This article has been accepted for publication in a future proceedings of this conference, but has not been fully edited. Content may change prior to final publication.
Citation information: DOI10.1109/COINS61597.2024.10622154.10222486, 2024 IEEE International Conference on Omni-layer Intelligent Systems (COINS)

Exploring 8-bit Arithmetic for Training Spiking Neural Networks

Fernandez-Hart, T.
Brunel University, CEDPS,
Department of Electronic and
Computer Engineering,
UB8 3PH, Uxbridge, UK
Email: Tim.Fernandez-Hart@brunel.ac.uk

Kalganova, T.
Brunel University, CEDPS,
Department of Electronic and
Computer Engineering,
UB8 3PH, Uxbridge, UK
Email: Tatiana.Kalganova@brunel.ac.uk

Knight, James C.
University of Sussex,
School of Engineering and Informatics,
Brighton, BN1 9QJ, UK
Email: J.C.Knight@sussex.ac.uk

***Abstract*—Spiking Neural Networks (SNNs) promise improvements in biological plausibility, better noise tolerance and enhanced temporal capabilities compared to other Artificial Neural Networks (ANNs). Furthermore, if they are deployed on dedicated neuromorphic platforms, they can offer large energy efficiency gains over ANNs. However, many SNNs are currently trained using standard Back Propagation Through Time on GPUs before deployment onto dedicated hardware. While there has been significant work in performing inference with reduced precision SNNs, its use during training remains under explored. This study investigates posit arithmetic, a recently developed numerical format, to asses its capabilities for training SNNs on future posit-enabled accelerators. We compare 8-bit posit and floating-point arithmetic against a 32-bit floating-point reference and demonstrate that, when quantising all training elements, 8-bit posits achieve performance comparable to 32-bit floating-point (FP) on both MNIST and the event-based SHD datasets. These findings suggest that posit arithmetic offers a promising avenue for future development of hardware for training SNNs.**

I. Introduction

Spiking Neural Networks (SNNs) represent a novel approach to artificial intelligence, drawing closer inspiration from the structure and function of biological neural networks than traditional Artificial Neural Networks (ANNs). Unlike ANNs that operate on continuous values, SNNs use discrete spikes to transmit information, offering several potential advantages. such as reduced power consumption, better noise resilience, improved temporal processing, and even the ability to self repair [11]. Their efficiency partly stems from the sparse spatial and temporal nature of these spikes, wherein neurons remain mostly inactive, consuming minimal power. This makes them well-suited for inference on resource-constrained edge computing devices [17] as well as specialised neuromorphic hardware. However, dedicated accelerators for training SNNs have not yet been developed so it's common practice to initially train SNNs on GPUs before transferring them to the target end device.

Despite their potential, SNNs face certain challenges, particularly in training processes. The discrete nature of spikes complicates training methodologies, as conventional gradient descent techniques cannot be directly applied due to non-differentiability. Consequently, alternative methods have been proposed, with one of the most prominent being surrogate gradient descent [15]. In this technique spikes are used in the forward pass but the heaviside function is replaced in the backwards pass with a differentiable, surrogate function. This technique has seen SNNs now begin to rival ANNs in some tasks but the use of GPUs for training remains resource intensive and, similarly to ANNs, is a limiting factor in model size and energy usage.

Using reduced precision arithmetic is a promising approach to address this limitation. This technique employs numerical representations with lower precision than the commonly used 32-bit Floating-Point numbers (FP32), resulting in reduced memory usage and computational complexity. Consequently, it lowers energy consumption and speeds up computations. However, it introduces rounding and truncation errors. The dominant format for representing real numbers in computers is the IEEE-754 Floating-Point standard [1], which offers a wide dynamic range but has been criticized for its multiple special conditions, wasteful reserved bit patterns, and numerous rounding modes. Posit arithmetic [6], introduced in 2017 as a potential hardware-friendly alternative to FP, incorporates unique regime bits in addition to sign and fraction bits. While posit arithmetic involves extra overhead due to dynamic regime bit lengths, it simplifies special conditions and maximizes bit pattern usage, potentially offering higher precision for a given bit width compared to FP.

