c) The average energy consumed per operation was computed from the data shown in a). The data corresponds to the hypothetical case were the network would be stopped as soon as the correct output is reached.

The processing efficiency of the system (energy per operation) was computed for different input patterns by integrating the power dissipated between the time at which the input pattern

was switched and the time to output. Fig. 4 shows the processing efficiency for the same network with different input examples and under different operating currents. With the average input currents scaled to either 15 or 45 nA per neuron respectively, the network takes several microseconds to converge and consumes tens or hundreds of microwatts in total, which amounts to a few nanowatts per multiply-accumulate operation. With the supply voltage set to 1.8 V, this corresponds to less than 0.1 pJ per operation in most cases. With the average input current set to 15 nA per neuron, the network produces the correct output within 15 µs in over 99 % of all cases (mean 8.5 µs; std. 2.3 µs). Running the circuit for 15 µs requires 0.12 ± 0.01 pJ per operation, such that about 1.7 trillion multiply-accumulate operations can be computed per second at a power budget of around 200 µW if input patterns are presented at a rate of 66 kHz. Without major optimizations to either process or implementation, this leads to an efficiency of around 8 TOp/J, to our knowledge a performance at least four times greater than that achieved by digital single-purpose neural network accelerators in similar scenarios [6, 38]. General purpose digital systems are far behind such specialized systems in terms of efficiency, with the latest GPU generation achieving around 0.05 TOp/J [36].

Tab. 1 summarizes the classification accuracy for different architectures and datasets for a software simulation of an ideal network without mismatch, a behavioral simulation of the inhomogeneous system with the parameterized transfer curves implemented in an abstract software model, and the full circuit simulation of the inhomogeneous hardware network. Additionally, the computed power efficiency is shown for the different architectures.

Table 1: Classification accuracy and power-efficiency of a $196-100-50-10$ network trained on the MNIST and TIDIGITS datasets. The classification accuracies of the behavioral models of the ideal as well as the inhomogeneous systems are averaged over 10 networks trained with different initializations. The parameters of the best performing one out of the 10 networks were used in the SPICE circuit simulations. As detailed circuit simulations are computationally expensive, subsets of the actual test sets were used to compute the classification accuracy of the simulated circuits (the first 500 samples from the MNIST test set; 500 random samples from the TIDIGITS test set).

	14×14 MNIST	TIDIGITS
Homogeneous model mean / best accuracy (%)	97.6 ± 0.1 / 98.0	87.3 ± 4.2 / 93.4
Inhomogeneous model mean / best accuracy (%)	97.6 ± 0.2 / 98.0	88.0 ± 3.8 / 94.3
SPICE simulation accuracy (%)	98.0	94.6
Energy-efficiency (TOp/J)	7.97	6.39

1.4 VLSI implementation

As a closed-loop demonstration of our framework, we designed a prototype VLSI chip and trained it for a classification task. A design based on the circuits shown in Fig. 2, containing

three layers of seven neurons each, was fabricated in 180 nm CMOS technology. After characterizing the individual neuron circuits through measurements as described in Sect. 1.2 we trained a $4-7-3$ network on 80 % of the Iris flower dataset [15], programmed the device with the found parameters, and used the remaining 20 % of the data to test the classification performance. The hardware implementation was able to classify 100% of the test data correctly (see Fig. 5e for the output of the network).

