Towards Addressing Noise and Static Variations of Analog Computations using Efficient Retraining

Bernhard Klein[1], Lisa Kuhn[1], Johannes Weis[2], Arne Emmel[2], Yannik Stradmann[2], Johannes Schemmel[2], and Holger Fröning[1]

[1] Institute of Computer Engineering, Heidelberg University, Germany
{bernhard.klein,holger.froening}@ziti.uni-heidelberg.de
lisa.kuhn@stud.uni-heidelberg.de
[2] Kirchhoff-Institute for Physics, Heidelberg University, Germany
{johannes.weis,arne.emmel,yannik.stradmann,schemmel}@kip.uni-heidelberg.de

Abstract. One of the most promising technologies to solve the energy efficiency problem for artificial neural networks on embedded systems is analog computing, which, however, is fraught with noise due to summations of unwanted or disturbing energy, and static variations related to manufacturing. While these inaccuracies can have a negative effect on the accuracy, in particular for naively deployed networks, the robustness of the networks can be significantly enhanced by a retraining procedure that considers the particular hardware instance. However, this hardware-in-the-loop retraining is very slow and thus often the bottleneck hindering the development of larger networks. Furthermore, it is hardware-instance-specific and requires access to the instance in question.

Therefore, we propose a representation of a hardware instance in software, based on simple, parallelization-friendly software structures, which could replace the hardware for the major fraction of retraining. The representation is based on lookup tables, splines as interpolated functions and additive Gaussian noise to cover static variations together with electrical noise of the multiplier array and column-wise integrators. The combined approach using the proposed representation together with some final epochs of hardware-in-the-loop retraining reduces the overall training time from over 10 hours to less than 2 hours compared to a full hardware-in-the-loop retraining, while notably increasing accuracy. This work highlights that including device-specific static variations and noise in the training process is essential for a time-efficient hardware-aware network training for analog computations, and that major parts can be extracted from the hardware instance and represented with simple and efficient software structures. This work is the first step towards hardware-specific but hardware-inaccessible training, addressing speed and accuracy.

Keywords: Analog Hardware Representation · Hardware-aware Training · Static Variations · Electrical Noise · Analog Computations.

1 Introduction

Machine learning (ML) has quickly established as key component for various applications, ranging from image processing over robotics and natural language processing to signal processing, and is pervasively deployed on devices including mobile, embedded and edge devices. However, there is a substantial gap in between ML's compute and memory requirement and hardware capability, amplified by a staggering Moore's law and the end of Dennard scaling.

As in the Post-Dennard performance scaling era performance in operations per second is best defined as the product of the power budget in Watts and the energy efficiency in operations per Joule, energy efficiency of arithmetic operations and their associated data movements are becoming first-class citizens, and limited resources for mobile, embedded and edge devices due to physical footprint and battery life are amplifying the problem. While digital CMOS processing is still the most common option for ML and various other computational tasks, the interest in alternatives is increasing. There is a plethora of options in the context of *Beyond Moore*, such as quantum computing, neuromorphic computing, and several more. Among those, possibly analog CMOS computing is least disruptive to the existing compute stack. While analog computations can substantially improve the energy efficiency of computations, it comes at the cost of uncertainty in computations, both with regard to *static variations* such as non-linearities as well as *dynamic noise*. While digital computing relies on binarization to increase the resilience towards such imperfections, analog computing is much more sensitive. Still, there are tradeoffs, for instance by choosing a suitable bit width one can avoid a dominating influence of otherwise prohibitive thermal noise in analog computing [19].

There exists a notable set of related work on analog computing, which usually is due to its uncertainty rather application-specific. In the context of machine learning, the work by Murmann describes a mixed-signal processor architecture for artificial neural networks (ANN) [9], which proposes a processing array that allows for much denser processing elements than its digital counterpart, and leverages this in combination with a reduced power budget per element for an increase in parallelism. Another analog computing platform for machine learning applications is BrainScaleS-2 (BSS-2), which utilizes analog operations for the computation of spiking networks as well as ANNs [15]. While this is another example for mixed-signal computation, the implementation details differ substantially.