Posits number systems are defined by posit$<n, es>$, where n is the total bits and es is the maximum bits used for the exponent. The regime bits encode values in runs of zeros or ones following the rule in Equation 2 and can be $n-1$ bits long. The regime bits are followed by the exponent bits (without bias) and if there are any remaining bits these are used for the fraction which always has an implicit 1, negating the need for subnormal numbers. Several studies have shown that for a FP number n bits wide, equal or better accuracy can be achieved by using a m bit wide posit (where $m \leq n$) [9]. The posit standard also includes a large accumulator register to enable exact dot products to be calculated called the *quire* [5]. It is sized at $16n$ and, is not always implemented [13] in hardware designs due to its size which can occupy half the area of the PAU. Moreover, posit arithmetic sensibly reduces the number of undefined values (NaN) to one, termed NaR

This article has been accepted for publication in a future proceedings of this conference, but has not been fully edited. Content may change prior to final publication.
Citation information: DOI10.1109/COINS61597.2024.10622154.10222486, 2024 IEEE International Conference on Omni-layer Intelligent Systems (COINS)

(Not-a-Real).

$$p = (-1)^s \text{useed}^r 2^e \left(1 + \frac{f}{2^{\text{fs}}}\right) \tag{1}$$

$$r = \begin{cases} -k & \text{if } R_0 = 0 \\ (k-1) & \text{if } R_0 = 1 \end{cases} \tag{2}$$

In this study, we use QTorch+ [8] to simulate 8-bit posit arithmetic for training SNN on two datasets. QTorch+ is based on a "two-kernel" model, where the matrix multiplication is performed in native precision (typically 32-bit) and the result is then quantised. This allows QTorch+ to simulate different configurations of posit and floating-point arithmetics among others. Furthermore, it is integrated into the PyTorch framework allowing control of the optimiser, gradient and loss values during training.

A. Related Work

To the best of our knowledge, this is the first study to evaluate posit arithmetic for training SNN. However, a few studies have previously explored using reduced precision floating-point arithmetic in addressing the same task. Shymyrbay et al. [12] proposed a novel gradient function to improve quatisation aware training, reporting significant memory savings without much drop in accuracy. However, when using their model during inference, scaling and biasing was required to compensate for the quantisation error. Lui and Neftci [10] demonstrated how a layer-wise Hessian trace analysis can quantify how a change in weights influences the loss. The authors claim that this metric can help with the intelligent allocation of a layer-specific bit-precision while training SNNs. However, they use a simplified neuron model, stochastic rounding, gradient scaling with a focus on fixed-point arithmetic.

Eshraghian and Lu [4] used an annealing algorithm to adjust the firing thresholds during training of binarised weights, balancing neuron's memory and firing rate. However, they leave the neuron state variables at full precision and needed an extensive hyperparameter optimisation prior to training.

Existing reduced precision SNN training methods often require significant hand-tuning of algorithms before training can occur. We propose a more general approach, that eliminates this need for researchers. Moreover, our work demonstrates how well a future 8-bit posit enabled accelerator would perform when training an SNN. Focusing on the most energy-intensive phase – training – we aim to use 8-bit values throughout the entire training pipeline. To achieve this, we compare the performance of two floating-point arithmetic systems: the standard IEEE-754 format and the recently developed posit arithmetic.

B. Contributions

- This work presents the first application of 8-bit posit arithmetic for training Spiking Neural Networks (SNNs) on both frame-based and event-based datasets. Our results demonstrate that SNNs trained with posit arithmetic achieve high accuracy, and that posits offer greater effectiveness compared to floating-point at the same bit width.
- We demonstrate that unlike 8-bit floating-point, 8-bit posits enable successful learning in SNNs without requiring additional efforts such as scaling, biasing, using non-compliant IEE754 formats, or mixed-precision/weights-only quantisation for both tasks considered.
- Furthermore, we use loss function selection as a method to explore training success for posit and FP arithmetic.

II. Methods

A. Datasets

We trained and tested our network using two different datasets: MNIST and the Spiking Heidelberg Digits (SHD). The MNIST dataset is a widely used benchmark for image classification, consisting of static, frame-based images of handwritten digits and, here, we treat the pixel intensity of each frame as a current injected into the first layer of spiking neurons. The SHD dataset [2] is a dedicated spiking dataset containing 10,420 spoken samples of the words 0-9 in English and German, from 12 participants. Each sample lasts between 0.24s and 1.17s and is encoded as spikes from 700 input channels generated from a model of the cochlea. For computational efficiency, we further temporally downsampled the SHD dataset into 50 time bins. Both datasets were processed and loaded using the SNN Tonic framework.

B. Hyperparameter Selection

Similarly to [4], hyperparameters were fine-tuned on the training sets across a range of FP8 and posit<8, es> configurations, adjusting both the exponent/mantissa combinations and es. This was done using Ray and HyperOpt with Asynchronous Successive Halving Algorithm to search the parameter space over 500 trials for each dataset-arithmetic combination. These aimed to minimise the loss over 20 epochs. The best hyperparameter values are shown in Table I.