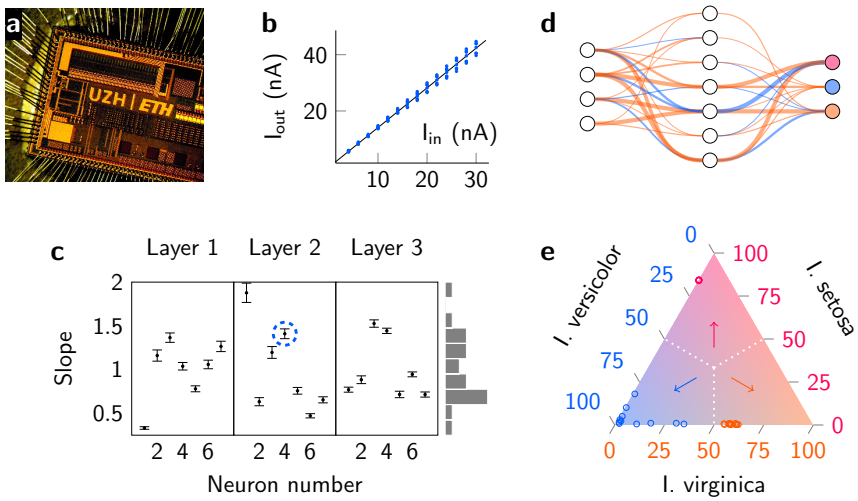

Figure 5: Running a classification task on the prototype VLSI implementation. a) Photograph of the fabricated device. The neural network is a small block at the center of the chip. b) Measurements of a single neuron (blue; corresponding to the marked point in c)) and the line fitted to the measurements (black). c) Measured slopes of all 3×7 neurons of the prototype device (means and standard deviations; slopes normalized per layer). d) Visualization of the $4-7-3$ network which was implemented and trained on the Iris flower dataset (positive weights are displayed in orange, negative ones in blue; line thickness corresponds to weight value). e) Correct classification of the test set performed by the programmed chip (responses of the three output neurons normalized to 100 %, displayed in barycentric coordinates; dot color represents the target class).

2 Discussion

The theory of analog neural networks and electronic realizations thereof have a substantial history that goes back to the1950s [43, 1]. However, the demonstrated accuracy of the electronic networks is typically below the theoretical performance and therefore, their full low-power potential was never fully leveraged.

Instead, digital designs have flourished in the interim and almost all current deep network designs are implemented in digital form [6, 7, 38]. Although small transistors are possible

in digital implementations, the typical size of a multiplier-accumulator (MAC) block usually means that these implementations use a smaller subset of functional blocks and therefore the use of MACs is time-multiplexed by shifting data around accordingly. As a consequence, the processing speed of digital implementations is limited by their clock frequency.

The simplicity of the analog VLSI circuits needed for addition - namely connecting together wires - allows an explicit implementation of each processing unit or neuron where no element is shared or time-multiplexed within the network implementation. The resulting VLSI network is maximally parallel and eliminates the bottleneck of transferring data between memory and processing elements. Using digital technology, such fully parallel implementations would quickly become prohibitively large due to the much greater circuit complexity of digital processing elements. While the focus in this work has been on an efficient analog VLSI implementation, hardware implementations using new forms of nano devices can also benefit from this training method. For example, the memristive computing technology which is currently being pursued for implementing large-scale cognitive neuromorphic and other technologies still suffers from the mismatch of fabricated devices [2, 26, 41]. The proposed training method in this work can be used to account for device non-idealities in this technology [35].

In fact, any system that can be properly characterized and has configurable elements stands to benefit from this approach. For example, spike-based neuromorphic systems [24] often have configurable weights between neurons. These systems communicate via biologically inspired digital-like pulses called spikes. Similar to the method outlined in this work, the relationship between an input spike rate and an output spike rate of a neuron can be measured in such a system, and the transfer functions then used as a constraint during the training process so as to achieve accurate results from the whole network even if the neuron circuits themselves are varied and non-ideal. In addition to the alternate hardware implementations, other network topologies such as convolutional networks can be trained using this proposed method. However, as all weights are implemented explicitly in silicon, the system design here would not benefit from the small memory footprint achieved via weight sharing in traditional convolutional network implementations. In principle, even recurrent architectures such as LSTM networks [22] can be trained using the same methods, where not only the static properties of the circuit are taken into account but also their dynamics.

With every device requiring an individual training procedure, an open question is how the per-device training time can be reduced. Initializing the network to a pre-trained ideal network, which is then fine-tuned for the particular devices is likely to reduce training time.

In the current setting, the efficiency of our system is limited by the worst-case per-example runtime, i.e. there may be a few samples where outputs require significantly longer to converge to the correct classification result than the majority. This can lead to unnecessarily long presentation times for many samples, thereby causing unnecessary power consumption. Smart methods of estimating presentation times from the input data could e.g. accelerate convergence for slowly converging samples by using higher input currents, and conversely, faster samples could be slowed down to lower the variability of convergence times and overall reduce energy consumption. Future research will focus on such estimators, and alternatively explore ways of

reducing convergence time variability during network training.