Furthermore, ANN architectures are apparently quite tolerant against uncertainties in the computation, as there exists a rich body of previous work on quantization and pruning [7,14]. Essentially, both techniques are unsafe optimizations [22] and introduce noise in the computation, however, previous work has shown that retraining is an effective measure to counter a potential loss in prediction quality [1]. Examples for work that enhances the robustness of ANNs for analog computing are based on knowledge distillation [23] and noise injecting training [12], and demonstrate the potential robustness of ANNs.

The main idea of this work stems from the observation that neural networks can be tolerant against inaccurate computations, but this inaccuracy has to be represented in the training process. While including the uncertain hardware in the forward path of the training process can be an option, so-called *hardware-in-the-loop training*, it is often slowing down this process substantially as such analog hardware is not optimized for peak throughput but rather peak energy efficiency. Based on this observation, the present work is concerned with finding a suitable *representation* of the hardware including non-idealities found in analog computing, such that the required hardware involvement can be reduced to a minimum, thereby saving training time. Although being precise enough in representing the hardware peculiarities, this representation has to be compact and efficient to compute, as otherwise training time penalties can occur. Last, as every analog computing instance differs, it has to be trainable in an automated manner to a particular hardware instance.

We therefore propose a representation based on lookup tables and splines as interpolated functions in combination with a configurable amount of additive noise, to represent static variations as well as dynamic noise. The lookup tables represent non-linearities, the splines describe saturation effects, while the additive noise mimicks the various noise sources found in analog computations. We show that this representation can be derived from an automated characterization of a particular hardware instance. We base our characterization on BSS-2 as a real-world prototype of analog computing, fabricated in a generic 65nm CMOS technology and available in sufficient quantities. We demonstrate the effectiveness of our abstract yet precise representation by comparing the accuracy of a keyword spotting task to the accuracy of an ANN trained directly on hardware as well as full-precision standard computations. In detail, this work makes the following contributions:

- Proposing a sufficiently precise and compute efficient representation of hardware non-idealities, suitable to be included in the training framework.
- Designing a training methodology to compensate all major hardware imperfections, reducing time-costly hardware-in-the-loop training.
- Comparing the performance of various representations in terms of accuracy and training time, including various baselines such as full-precision accuracy.

As a result, the training of neural networks on inaccurate hardware is substantially simplified and accelerated, furthermore allowing researchers to reason in an abstract way about the implications of uncertain computations on ML tasks. We see the resulting methodology as key for further research on robust network architectures and explorations in the context of HW/SW-Codesign with uncertain hardware.

2 Background and Related Work

While in principle analog computing exists also based on optical [6], photonic [16] or phase-change-memory [11] technology, most commonly found and focus of the

present work is electronic CMOS technology [9]. In such, computations such as multiplication and addition can be easily represented using the physical laws of network analysis. The key operation of most ANNs is the dot product. As one option among various alternatives, the input operands of a multiply-accumulate (MAC) operation can be represented as current pulses, with a pulse's length Δt_i (time) being the input activation and its amplitude the synaptic weight $I_{i,j}$. A single multiplication is then the time integral over this pulse, therefore charge, whilst the result of the MAC over all inputs equals $Q_j = \sum_i I_{i,j} \cdot \Delta t_i$. In particular, BSS-2 operates in exactly this scheme.

While in theory analog computations following Kirchhoff's laws require no energy, in practice charging and de-charging capacitors results in inefficiencies, the required translation from digital signaling to analog quantities requires conversion, and leakage currents and conductances are not ideal, which all contribute to a notable amount of energy. Still, as long as thermal noise is not the dominant non-ideality, analog electrical computing is considered orders of magnitude more energy-efficient than its digital counterpart [9].

