Hardware Approximation of Exponential Decay for Spiking Neural Networks

Sherif Eissa
Eindhoven University of Technology
Eindhoven, The Netherlands
s.s.b.eissa@tue.nl

Sander Stuijk
Eindhoven University of Technology
Eindhoven, The Netherlands
s.stuijk@tue.nl

Henk Corporaal
Eindhoven University of Technology
Eindhoven, The Netherlands
h.corporaal@tue.nl

***Abstract*—Spiking neural networks (SNNs) may enable low-power intelligence on the edge by combining the merits of deep learning with the computational paradigms found in the human neo-cortex. The choice of neuron model is an open research topic. Many spiking models implement neural dynamics from biology that involve one or more exponential decay functions. Previous work focused on accurate modeling of the exponential decay function on neuromorphic hardware to the last significant bit (LSB). In this paper, we explore the limits of error resilience in SNNs by aggressively approximating their exponential decay functions and allowing for losses within our bit precision. Three approximation methods are presented and implemented with varying degrees of precision resulting in 10 different implementations. Their hardware cost and inference accuracy on benchmark networks and applications are compared. To improve the inference accuracy, we implemented fine-tuning. We also introduced hardware programmability for certain time constants as hyperparameters. Our results show resilience to lossy approximation, fast fine-tuning, and a low energy consumption of 47 fJ per operation.**

***Index Terms*—Neuromorphic Computing, Spiking Neural Networks, Exponential function, Error Resilience, Deep Learning**

I. Introduction

SNNs are the next generation of deep learning models that can enable efficient low power real-time edge applications. SNNs can leverage event-based neuromorphic systems to exploit sparsity. In addition to weights, neuron models contain states (e.g, neuron's potential) which hold the temporal history left by previous inputs. Figure 1 shows the dynamics of the famous Leaky Integrate and Fire (LIF) neuron [1]. Upon receiving an input spike, the neuron's potential increases according to the synaptic strength (weight). This potential leaks (i.e., exponentially decays) through time [1] until the neuron returns to the rest potential if it remains unexcited for enough time. If the neuron's potential reaches a certain threshold, it fires, releasing its potential as an output spike to connected neurons and it returns to its resting potential.

Implementing exponential decay accurately in hardware is resource and energy inefficient. Deep networks have shown to be error resilient [2]. In this paper, we exploit this resilience to approximate the decay function to save resources and energy. We explore three methods; two known methods (i.e., PWL and LUT) and a novel one called logarithmic scaling. We systematically study aggressive approximation by implementing each method with different precision levels and comparing their costs and accuracy on benchmark networks. We also recover accuracy loss through fine-tuning.

This work is funded by the NWO Perspectief program EDL (P16-25).

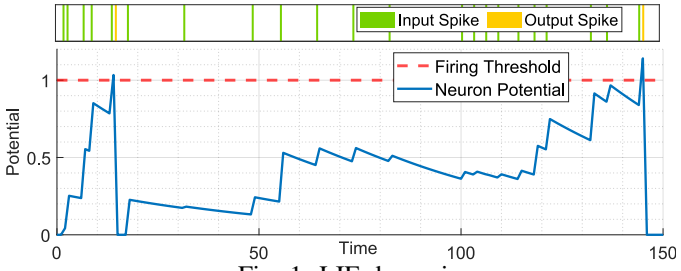

Fig. 1: LIF dynamics.

All three methods use fixed time constants. In practice, SNNs require decay functions with different time constants. Related work scale the inter-spike interval (ISI) to be in accordance with the time constant of a neuron's state (τ) [3], we call this *pre-processing*. We study the feasibility of programming the hardwares' time constant at compile-time hereby saving cost for systems with few time constant values.

Section II discusses related work. Section III presents our proposed approximations while programmability is discussed in Section IV. Sections V and VI describe our experiments on benchmark networks and applications and hardware evaluation. Section VII concludes our work.

II. Related Work

[4], [5] approximated the exponential function using Taylor series and CORDIC based routines, respectively. These solutions are iterative and are not pipelined. Pipelining is essential for digital neuromoprhic hardware to enable multiplexing of many neurons on the same computational hardware [6].

TS-EFA [3] and [7] are Look-up Table (LUT) based implementations that use two LUTs to calculate two factors. [7] calculates remaining third factor as a 4th order Taylor series. We extend them in Section III-A with aggressive approximation and we implement TS-EFA as our baseline.