This proof-of-principle study is an important step towards the construction of large scale, possibly ultra-low-power analog VLSI deep neural network processors, paving the way for specialized applications which had not been feasible before due to speed or power constraints. Small, efficient implementations could allow autonomous systems to achieve almost immediate reaction times under strict power limitations. Scaled-up versions can allow for substantially more efficient processing in data centers, allowing for a greatly reduced energy footprint or permitting substantially more data to be effectively processed. Conversely, digital approaches and GPU technology are aiming for general purpose deep network acceleration, and thus naturally have an advantage in terms of flexibility compared to the fixed physical implementation of the proposed analog devices. However, there is increasing evidence that neural networks pre-trained on large datasets such as ImageNet provide excellent generic feature detectors [13, 42], which means that fast and efficient analog input pre-processors could be used as an important building blocks for a large variety of applications.

References

[1] J. Alspector and R.B. Allen. A neuromorphic VLSI learning system. In P. Losleben, editor, *Proceedings of the 1987 Stanford Conference on Advanced Research in VLSI*, pages 313–349, Cambridge, MA, USA, 1987. MIT Press.

[2] S Ambrogio, S Balatti, F Nardi, S Facchinetti, and D Ielmini. Spike-timing dependent plasticity in a transistor-selected resistive switching memory. *Nanotechnology*, 24(38):384012, 2013.

[3] Andreas G Andreou, Kwabena Boahen, Philippe O Pouliquen, Aleksandra Pavasovic, Robert E Jenkins, Kim Strohbehn, et al. Current-mode subthreshold MOS circuits for analog VLSI neural systems. *IEEE Transactions on neural networks*, 2(2):205–213, 1991.

[4] John Backus. Can programming be liberated from the von neumann style?: A functional style and its algebra of programs. *Commun. ACM*, 21(8):613–641, 1978.

[5] T.H. Borgstrom, M Ismail, and S.B. Bibyk. Programmable current-mode neural network for implementation in analogue MOS VLSI. *IEE Proceedings G*, 137(2):175–184, 1990.

[6] Lukas Cavigelli, David Gschwend, Christoph Mayer, Samuel Willi, Beat Muheim, and Luca Benini. Origami: A convolutional network accelerator. In *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, pages 199–204. ACM, 2015.

[7] Yu-Hsin Chen, Tushar Krishna, Joel Emer, and Vivienne Sze. 14.5 eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 262–263. IEEE, 2016.

[8] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. Dadiannao: A machine-learning supercomputer. In *Microarchitecture, 2014 47th Annual IEEE/ACM International Symposium on*, pages 609–622. IEEE, 2014.

[9] FranÃğois Chollet. Keras. https://github.com/fchollet/keras, 2015.

[10] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Low precision arithmetic for deep learning. *arXiv preprint arXiv:1412.7024*, 2014.

[11] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, pages 3105–3113, 2015.

[12] Tobi Delbruck, Raphael Berner, Patrick Lichtsteiner, and Carlos Dualibe. 32-bit configurable bias current generator with sub-off-current capability. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 1647–1650. IEEE, 2010.

[13] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. DeCAF: A deep convolutional activation feature for generic visual recognition. In *ICML*, pages 647–655, 2014.

[14] Clément Farabet, R Paz-Vicente, JA Pérez-Carrasco, Carlos Zamarreño-Ramos, Alejandro Linares-Barranco, Yann LeCun, Eugenio Culurciello, Teresa Serrano-Gotarredona, and Bernabe Linares-Barranco. Comparison between frame-constrained fix-pixel-value and frame-free spiking-dynamic-pixel convnets for visual processing. *Frontiers in Neuroscience*, 6:1–12, 2012.

[15] RA Fisher. Iris flower data set, 1936.

[16] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[17] Vinayak Gokhale, Jonghoon Jin, Aysegul Dundar, Ben Martini, and Eugenio Culurciello. A 240 g-ops/s mobile coprocessor for deep neural networks. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, pages 696–701. IEEE, 2014.

[18] Matthew Griffin, Gary Tahara, Kurt Knorpp, Ray Pinkham, and Bob Riley. An 11-million transistor neural network execution engine. In *Solid-State Circuits Conference, 1991. Digest of Technical Papers. 38th ISSCC., 1991 IEEE International*, pages 180–313. IEEE, 1991.