2.1 BrainScaleS-2

The presented framework is tested using BrainScaleS-2—a mixed-signal neuromorphic platform based on a custom ASIC manufactured in 65 nm CMOS technology [15]. It supports the accelerated emulation of spiking neural networks (SNN) as well as the processing of MAC operations within its analog core [21]. The ASIC consists of 512 neuron circuits, each of which is connected to a column of 256 synapses as well as a dedicated ADC channel for activation readout. The synapses are of 6 bit precision. As its sign can be selected row-wise, to achieve signed weights we combine two hardware synapses to a single virtual synapse (Fig. 1). When computing a matrix-vector multiplication, the input is encoded as a vector of 5 bit unsigned integers. Its entries are multiplied inside the synapse array and accumulated on a capacitor representing the neuron's activation, which is digitized using columnar ADCs with 8 bit precision. Each vector may be sent multiple times N_{sends} with an adjustable pause between individual vector entries T_{pause} to optimize signal amplitudes for the circuits' dynamic range.

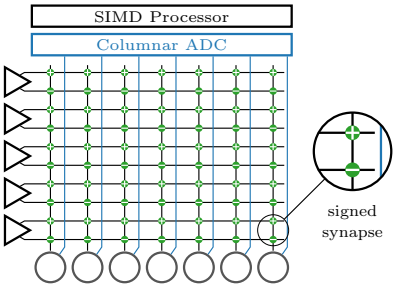


Fig. 1: Block diagram of the BSS-2 analog core, showing synapse drivers (triangles), neurons (large circles), and synapses (small circles in matrix) [2].

The process of analog computation is affected by multiple sources of uncertainties: As the analog matrix-vector multiplication is embedded in a full digital setting, multiple *digital-analog and analog-digital conversions* are required. The involved circuitry does not maintain perfect linearity throughout its full dynamic range; especially apparent through offsets, saturation and range-specific amplification effects. Most notably, low vector inputs are exaggerated within the respec-

tive digital-to-analog conversion. Each computational element involves multiple *amplifiers*, all of which are subject to different degrees of saturation. Most prominently, saturation effects are observable for the circuits driving synaptic signals to the neuronal accumulators, being strongest for negative summands if the accumulated activation from preceding vector entries is already low. Moreover, the signals emitted by the *synapses* weaken if multiple high-valued summands of the same sign arrive within a short time interval. Last, a driver's signal deteriorates with physical distance, most notably for small values. In addition, all modules of an ASIC are subject to device mismatch due to imprecisions within the manufacturing process. For BrainScaleS-2, effects of these *static variations* primarily emerge for the analog circuits and result in a random distribution of the computational characteristics of all neurons and synapses. The hardware therefore provides calibration mechanisms that can be used to reduce the deviations between its computational elements, even though static variations cannot be eliminated completely in practice. Also, the result from any analog computation is superimposed by *electrical noise* of various sources, including coupled noise from neighboring circuitry and thermal noise. Integrating this noise results in deviations of the output activations that can be observed after digitization, ultimately reducing the obtainable resolution. Models for analog hardware therefore need to be robust against non-linearities, post-calibration mismatch and electrical noise.

2.2 Noisy Computations in the Context of Machine Learning

Most often, the dynamic noise of analog computations is modeled as additive zero-mean Gaussian noise, while the variance differs among the related works. For instance, in Rekhi et al. [13] the variance is a function of the number of bits of the output, in Joshi et al. [5] it depends on range of values a non-volatile memory device can store, while in Zhou et al. [23] the amount of additive noise is a configurable parameter to explore the robustness of network architectures against such. Static variations have also been considered before, but rather with regard to interactions among multiple devices as found in arrays of analog processing elements [4,3].

An early work on robust neural networks reports the benefits of noise injection during training on the example of MLPs [10]. Noisy machines [23] focuses on the additive Gaussian noise and neglects static variations to propose a methodology based on knowledge distillation to highlight and combat the reduced learning capactity of ANNs when executed on noisy hardware. A more general treatment of fault-tolerance of ANNs can be found in [18].

While we are also concerned about finding more robust ANNs, our work distinguishes from related work by the fact that we are not only concerned with *dynamic noise*, but also *static variations* and the inevitable amount of variance found across multiple hardware instances. While hardware-in-the-loop training is suitable to address the noise and variances, it comes with a substantial slowdown.