Parameter	MNIST	SHD
Membrane decay	0.9146	0.9231
Dropout 1	0.0338	0.0931
Dropout 2	N/A	0.1132
es	3	2
Learning Rate	0.0095	0.0037
Slope	3.5857	1.0236
Threshold 1	0.3704	3.5484
Threshold 2	1.3444	4.2336
Threshold 3	13.710	N/A

TABLE I: Best hyperparameters for 8-bit posit arithmetic on SHD and MNIST.

C. Frameworks and Libraries

The PyTorch framework was used throughout with the library Qtorch+ [8] for quantisation and SNNTorch [3] for SNN simulation. As used in previous studies [10] QPytorch+ provides support for quantised training by means of simulating

This article has been accepted for publication in a future proceedings of this conference, but has not been fully edited. Content may change prior to final publication.
Citation information: DOI10.1109/COINS61597.2024.10622154.10222486, 2024 IEEE International Conference on Omni-layer Intelligent Systems (COINS)

reduced precision arithmetics including posit, floating-point and fixed-point. For the sake of training speed, the dot product in QTorch+ is done at 32-bits and then quantised to 8-bits. This design decision is a common choice for quantisation frameworks [16] and emulates a small quire register for posits.

SNNTorch [3] takes advantage of PyTorch's efficient GPU training for training SNNs. Although, compared to other libraries, it is more limited in the number of potential SNN parameters it can simulate. SNNTorch also allows training of both each neuron's membrane decay and threshold, albeit at the cost of extra computation and increased model size. Our preliminary tests showed that including these additional parameters did not significantly improve accuracy and so were omitted. Beta and layer thresholds were optimised as hyperparameters.

D. Network Training

All networks consisted of leaky integrate and fire (LIF) neurons. The training process was performed for a total of 200 epochs with the test set only used after hyperparameter tuning for testing at the end of each epoch. During the forwards pass the input to each neuron and its membrane voltage were quantised to the specified arithmetic for each layer. Surrogate gradient decent was used throughout with a fast sigmoid gradient function. The ADAM optimisation algorithm was used with $\beta_1 = 0.9$ and $\beta_2 = 0.999$, quantising the gradients, error, accumulator, weights and momentum at each step.

For the MNIST task a convolutional network with the data normalised between 0, and 1 and the fed as current input to the first convolutional layer (16C5), followed by a max-pool (MP2) and an LIF layer. This was repeated for the second convolutional layer using the spike output from the previous layer (64C5) followed by another max-pool and finally a fully connected classifier with 10 output LIFs.

For the SHD task, the data is already encoded as spikes across 700 input channels and similar to Eshraghian and Lu [4] this feeds into the first layer of 3000 LIF neurons. This LIF layer is fully connected to the output layer of 20 LIF neurons. Although others have used deeper networks for the SHD dataset, we chose this network structure for a better comparison with a recent work in quantised SNNs. Dropout layers were placed after each LIF neuron in the SHD network and after the final LIF neuron in the MNIST task.

E. Loss Function Testing

During initial testing we noted that loss function choice had a dramatic effect on training performance. Although several loss functions are available in SNNTorch, we chose two, ce_rate_loss () (CE-rate) and mse_count_loss() (MSE-count) to look at in more depth. CE-rate represents the spike rate cross entropy loss and is calculated by sequentially passing spikes from each time step through PyTorch's cross entropy loss function. It integrates both LogSoftMax and NLLLoss into a single function, aggregating the loss across time, encouraging the correct class to fire at every time step while suppressing firing from incorrect classes.

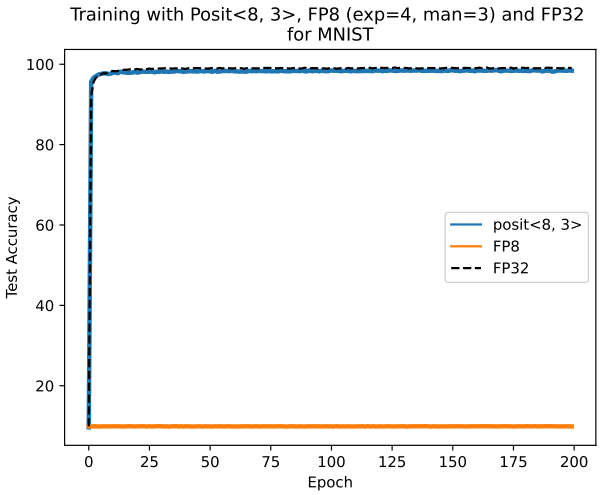

Fig. 1: Test set accuracy per epoch during training on MNIST for posit<8, 3>and FP8 with $exp = 4$ and $man = 3$.