[19] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[20] Jennifer Hasler and Bo Marr. Finding a roadmap to achieve large neuromorphic hardware systems. *Frontiers in neuroscience*, 7, 2013.

[21] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[22] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[23] Giacomo Indiveri, Federico Corradi, and Ning Qiao. Neuromorphic architectures for spiking deep neural networks. In *IEEE International Electron Devices Meeting (IEDM)*, 2015.

[24] Giacomo Indiveri, Bernabe Linares-Barranco, Tara Julia Hamilton, André van Schaik, Ralph Etienne-Cummings, Tobi Delbruck, Shih-Chii Liu, Piotr Dudek, Philipp Häfliger, Sylvie Renaud, Johannes Schemmel, Gert Cauwenberghs, John Arthur, Kai Hynna, Fopefolu Folowosele, Sylvain SAÏGHI, Teresa Serrano-Gotarredona, Jayawan Wijekoon, Yingxue Wang, and Kwabena Boahen. Neuromorphic silicon neuron circuits. *Frontiers in Neuroscience*, 5(73), 2011.

[25] Giacomo Indiveri and Shih-Chii Liu. Memory and information processing in neuromorphic systems. *Proceedings of the IEEE*, 103(8):1379–1397, 2015.

[26] Kuk-Hwan Kim, Siddharth Gaba, Dana Wheeler, Jose M Cruz-Albrecht, Tahir Hussain, Narayan Srinivasa, and Wei Lu. A functional hybrid memristor crossbar-array/CMOS system for data storage and neuromorphic applications. *Nano letters*, 12(1):389–395, 2011.

[27] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[28] Kadaba R Lakshmikumar, Robert Hadaway, and Miles Copeland. Characterisation and modeling of mismatch in MOS transistors for precision analog design. *IEEE Journal of Solid-State Circuits*, 21(6):1057–1066, 1986.

[29] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[30] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The MNIST database of handwritten digits, 1998.

[31] R Gary Leonard and George Doddington. Tidigits speech corpus. *Texas Instruments, Inc*, 1993.

[32] Peter Masa, Klaas Hoen, and Hans Wallinga. A high-speed analog neural processor. *Micro, IEEE*, 14(3):40–50, 1994.

[33] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D Flickner, William P Risk, Rajit Manohar, and Dharmendra S Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[34] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. Learning to be efficient: Algorithms for training low-latency, low-compute deep spiking neural networks. In *ACM Symposium on Applied Computing*, 2016.

[35] Dimin Niu, Yiran Chen, Cong Xu, and Yuan Xie. Impact of process variations on emerging memristor. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 877–882. IEEE, 2010.

[36] NVIDIA. NVIDIA Tesla P100 – the most advanced datacenter accelerator ever built. featuring Pascal GP100, the world's fastest GPU. *NVIDIA Whitepaper*, 2016.

[37] Peter O'Connor, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. Real-time classification and sensor fusion with a spiking deep belief network. *Frontiers in Neuromorphic Engineering*, 7, 2013.

[38] SW Park, J Park, K Bong, D Shin, J Lee, S Choi, and HJ Yoo. An energy-efficient and scalable deep learning/inference processor with tetra-parallel MIMD architecture for big data applications. *IEEE transactions on biomedical circuits and systems*, 2016.

[39] Marcel JM Pelgrom, Hans P Tuinhout, and Maarten Vertregt. Transistor matching in analog CMOS applications. *IEDM Tech. Dig*, pages 915–918, 1998.

[40] M.J.M. Pelgrom, Aad C.J. Duinmaijer, and A.P.G. Welbers. Matching properties of MOS transistors. *IEEE Journal of Solid-State Circuits*, 24(5):1433–1439, Oct 1989.

[41] Mirko Prezioso, Farnood Merrikh-Bayat, BD Hoskins, GC Adam, Konstantin K Likharev, and Dmitri B Strukov. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature*, 521(7550):61–64, 2015.

[42] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 806–813, 2014.

[43] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386–408, nov 1958.

[44] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[45] S. Satyanarayana, Y.P. Tsividis, and H.P. Graf. A reconfigurable VLSI neural network. *IEEE Journal of Solid-State Circuits*, 27(1):67–81, Jan 1992.