3 Hardware Representation

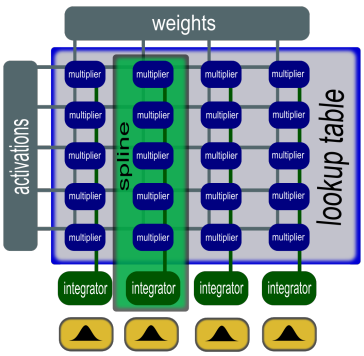


Fig. 2: Schematic overview of the representation: *lookup table* for the static variations of the multiplier array, *splines* for the column and integrator specific variations, and additive *Gaussian noise* that models the electrical noise, among others.

The provided method is designed to represent chip specific peculiarities of inaccurate matrix multiply operations in the training process and is applicable for uncertain accelerators which are based on a multiplier array and accumulate with column-wise integrators. Although the proposed hardware representation requires that the columns are largely independent of each other and that there are no major time dependencies besides electrical noise, it is a very general concept that can be applied to a wide variety of accelerators. The peculiarities of the accelerator are distilled from measurements and mapped in a hardware representation, which is suitable to train machine learning models for this specific hardware as quickly as possible. Fig 2 illustrates the three distinct parts of the model which simulate the hardware and capture multiplier-array- and integrator-specific static variations together with electrical noise, which are a *lookup table* to represent the multiplier non-linearities, *splines* as piecewise polynomial functions to represent integrator imperfections, and *Gaussian additive noise* to represent the electrical noise observed during analog computations. In the following, we describe these components in more detail.

Lookup Table All multiplier effects are covered by storing the output for all possible inputs for each multiplier in a lookup table. On condition that the effects are reproducible and do not change over time, the lookup table enables to model complex behavior of the multipliers with highest possible generality. For the BSS-2 hardware such a column-row-activation-weight-based lookup table can be created by performing *row-wise measurements* to capture the row and column specific details with minimal interference. Since only one row is active at the same time the electrical signal strength is low, therefore saturation effects are negligible. The signal is amplified with $N_{\text{sends}} = 20$, $T_{\text{pause}} = 8$ to a medium high level to be measured with good quality. All other measurements and hardware-in-the-loop training uses $N_{\text{sends}} = 1$, $T_{\text{pause}} = 8$ to minimize saturation effects.

Splines While the lookup table can model multiplier specific variations, it is not capable to model effects which are based on interactions between them or belong to the *integrator*. For BSS-2, saturation effects dominate these *column-wise variations*. On condition that only multipliers connected to the same integrator can interact and interaction between integrators is negligible, properly designed

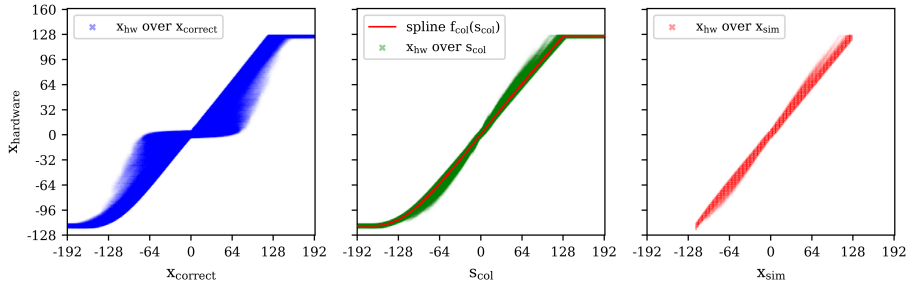


Fig. 3: Mismatch between values expected from a correct matrix multiply x_{correct} to measured results x_{hardware} for a random column (left). A relevant part of this errors can be corrected by using the sums of the lookup table s_{col} (middle), together with a interpolated spline to represent column specific saturation effects. On the right: comparison of representation x_{sim} and measurement x_{hardware} results. An optimal representation would lead to a straight line broadened by electrical noise.