The MSE-count loss function accumulates a spike count for each neuron over time and applies PyTorch's mean square error loss function with a target of $(\text{time steps} \times \text{correct rate})$ for correct classes and $(\text{time steps} \times \text{incorrect rate})$ for incorrect classes. We set $(\text{correct rate} = 0.9)$ and $(\text{incorrect rate} = 0.1)$ to maximise learning without depressing neuron firing overall.

Although it is common practice to use a cross entropy loss function for classification tasks we felt the inclusion of another, different loss function class was instructive in the understanding of how these functions interact with the underlying arithmetic formats.

III. Results

A. Frame-Based Dataset: MNIST

1) Training: As Figure 2 illustrates, Posit<8, 3> achieved an accuracy of 98.57%, largely mirroring the FP32 convergence rate and very close to its accuracy of 99.20%. whereas FP8 with a 4-bit exponent and 3-bit mantissa was not able to attain greater than chance level performance. Other FP8 configurations were tested during the hyperparameter sweep, such as 3-bit exponent and 4-bit mantissa and 5-bit exponent and 2-bit mantissa but no improvement in accuracy was seen.

2) Loss Function: Interestingly, we observed not only a significant effect on training with choice of loss function, but also an interaction between loss function choice and es value. Figure 2 shows the loss function CE-rate training normally if $es = 3$. However, when $es = 2$ only chance level performance is obtained. In contrast, performance when using the MSE-count loss function was not contingent on es choice.

This article has been accepted for publication in a future proceedings of this conference, but has not been fully edited. Content may change prior to final publication.
Citation information: DOI10.1109/COINS61597.2024.10622154.10222486, 2024 IEEE International Conference on Omni-layer Intelligent Systems (COINS)

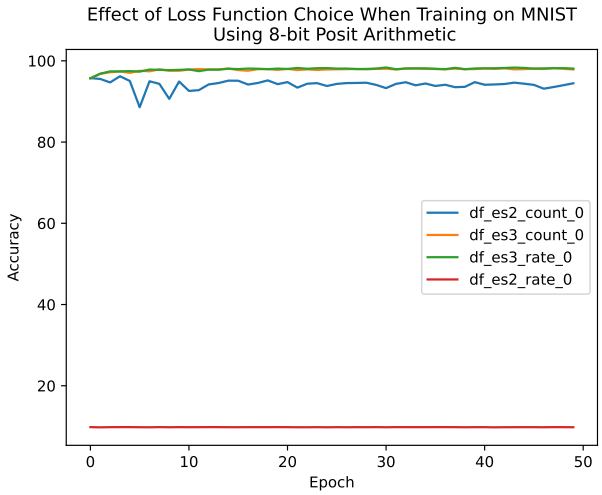

Fig. 2: The effect on accuracy of loss function choice when training on the MNIST dataset using 8-bit posits.

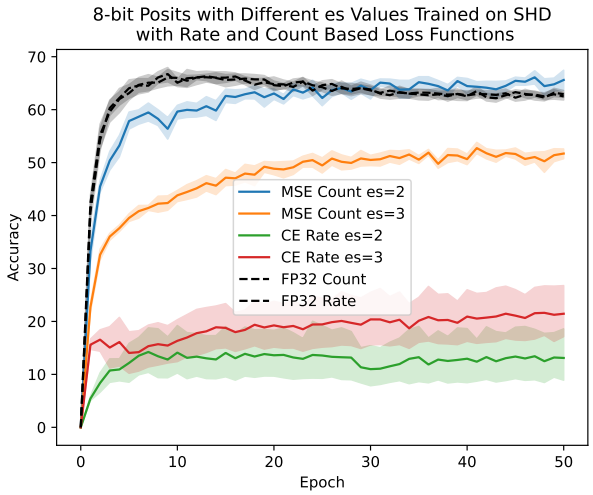

Fig. 4: Comparing the accuracy of two different loss functions used to train on SHD with posit<8, 2>and FP32.