[8] implements a piece-wise linear (PWL) approximation that targets the Softmax activation function in Artificial Neural

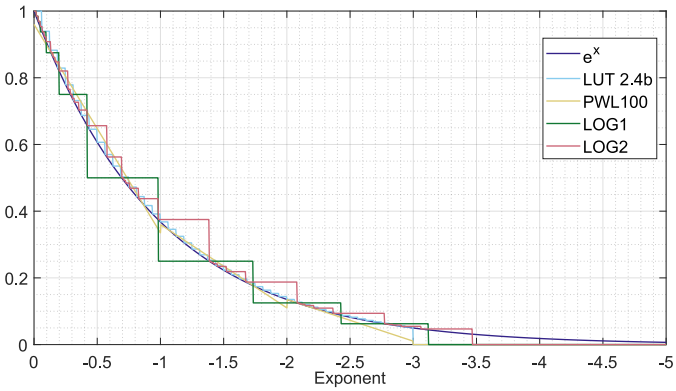

Fig. 2: Some implementations plotted together.

Networks. The slopes are realized using bit-shifts and adders. We extend this work for SNNs in Section III-B.

III. Exponential Decay Approximation

In this section, we will discuss the theory and implementation of our three methods: LUT-based, PWL, and our novel logarithmic scaling methods. Our contribution, besides logarithmic scaling, is an aggressive approximation, allowing for significant loss in SNN inference accuracy rather than functional accuracy. The cost of pre-processing is hidden. We map exponent values between around -3 and 0 since exponents beyond -3 cause heavy decay ($< 5\%$) and are considered zero. Figure 2 shows different implementations.

A. LUT-based implementation

In this method, we make use of the product rule of exponential functions. Due to our limited mapping, we only require two integer bits for our calculations. We use two LUTs to calculate the output potential according to equation (1) where x_L and x_S are the two factors stored in the two LUTs. We scale the precision with different LUT sizes; 16x16, 16x8 and 8x8 to implement exponential decay with 2.6, 2.5 and 2.4 bits precision, respectively. This method requires two multipliers.

$$S_{out} = S_{in} \times e^{-x} = S_{in} \times e^{-x_L} \times e^{-x_S} \tag{1}$$

The relative error of LUT implementation is uniformly distributed across the input domain as a periodic saw-tooth shaped function (see Figure 2). The period and magnitude of the relative error is dependent only on the bit precision.

B. Piecewise Linear (PWL) Functions

PWL converts a function into regions interpolated by linear functions. We evaluate 4 PWL implementations of varying complexity: PWL25, PWL50, PWL100 and Bi-linear. Bi-linear consists of two linear functions for $x \in [-3, 0]$ while others consist of one linear function for $x \in [-3, -2]$ and linear functions every 0.25, 0.5 and 1 step for $x \in [-2, 0]$ for PWL25, PWL50 and PWL100 respectively.

Similar to [8], we approximate the slopes of functions to the nearest 3 binary factors and implement them using 3 shift registers and adders. In addition, a MUX and registers are used to choose the appropriate linear function except for PWL25 which uses LUTs. Finally, a multiplier is used to multiply the decay with neuron's state and produce the neuron's final state.

C. Logarithmic Scaling (LOG)

Logarithmic scaling breaks down the exponential function to easily calculated factors using the product rule. While LUT-based approximation uses equidistant points to calculate exponent factors, simplifying point selection to simple memory accesses, but requiring two multipliers to calculate the factor and the final result. Logarithmic scaling, on the other hand, chooses easily calculated factors, putting strain on point selection, but simplifying calculations.

LOG uses logarithmically spaced factors in the *input* domain ($i.e, x_1 \approx 0.5x_2$) for *small* negative exponents ($x > ln(0.5)$) and in the *output* domain ($i.e, y_1 = 2y_2$) for *large* negative exponents ($x < ln(0.5)$) (see LOG1 in Figure 2) where factors take the form $S_{out} = S_{in} - S_{in}2^{-n}$ for small negative exponents and $S_{out} = S_{in}2^{-n}$ for large negative exponents - see Figure 3. This simplifies translating factors and neuron states to shift and subtract operations (i.e, no multipliers are needed). However, it complicates point selection to parallel comparison and thermometer decoding. Each LOG stage produces one factor similar to equation (1) whereas stages can be cascaded to improve precision.