functions can model this inter-multiplier and integrator specific behavior. The sum of values s_{col} in a column c extracted from the lookup table e_{lookup},

$$s_{\text{col}} = \sum_{r}^{I} e_{\text{lookup}}[r, c, a_r, w_r] \tag{1}$$

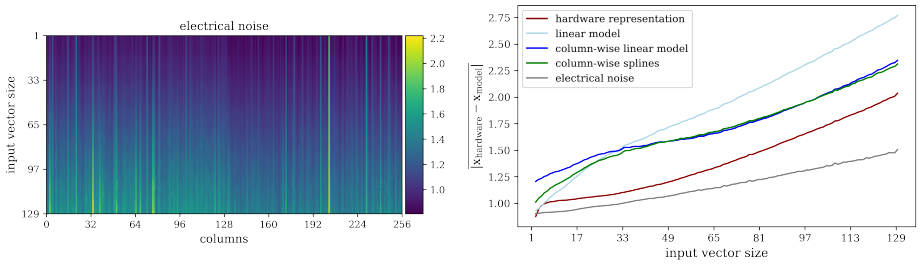
is mapped with column-specific function f_{col},

$$x_{\text{sim}} = f_{\text{col}}(s_{\text{col}}) \tag{2}$$

to the predicted hardware values x_{sim}, with I as input vector size, activation a_r and weight w_r of row r. Splines—piecewise polynomial functions—can be interpolated to act as this functions. A *full range measurement* is performed for all possible activations, weights and input vector sizes to include the limits of the value space and thus extract the saturation effects of the accelerator. All measured values x_{hardware} whose summed lookup table values s_{col} are in the same interval $\Delta s_i = [i\Delta s, (i+1)\Delta s]$ are averaged to get a stable mapping from Δs_i to $\overline{x_{\text{hardware}}}$. Fig. 3 shows exemplary the operation of lookup table and spline.

Electrical Noise With all *static variations* modeled by the *lookup table* and *splines* there are still sources of *electrical noise* which can not be covered with a deterministic model. However, due to their very statistical properties, they can be represented by adding noise sampled from a zero-mean *Gaussian distribution*. The variance is determined by a measurement where activation and weight matrices are chosen randomly and the same random input matrices are used to computed the outputs many times to measure the standard deviation of exactly the same operation. As Fig. 4a illustrates, the *electrical noise* increases

slightly with the number of summands and differs significantly between integrators, therefore the hardware representation of the *Gaussian variance* depends on the input vector size and column in the same way as observed on the hardware.



(a) (b)

Fig. 4: Left: Electrical noise measured as standard deviation of repeated calculations with identical inputs. With a larger input vector the average noise increases (vertical gradient), since more noisy values are accumulated, but also the variances between columns are significant (vertical lines). Right: Model performance as difference to hardware measurements for random inputs. A perfect model would be align with the electrical noise (gray) of the hardware. The proposed hardware representation outperforms other linear-regression and splines based models. Noise and model imperfections increase with input vector size due to more involved components and saturation effects.

Performance of various models to represent BSS-2 Fig. 4b compares linear models together with a column-wise spline-based model to the proposed hardware representation. The better a model is, the closer it comes to the baseline given by electrical noise. Due to more involved inaccurate and noisy components, a larger input vector size increases the error of all models. With increasing input vector size also the error of column-specific variations becomes more dominant since more values are amplified by them. Therefore for larger input vectors the column-wise models perform better than their non-specific counterparts. Although larger input vectors lead to more hidden saturation effects and thus increase the mismatch between representation and hardware, including multiplier-specific variations by its lookup table enables the proposed hardware representation to outperform all other models.

While the proposed general hardware representation models saturation effects with columnar splines, the BSS-2 hardware accumulates positive and negative summands separately. This leads to *hidden saturation effects* which occur in the sign intrinsic accumulation process and due to positive and negative cancellation are not visible in the final sum. However, the representation has to be compact and of high throughput, thus it cannot model all hardware imper-

fections in detail, particularly since accurate hardware emulations mostly are notably slower than execution on the hardware itself, which is in direct conflict to the design goals of a representation used to speed up the training process.