[46] Dominik Scherer, Hannes Schulz, and Sven Behnke. Accelerating large-scale convolutional neural networks with parallel graphics multiprocessors. In *Artificial Neural Networks–ICANN 2010*, pages 82–91. Springer, 2010.

[47] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, January 2015.

[48] Evangelos Stromatias, Daniel Neil, Michael Pfeiffer, Francesco Galluppi, Steve B Furber, and Shih-Chii Liu. Robustness of spiking deep belief networks to noise and reduced bit precision of neuro-inspired hardware platforms. *Frontiers in neuroscience*, 9, 2015.

[49] E.A. Vittoz. Analog VLSI implementation of neural networks. In *Proc. IEEE Int. Symp. Circuit and Systems*, pages 2524–2527, New Orleans, 1990.

Acknowledgments: We are very grateful to Ning Qiao for his help with the VLSI implementation. The work was supported by the Swiss National Science Foundation grant SpikeComp (200021_146608) and the University of Zurich.

3 Methods

3.1 Description of the example circuit

The example networks described in Sect. 1.2 have been implemented based on the circuits shown in Fig. 2. With M_0 as a diode-connected nFET, the soma circuit essentially performs a rectification of the input current I_{in}. Further, the current is copied to M_1 and, through M_2 and M_3, also to M_4, such that M_2 together with pFETs from connected synapse circuits, as well as M_4 together with nFETs from connected synapse circuits form scaling current mirrors, generating scaled copies of the rectified input current I_{in}. The scaling factor is thereby determined by the dimensions of M_{10} to M_{15}. The transistors M_{16} to M_{20} operate as switches and are controlled by the digital signals $w_{\pm}$ w_0, w_1, and w_2. The value of $w_{\pm}$ determines whether the positive branch (pFETs M_{13} to M_{15}; adding current to the node I_{out}) or the negative branch (nFETs M_{10} to M_{12}; subtracting current from the node I_{out}) is switched on and thereby the sign of the synaptic multiplication factor. Setting w_0, w_1, and w_2 allows switching on or off specific contributions to the output current. In the example implementation the widths of M_{10} to M_{12}, and M_{13} to M_{15}, respectively, were scaled by powers of 2 (see Tab. 2), such that a synapse would implement a multiplication by a factor approximately corresponding to the binary value of (w_0, w_1, w_2). While our results are based on a signed 3-bit version of the circuit, arbitrary precision can be implemented by changing the number of scaling transistors and corresponding switches. The dimensions of M_3 and M_4 were adjusted such that the currents through transistors of the positive and the negative branch of one particular bit of a synapse were roughly matched when switched on.

Table 2: Transistor dimensions used in all circuit simulations.

Device	W (µm)	L (µm)	W/L
$M_0 - M_4$	2.7	0.45	6
$M_{10},\ M_{13}$	0.27	0.54	0.5
$M_{11},\ M_{14}$	0.54	0.54	1
$M_{12},\ M_{15}$	1.08	0.54	2
$M_{16} - M_{20}$	0.54	0.54	1

Multilayer networks were constructed using the circuits described above by connecting layers of soma circuits through matrices made up of synapse circuits. The first stage of a network constructed in this way thereby is a layer of soma circuits, rather than a weight matrix, as is typically the case in artificial neural network implementations. This is because we prefer to provide input currents rather than voltages and only soma circuits take currents as inputs. As a consequence, due to the rectification, our network can not handle negative input signals. To obtain current outputs rather than voltages, one synapse is connected to each unit of the output layer and its weight set to 1 to convert the output voltages to currents.

3.2 Circuit simulation details

All circuits were simulated using NGSPICE release 26 and BSIM3 version 3.3.0 models of a TSMC 180 nm process. The SPICE netlist for a particular network was generated using custom Python software and then passed to NGSPICE for DC and transient simulations. Input patterns were provided to the input layer by current sources fixed to the respective values. The parameters from Tab. 2 were used in all simulations and V_{dd} was set to 1.8 V. Synapses were configured by setting their respective configuration bits $w_{\pm}$, w_0, w_1, and w_2 to either V_{dd} or ground, emulating a digital memory element. The parasitic capacitances and resistances to be found in an implementation of our circuits were estimated from post-layout simulations of single soma and synapse cells. The main slowdown of the circuit can be attributed to the parasitic capacitances of the synapses, which were found to amount to 11 fF per synapse.