The number of points is limited, which makes point selection feasible. Furthermore, only the first stage compares large negative exponents. For large exponents, n goes from 1 to 5, while for small exponents, n goes from 2 to 6. Hence, the first stage requires 10 comparisons while the others only require 5.

Each extra stage heavily increases accuracy - see LOG1 vs LOG2 in Figure 2. This comes at a significant hardware cost. However, increasing n for small exponents increases precision for very small inputs at an insignificant cost. LOG can be iterative by reusing a single stage, to provide trade-off between accuracy and energy/latency but at the cost of pipelining [4].

We implemented LOG1, LOG2 and LOG3 with one, two, and three factorization stages. Figure 3 shows a simplified dataflow diagram of the first LOG stage. Parallel comparisons are done and counted (pop-count) to calculate shift value (n). We subtract the chosen factor from the exponent to produce exponent remainder for the next stage if needed.

LOG has a desired effect for SNNs. It has a low relative error for smaller exponents and a relaxed relative error for larger ones due to the logarithmic scaling of points (Figure 2).

For inference accuracy, we are interested in the compounded error in between two output spikes which can cause a shift in output spike times between the exact and approximated models. If the ISI between two spikes is large (e.g, greater than τ) and thus the exponent is large, the neuron state decays significantly. This implies that the history held by the neuron state has become less significant to the output and does not require precise calculation. For small ISIs, we observed two cases: a) if the input spike responses are strong (due to having a strong synaptic weight), the amount of compounded error is

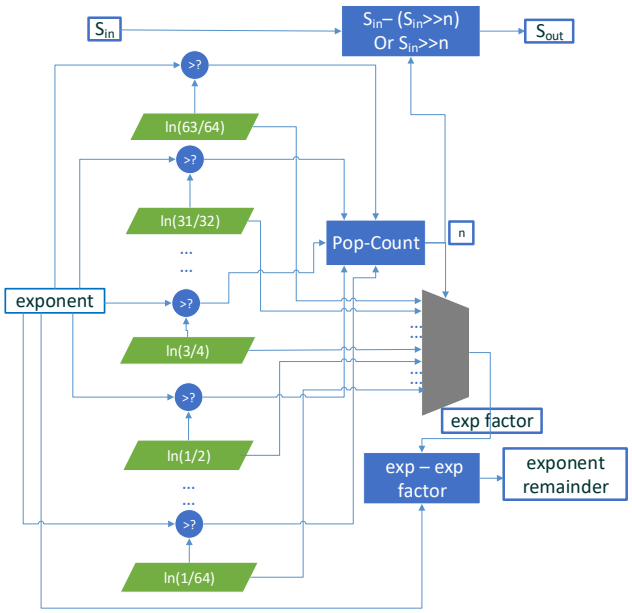

Fig. 3: Simplified RTL of first stage of Logarithmic scaling.

relatively low due to the low number of events needed to reach the firing condition, b) if the input spike responses are weak, the amount of compounded error is relatively high due to the high number of events needed to reach firing condition. This can cause a significant error in relative output spike timing. Hence, relative error is more significant for small negative exponents. The relationship between events and the overall error can be expressed as:

$$DecayError = EPE^{NE}, \tag{2}$$

where EPE is the average error per event and NE is the number of input events received between two output spikes.

LOG does not require multiplication. Each stage consists of comparators, 1 shift register and 1 subtractor for the output, and a MUX and a subtractor for calculating the exponent remainder in intermediate stages.

IV. Programmability

It is common for SNNs found in literature to have one time constant parameter per layer or even for the entire network [9], [10], [11], [12], [13]. Hence, a fairly good neuromorphic compiler can compile a network such that each computational hardware deals with a specific time constant parameter for all its neurons. Hence, programming our hardware with a specific time constant is a useful feature to save pre-processing cost.

LUTs can be programmed to be directly accessed with ISI without pre-processing. This idea was demonstrated in [3]. However, in our implementation, the limited sizes of our LUTs might be insufficient for some networks according to their time constants and their time granularity ($\frac{dt}{\tau}$). The condition for programmability is that the decay from ISIs beyond the LUT sizes (i.e, $ISI > LUT^1 \times LUT^2$) can be safely approximated to zero ($e^{\frac{dt}{\tau} LUT^1 \times LUT^2} \approx 0$). This can be verified by simulation on inference engines. Otherwise, we recommend larger LUTs to avoid performance variation.