While PyTorch was used to be interchangeable with the BSS-2 interface *hxtorch* [17], the concept is universal and applicable with all major ML frameworks.

4 Training Methods and Results

Keyword Spotting Task We consider the keyword spotting task of the Speech Commands Dataset V1 [20] as being both representative and sufficiently complex to unveil differences in between proposed hardware representation and hardware itself. While there are different options to model such time-series problems, including recurrent neural networks, LSTMs and transformers, this work focuses on a multi-layer perceptron with log-mel filter [8]—illustrated in Fig. 5. Other architecture options are considered to be beyond the scope of the present work.

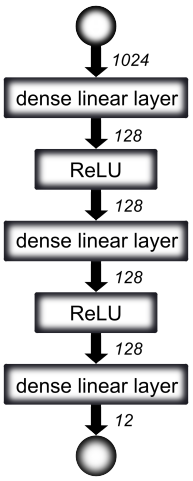


Fig. 5: Keywordspotting model

Training Methods based on the Hardware Representation For compression of ML models, retraining with the compressed model has proven to be very successful to compensate accuracy loss induced by the less accurate representations [1]. Here we follow a similar approach to compensate for hardware imperfections. In the retraining step, the already trained full-precision model is trained again but this time with the hardware or their representation used in the forward path. The intuition is, that then the backpropagation will automatically address the hardware imperfections by minimizing the errors injected in the forward path. Notably, complete hardware-in-the-loop retraining is the most precise way to integrate the hardware but also by far the slowest option.

Table 1 compares different training strategies reporting accuracy and retraining time, notably a *plain* training based on 300 epochs (which is also used as initialization for all other methods), a *quantized* retraining to achieve the required quantization (uint5 activations and int6 weights) of BSS-2, a *noise-only* retraining based on the plain method that adds injected zero-mean additive Gaussian noise, a *representation (rep.) no noise* retraining based on this work's lookup table and splines, a *representation (rep.) with noise* retraining based on this work's lookup table, splines and additive noise, a *representation (rep.) increasing (inc.) noise* retraining based on this work's lookup table, splines and a noise increasing over the epochs from 0 % for first epoch to 100 % of the original noise level on the last epoch, a *hardware-in-the-loop* retraining that uses a BSS-2 instance in the forward path, and a *combined* retraining that first relies on either the quantized model or on our representation with increasing noise, before finally retraining for a couple of epochs using hardware-in-the-loop.

Table 1: Comparison of various retraining methods

Method		Retraining		Accuracy	
Name	**Description**	**Epochs**	**Time in minutes**	**SW**	**BSS-2**
plain	original full precision model	0	0.0	80.8 %	12.3 %
quantized	uint5 activations, int6 weights	300	13.1	79.6 %	25.8 %
noise only	plain model & noise	300	10.4	80.5 %	18.4 %
rep. no noise	static variances without noise	300	83.4	76.5 %	26.4 %
rep. with noise	hw. rep. with noise	300	83.4	73.5 %	35.1 %
rep. inc. noise	hw. rep. increasing noise	300	83.6	76.0 %	41.0 %
hw-in-the-loop	full hardware-in-the-loop	300	652.3		66.8 %
hw-in-the-loop	full hardware-in-the-loop	350	769.5		66.7 %
combined qt.	quant. (300 ep.) & hw. (5 ep.)	305	13.1 + 11.5		62.1 %
combined qt.	quant. (300 ep.) & hw. (50 ep.)	350	13.1 + 117.5		67.3 %
combined rep.	rep. (300 ep.) & hw. (1 ep.)	301	83.6 + 2.2		64.9 %
combined rep.	rep. (300 ep.) & hw. (5 ep.)	305	83.6 + 11.5		67.4 %
combined rep.	rep. (300 ep.) & hw. (10 ep.)	310	83.6 + 23.5		69.7 %
combined rep.	rep. (300 ep.) & hw. (50 ep.)	350	83.6 + 117.5		70.1 %