Individual hardware instances of our system were simulated by randomly assigning small deviations to all transistors of the circuit. Since the exact nature of mismatch is not relevant for our main result (our training method compensates for any kind of deviation, regardless of its cause), the simple but common method of threshold matching was applied to introduce device-to-device deviations [28]. Specifically, for every device, a shift in threshold voltage was drawn from a Gaussian distribution with zero mean and standard deviation $\sigma_{\Delta VT} = A_{VT}/\sqrt{W/L}$, where the proportionality constant A_{VT} was set to 3.3 mVμm, approximately corresponding to measurements from a 180 nm process [39].

3.3 Characterization of the simulated circuit

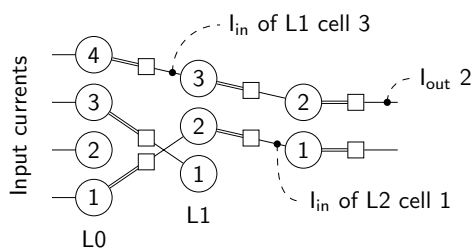

Figure 6: Illustration of the measurement procedure applied to the simulated circuits. The diagram shows one possible weight configuration that might come up during the parameter extraction procedure of a network with one input, one hidden, and one output layer. Circles represent soma circuits and squares synapse circuits. Voltages are represented by double lines, whereas currents are represented by single lines. Only synapses set to non-zero values are shown. Every unit receives exactly one input signal, and produces, together with a connected synapse circuit, at maximum one output current, which can be measured as the input to a unit of the consecutive layer. The input to the network is provided in terms of a set of input currents, the output is transformed to currents by means of an additional array of synapses after the last layer.

To determine the transfer curves of individual neurons, the input-output relations of the respective soma circuits need to be measured. To save simulation time, a parallel measurement scheme was applied, based on the assumption that each neuron can be measured directly, rather than just the neurons in the output layer. Rather than measuring the log domain output voltages V_n and V_p we chose to record the input currents I_{in} to subsequent layers. The advantages of this approach are that quantities are not log-transformed and that potential distortions arising from the synapse circuits are taken into account. Furthermore, with this method only one probe is required per neuron, rather than two separate ones for in- and output signals. Moreover, the unit weight of a synapse (which is not know a priori) here becomes a property of the soma, so that weights are automatically normalized. To determine the transfer curves of the units in the different layers the weights were set to a number of different configurations and the input currents to the various units were measured for different input patterns provided to the network. Specifically, by setting the respective synapse circuits to their maximum value, every unit was configured to receive input from exactly one unit of the previous layer. One such configuration is shown in Fig. 6. The input currents to all units of the input layer were then set to the same value and the inputs to the units of the deeper layers were recorded. By generating many such connectivity patterns by permuting the connectivity matrix, and setting the input currents to different values, multiple data points (input-output relations) were recorded for each unit, such that continuous transfer curves could be fitted to the data. For the example networks described in Sect. 1.2, 40 measurements turned out to be sufficient, resulting in roughly 10 data points per unit. Rectified linear functions $f(r) = \max\{0, a \cdot r\}$ were fitted to the data and the resulting parameters a were used as part of the training algorithm. The parameters were normalized layer-wise to a mean slope of 1. Even though the sizes of the transistors implementing the positive and negative weight contributions are identical, their responses are not matched. To characterize their relative contributions, inputs were given to neurons through positive and negative connections simultaneously. Comparing the neuron response to its response with the negative connection switched off allows to infer the strength of the unit negative weight, which can then be used in the training algorithm.

3.4 Training and evaluation details

The $196 - 100 - 50 - 10$ networks were trained on the MNIST and TIDIGITS datasets using the ADAM optimizer [27] and the mean squared error as loss function. The low-precision training (three signed bits per synapse) was done using a high-precision store and low-precision activations in the manner of the method simultaneously described in [48, 10]. An L1 regularization scheme was applied to negative weights only to reduce the number of negative inputs to neurons, as they would slow down the circuits. The Keras software toolkit [9] was used to perform the training. A custom layer consisting of the parameterized activation function $f(x) = \max\{0, a \cdot Wx\}$, using the extracted parameter a was added and used to model the neuron activation function.