For PWL implementations, programmability complicates region selection. This reduces the benefits of programmability which looks to remove the cost of one multiplication.

For logarithmic scaling, programmability is very simple. It only requires scaling the comparator constants by multiplying them with the time constant τ.

V. Experimental Results

A. Models

To test our implementations, we chose three benchmark networks. We chose networks with simplified neuron models that are suitable for event-based computation such as LIF or Adaptive exponential integrate-and-fire (AdEx) as well as inhibitory (negative excitation) neurons. We also chose networks that capture different properties found in SNN, such as deep versus shallow, feed-forward versus lateral versus recurrent, and dense versus sparse architectures.

1) SRNN is a spiking recurrent neural network architecture [14]. It contains feed-back connections per layer. It applies LIF and AdEx neurons. SRNN uses trainable time constants which are shown to increase learning capability. However, it increases the memory footprint and bandwidth during operation and disables programmability of the exponential decay hardware. We tested this network on one of its reported benchmark applications; the Spiking Heidelberg Dataset [15].

2) LSNN is a sequential spiking neural network [12]. The architecture contains LIF and AdEx as well as inhibitory neurons. The architecture is relatively shallow and lateral. The authors also apply a rewiring technique, Deep R [12], which prunes connections. We tested this network on one of its reported benchmark applications; Sequential-MNIST [16].

3) [13] is an SNN architecture able to train very deep networks like VGG [17] and ResNet [18] on relatively advanced applications in the SNN world. It avoids the expensive back-propagating *through time* technique used in Recurrent Neural Networks and most SNNs trained with gradient descent [10]. We tested this architecture on one of its reported benchmarks; CIFAR-10 dataset [19] on VGG9.

B. Results

We tested all our ten implementations on all three benchmark networks and applications. Table I summarizes the inference accuracy of all implementations on all benchmarks.

For LUT implementations, 2.6b performed well while 2.5b and 2.4b suffered performance drops on SRNN and LSNN benchmarks. For PWL implementations, PWL25 and PWL50 performed well while PWL100 and bi-linear suffered from significant performance drops across all applications. For LOG implementations, LOG2 and LOG3 performed well while LOG1 suffered minor performance drops on SRNN and LSNN benchmarks. When performance degraded, fine-tuning (i.e, retraining) recovered accuracy loss in little time ($\approx 1epoch$). Since these approximations are event-based (ISI dependent),

TABLE I: Inference accuracy of implementations on benchmarks applications.

Exp Function	LSNN S-MNIST	SRNN SHD	[13] Top 5 CIFAR-10	[13] Top 1 CIFAR-10
Baseline	93.00%	84.72%	99.09%	90.14%
LUT 2.6b	93.00%	84.32%	99.09%	90.14%
LUT 2.5b	91.00%	80.12%	99.09%	90.16%
Fine-tuned LUT 2.5b	92.90%	84.2%	—	—
LUT 2.4b	73.50%	80.43%	99.14%	90.14%
Fine-tuned LUT 2.4b	92.50%	83.25%	—	—
PWL25	92.90%	84.54%	99.15%	90.26%
PWL50	92.90%	84.23%	98.98%	90.38%
PWL100	64.60%	82.16%	96.36%	87.94%
Fine-tuned PWL100	93.00%	85.63%	99.05%	90.15%
Bi-linear	52.30%	81.32%	93.51%	81.19%
Fine-tuned Bi-linear	92.70%	84.63%	99.10%	90.30%
LOG3	93.40%	84.98%	99.07%	90.25%
LOG2	92.60%	85.34%	99.07%	90.30%
LOG1	89.50%	79.59%	99.35%	89.96%
Fine-tuned LOG1	92.70%	84.76%	—	—

TABLE II: Hardware evaluation of implementations.