Comparing Various Representations While the plain model performs very poorly on BSS-2, including more hardware characteristics with quantization, additive noise and hardware representation leads to an increased accuracy of the retrained model. Although most related work focuses on counter measures against noise [5,12,23], the experiments show that *static variations* are equally important and have to be considered. Moreover we found that increasing the noise level during the retraining leads to better results. This suggests that increasing the difficulty, in our case hardware imperfections, slowly during retraining makes it easier for the model to adapt. This might also be the reason for the astonishing fact that combining representation retraining with only a few epochs hardware-in-the-loop retraining leads to higher test accuracies than full hardware-in-the-loop retraining.

These first experiments clarify that although the representation is not yet exact enough to replace hardware-in-the-loop completely, it covers enough hardware particularities during the training to be much more accurate than a naive deployment. Harnessing the best of both worlds with a combined approach outperforms hardware-in-the-loop retraining in terms of accuracy and training time.

Training Time While the training with hardware in the loop is bound by BSS-2 throughput, the representation training is only limited by the available compute resources. For the experiments a single *NVIDIA Titan Xp* was used, however, the throughput scales with the resources and by using more or more powerful GPUs the required training time can be reduced easily. Although with this lightweight training resource, in contrast to typical high-end, multi-GPU training systems, a speedup of 7.8 for representation training and 6.9 for combined training (5 hw. epochs) over hardware-in-the-loop retraining can be reported.

5 Summary and Outlook

A representation is proposed to minimize the involvement of analog hardware during retraining, based on distilling all major hardware characteristics from measurements, and represented during training with a lookup table, splines and additive Gaussian noise, modeling particularities of the multiplier array, saturation effects of the integration, analog and digital conversion processes and inevitable electrical noise. Although the representation cannot replace the costly hardware-in-the-loop retraining step completely, it replaces most of it with much faster representation training, such that the amount of remaining epochs with the hardware is reasonable small compared to the rest and the overall training time is reduced from more than 10 hours to less than 2 hours, notably increasing accuracy. Overall, the proposed method demonstrates that including device-specific static variations and noise in the training process is essential to train hardware-aware robust neural networks for analog computations, and that major parts can be extracted from the hardware and represented with simple and parallelization-friendly software structures. This is the first step towards hardware-specific but hardware-inaccessible training, addressing speed and accuracy.

Acknowledgements

The development of BrainScaleS-2 has received funding from the German Federal Ministry of Education and Research under grant number 16ES1127, the EU (H2020/2014-2020: 720270, 785907, 945539 (HBP)) and the Lautenschläger-Forschungspreis 2018 for Karlheinz Meier. We also acknowledge the financial support from the COMET program within the K2 Center "Integrated Computational Material, Process and Product Engineering (IC-MPPE)" (Project No 859480). This program is supported by the Austrian Federal Ministries for Transport, Innovation and Technology (BMVIT) and for Digital and Economic Affairs (BMDW), represented by the Austrian research funding association (FFG), and the federal states of Styria, Upper Austria and Tyrol.

References

1. Courbariaux, M., Bengio, Y., David, J.P.: BinaryConnect: Training deep neural networks with binary weights during propagations. In: Advances in Neural Information Processing Systems. vol. 28. Curran Associates, Inc. (2015), https://dl.acm.org/doi/10.5555/2969442.2969588
2. Cramer, B., et al.: Training spiking multi-layer networks with surrogate gradients on an analog neuromorphic substrate (2020), https://arxiv.org/abs/2006.07239
3. Feinberg, B., Wang, S., Ipek, E.: Making memristive neural network accelerators reliable. In: 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA). pp. 52–65 (2018). https://doi.org/10.1109/HPCA.2018.00015
4. Jain, S., Sengupta, A., Roy, K., Raghunathan, A.: Rx-caffe: Framework for evaluating and training deep neural networks on resistive crossbars (2018), http://arxiv.org/abs/1809.00072