Different sets of empirically found hyperparameters were used during training for the MNIST

and TIDIGITS datasets. A reduced resolution version (14×14 pixels) of the MNIST dataset was generated by identifying the 196 most active pixels (highest average value) in the dataset and only using those as input to the network. The single images were normalized to a mean pixel value of 0.04. The learning rate was set to 0.0065, the L1 penalty for negative weights was set to 10^{-6}, and the networks were trained for 50 epochs with batch sizes of 200.

Each spoken digit of the TIDIGITS dataset was converted to 12 mel-spectrum cepstral coefficients (MFCCs) per time slice, with a maximum frequency of 8 kHz and a minimum frequency of 0 kHz, using 2048 FFT points and a skip duration of 1536 samples. To convert the variable-length TIDIGITS data to a fixed-size input, the input was padded to a maximum length of 11 time slices, forming a 12x11 input for each digit. First derivative and second derivatives of the MFCCs were not used. To increase robustness, a stretch factor was applied, changing the skip duration of the MFCCs by a factor of 0.8, 0.9, 1.0, 1.1, and 1.3, allowing fewer or more columns of data per example, as this was found to increase accuracy and model robustness. A selection of hyperparameters for the MFCCs were evaluated, with these as the most successful. The resulting dataset was scaled pixel-wise to values between 0 and 1. Individual samples were then scaled to yield a mean value of 0.03. The networks were trained for 512 epochs on batches of size 200 with the learning rate set to 0.0073, and the L1 penalty to 10^{-6}.

3.5 Performance measurements

The accuracy of the abstract software model was determined after training by running the respective test sets through the network. Due to prohibitively long simulation times, only subsets of the respective test sets were used to determine the accuracy of the SPICE-simulated circuits. Specifically, the first 500 samples of the MNIST test set and 500 randomly picked samples from the TIDIGITS test set were used to obtain an estimate of the classification accuracy of the simulated circuits. The data was presented to the networks in terms of currents, by connecting current sources to the I_{in} nodes of the input layer. Individual samples were scaled to yield mean input currents of 15 nA or 45 nA per pixel, respectively. The time to output for a particular pattern was computed by applying one (random) input pattern from the test set and then, once the circuit had converged to a steady state, replaced by the input pattern to be tested. In this way, the more realistic scenario of a transition between two patterns is simulated, rather than a 'switching on' of the circuit. The transient analysis was run for 7 µs and 15 µs with the mean input strength set to 45 nA and 15 nA, respectively, and a maximum step size of 20 ns. At any point in time, the output class of the network was defined as the index of the output layer unit that was the most active. The time to output for each pair of input patterns was determined by checking at which time the output class of the network corresponded to its asymptotic state (determined through an operating point analysis of the circuit with the input pattern applied) and would not change anymore. The energy consumed by the network in a period of time was computed by integrating the current dissipated by the circuit over the decision time and multiplying it by the value of V_{dd} (1.8 V in all simulations).

3.6 VLSI prototype implementation

A $7-7-7$ network, consisting of 21 neurons and 98 synapses was fabricated in 180 nm CMOS technology (AMS 1P6M). The input currents were provided through custom bias generators, optimized for sub-threshold operation [12]. Custom current-to-frequency converters were used to read out the outputs of neurons and send them off chip in terms of inter-event intervals. The weight parameters were stored on the device in latches, directly connected to the configuration lines of the synapse circuits. Custom digital logic was implemented on the chip for programming biases, weights, and monitors. Furthermore, the chip was connected to a PC, through a Xilinx Spartan 6 FPGA containing custom interfacing logic and a Cypress FX2 device providing a USB interface. Custom software routines were implemented to communicate with the chip and carry out the experiments. The fabricated VLSI chip was characterized through measurements as described in Sect. 1.2, by probing individual neurons one by one. The measurements were repeated several times through different source and monitor neurons for each neuron to be characterized to average out mismatch effects arising from the synapse or readout circuits. The mean values of the measured slopes were used in a software model to train a network on the Iris flower dataset. The Iris dataset was randomly split into 120 and 30 samples used for training and testing, respectively. The resulting weight parameters were programmed into the chip and individual samples of the dataset were presented to the network in terms of currents scaled to values between 0 and 325 nA. The index of the maximally active output unit was used as the output label of the network and to compute the classification accuracy.