Implementation	Delay (ns)	Area (μm^2)	Power (μW)	PDP (fJ)
Pre-processing	1.2	360	19	23
Baseline [3]	5.75	24060	97	558
Baseline [3] P	4.55	23700	78	355
LUT 2.6b	5.75	6060	88	507
LUT 2.6b P	4.55	5700	69	314
LUT 2.5b	5.75	5760	87	501
LUT 2.5b P	4.55	5400	68	309
LUT 2.4b	5.75	5460	86	496
LUT 2.4b P	4.55	5100	67	305
PWL25	4.85	4540	64	313
PWL50	**4.45**	4000	**67**	**300**
PWL100	4.25	4000	65	279
Bilinear	4.25	4000	62	266
LOG3	7.2	3160	114	820
LOG3 P	6	2800	95	570
LOG2	5.2	**2560**	85	445
LOG2 P	4	2200	66	265
LOG1	**3.2**	**1300**	**43**	**137**
LOG1 P	2	940	24	47

they are ill-suited for CPU/GPU execution and are slower to simulate. However, it is fine for training only a few epochs. Hence, we conclude that approximations should be applied after training and with the help of quick fine-tuning.

VI. Hardware Evaluation

Designs were implemented in C++ using Vivado HLS. We used Cadence Genus for RTL synthesis. Post-synthesis results were obtained using 45nm GDPK technology library in the slow corner at 1.2V using 16-bit fixed point data type for the network and exponents and 8-bit integer data type for the ISI.

Table II summarizes the hardware costs of all our implementations and TS-EFA [3] as a baseline, where 'P' refers to programmable designs and pre-processing is shown separately. We highlight in bold the best implementation that preserves accuracy and the best implementation regardless of accuracy.

Our results show that LOG2 and PWL50 implementations are the best implementations that preserve inference accuracy. However, programmability favors LOG2. LOG1 outperforms all other implementations, but requires fine-tuning.

VII. Conclusions

In this paper, we have presented different approximations for exponential decay, including a novel implementation. We have systematically implemented different levels of precision and tested their performance on benchmark networks. Logarithmic Scaling outperforms known approximations methods while being easy to program to specific time constants without any performance variation. Our work concluded that an exact calculation is an overkill and networks can be error resilient and can very quickly fit (i.e, be fine-tuned) to other decay approximations. Our work also encourages the use of simplified decay functions (e.g, linear decay function [20]) after training. The implications of using different leak functions other than the exponential decay function require more in-depth analysis into the effects of decay on network performance and robustness [21].

References

[1] Wulfram Gerstner et al. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press, 2014.

[2] Z. Du et al. Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*.

[3] J. Kim et al. Ts-efa: Resource-efficient high-precision approximation of exponential functions based on template-scaling method. In *2020 21st International Symposium on Quality Electronic Design (ISQED)*.

[4] D. Wu et al. Seco: A scalable accuracy approximate exponential function via cross-layer optimization. In *ISLPED*, pages 1–6, 2019.

[5] M. Heidarpour et al. A cordic based digital hardware for adaptive exponential integrate and fire neuron. *IEEE Transactions on Circuits and Systems*, 2016.

[6] M. Davies et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

[7] J. Partzsch et al. A fixed point exponential function accelerator for a neuromorphic many-core system. In *ISCAS*, pages 1–4, 2017.

[8] X. Geng et al. Hardware-aware exponential approximation for deep neural network. *Asian Conference on Computer Vision*, 2018.

[9] I. M. Comsa et al. Temporal coding in spiking neural networks with alpha synaptic function. In *ICASSP*, 2020.

[10] Jacques Kaiser, Hesham Mostafa, and Emre Neftci. Synaptic plasticity dynamics for deep continuous local learning. *CoRR*, 2018.

[11] L. Zhang et al. Tdsnn: From deep neural networks to deep spike neural networks with temporal-coding. *AAAI*, 2019.

[12] Guillaume Bellec, Wolfgang Maass, et al. Long short-term memory and learning-to-learn in networks of spiking neurons. *CoRR*, 2018.

[13] Chankyu Lee et al. Enabling spike-based backpropagation for training deep neural network architectures. *Frontiers in Neuroscience*, 2020.

[14] B. Yin, F. Corradi, and S. Bohté. Effective and efficient computation with multiple-timescale spiking recurrent neural networks. *ICONS*, 2020.

[15] B. Cramer, F. Zenke, et al. The heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[16] Yann LeCun et al. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, number 11, 1998.

[17] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR 2015*.

[18] K. He et al. Deep residual learning for image recognition. *CVPR 2016*.

[19] A. Krizhevsky et al. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[20] T. Liu et al. Fpt-spike: a flexible precise-time-dependent single-spike neuromorphic computing architecture. *CCF Trans. HPC*, 2020.

[21] S. Chowdhury et al. Towards understanding the effect of leak in spiking neural networks. *arXiv e-prints*, page arXiv:2006.08761, 2020.