5. Joshi, V., et al.: Accurate deep neural network inference using computational phase-change memory. Nature Communications **11**(1), 2473 (May 2020). https://doi.org/10.1038/s41467-020-16108-9
6. Lin, X., et al.: All-optical machine learning using diffractive deep neural networks. Science **361**(6406), 1004–1008 (2018). https://doi.org/10.1126/science.aat8084
7. Liu, Z., et al.: Rethinking the value of network pruning. In: International Conference on Learning Representations (2019), https://arxiv.org/abs/1810.05270
8. Mermelstein, P.: Distance measures for speech recognition, psychological and instrumental. Pattern recognition and Artificial Intelligence **116**, 374–388 (1976)
9. Murmann, B.: Mixed-signal computing for deep neural network inference. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **29**(1), 3–13 (2021). https://doi.org/10.1109/TVLSI.2020.3020286
10. Murray, A., Edwards, P.: Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training. IEEE Transactions on Neural Networks **5**(5), 792–802 (1994). https://doi.org/10.1109/72.317730
11. Nandakumar, S.R., Le Gallo, M., Boybat, I., Rajendran, B., Sebastian, A., Eleftheriou, E.: A phase-change memory model for neuromorphic computing. Journal of Applied Physics **124**(15), 152135 (2018). https://doi.org/10.1063/1.5042408
12. Qin, M., Vucinic, D.: Noisy computations during inference: Harmful or helpful? CoRR **abs/1811.10649** (2018), http://arxiv.org/abs/1811.10649
13. Rekhi, A.S., et al.: Analog/mixed-signal hardware error modeling for deep learning inference. In: 56th Annual Design Automation Conference. DAC, Association for Computing Machinery (2019). https://doi.org/10.1145/3316781.3317770
14. Roth, W., et al.: Resource-efficient neural networks for embedded systems. CoRR **abs/2001.03048** (2020), http://arxiv.org/abs/2001.03048
15. Schemmel, J., Billaudelle, S., Dauer, P., Weis, J.: Accelerated analog neuromorphic computing. CoRR **abs/2003.11996** (2020), https://arxiv.org/abs/2003.11996
16. Shen, Y., et al.: Deep learning with coherent nanophotonic circuits. Nature Photonics **11**(7), 441–446 (Jul 2017). https://doi.org/10.1038/nphoton.2017.93
17. Spilger, P., et al.: hxtorch: PyTorch for BrainScaleS-2 — perceptrons on analog neuromorphic hardware. In: IoT Streams for Data-Driven Predictive Maintenance and IoT, Edge, and Mobile for Embedded Machine Learning. pp. 189–200. Springer (2020), https://doi.org/10.1007/978-3-030-66770-2_14
18. Torres-Huitzil, C., Girau, B.: Fault and error tolerance in neural networks: A review. IEEE Access **5**, 17322–17341 (2017). https://doi.org/10.1109/ACCESS.2017.2742698
19. Vittoz, E.: Future of analog in the vlsi environment. In: IEEE International Symposium on Circuits and Systems. pp. 1372–1375 vol.2 (1990). https://doi.org/10.1109/ISCAS.1990.112386
20. Warden, P.: Speech commands: A dataset for limited-vocabulary speech recognition. CoRR **abs/1804.03209** (2018), http://arxiv.org/abs/1804.03209
21. Weis, J., et al.: Inference with artificial neural networks on analog neuromorphic hardware. In: IoT Streams for Data-Driven Predictive Maintenance and IoT, Edge, and Mobile for Embedded Machine Learning. pp. 201–212. Springer (2020), https://doi.org/10.1007/978-3-030-66770-2_15
22. Whatmough, P., Wei, G.Y., Brooks, D.: Deep Learning for Computer Architects. Morgan & Claypool Publishers (2017)
23. Zhou, C., et al.: Noisy machines: Understanding noisy neural networks and enhancing robustness to analog hardware errors using distillation. CoRR (2020), https://arxiv.org/abs/2001.04974