B. Event-Based Dataset: SHD

1) Training: As illustrated in Figure 3, posit<8, 3> was able to learn without any degradation in performance compared to the FP32. After 200 epochs of training, posit<8, 3> achieved a mean accuracy over 10 runs of 65.92% with a standard deviation of 6.15 and a peak accuracy of 72.17%. This compares favourably to FP32 case that achieved a mean of 61.93% with a slightly higher standard deviation of 9.69 and a peak accuracy of 70.19%. Similarly to MNIST, despite an extensive hyperparameter search, FP8 proved inadequate for learning the SHD classification task.

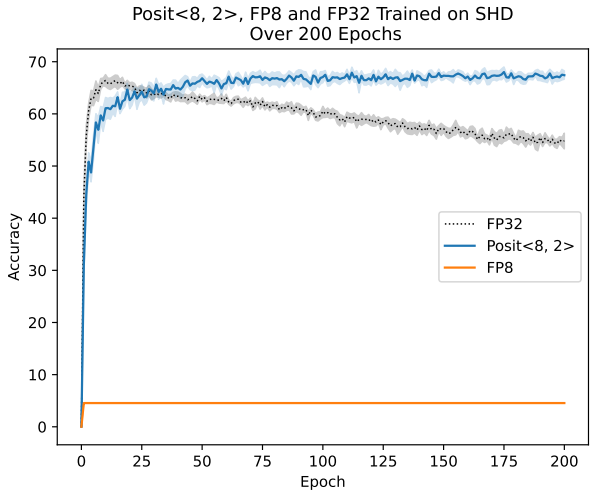

Fig. 3: Test set accuracy during training with FP8 and posit<8, 2>. All used mse_count_loss as the loss function. The central estimate is the mean over 10 runs with 95% CI shown.

2) Loss Function: Figure 4 illustrates the training comparison between using either MSE-count or CE-rate as the loss function, with 8-bit posits, where es=2 or es=3. The results are summarised over 10 runs, using the mean as the central estimate with 95% confidence intervals (CI).

The accuracy of the MSE-count loss function is generally higher than the CE-rate loss function for all es values. Specifically, the MSE-count function achieves a maximum accuracy of 70.40% for es=2, while the CE-rate loss function only reaches 37.01% for es=2 with a mean of only 12.68%. CE-rate performed slightly better for es=3 with a peak accuracy of 35.64% (mean=21.8%, s.d.=3.79), but MSE-count fared worse with es=3 with a peak accuracy of 55.96% (mean=45.85%, s.d.=8.73). Additionally, CE-rate tended to be more variable with overlapping of the 95% confidence interval bands. For comparison, both loss functions achieved a similar accuracy of 69.83% for CE-rate and 70.19% for the MSE-count when using FP32 arithmetic.

IV. Discussion

While previous studies have demonstrated the feasibility of using 8-bit floating-point arithmetic for learning tasks such as MNIST classification, significant interventions including complex algorithms, pre-processing of data, or mixed precision arithmetic were often required [14]. In contrast, posit arithmetic demonstrates potential for efficient and straightforward SNN training by enabling successful learning without additional interventions or mixed-precision techniques. However, selecting the optimal es value and appropriate loss function remain crucial. Additionally, in the MNIST task, the reference FP32 case suffers greatly from over fitting in this case which is probably a reflection of the wide and shallow network architecture. This would likely be solved with a deeper, narrower architecture.

One reason for training failure in FP8 is the representation of very small gradients. For example, in the SHD case where no output spikes are recorded during the first batch the loss is -2.9957. From here, the gradients for the first weights are

This article has been accepted for publication in a future proceedings of this conference, but has not been fully edited. Content may change prior to final publication.
Citation information: DOI10.1109/COINS61597.2024.10622154.10222486, 2024 IEEE International Conference on Omni-layer Intelligent Systems (COINS)

given by Equation 3, where L is the loss, S the spike output, U the membrane voltage, I the input current and W the weight. Given that $\frac{\partial U}{\partial I} = 1$ and $\frac{\partial I}{\partial W} = X$ Equation 3 simplifies.

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial S}\frac{\partial S}{\partial U}\frac{\partial U}{\partial I}\frac{\partial I}{\partial W} \tag{3}$$

If our loss function is CE-rate (logSoftMax ($\hat{y}$) followed by NegativeLogLiklihoodLoss (NLLLoss)), then $\frac{\partial L}{\partial S}$ for a correct class is given by Equation 4 and an incorrect class is 0.05. These are then divided by the number of time steps 50 giving -0.019 for a correct class and 0.001 for an incorrect class.

$$\frac{\partial L}{\partial S} = \frac{\partial NLLLoss}{\partial S} = (\hat{y}_{correct} = \hat{y}_{correct}) \tag{4}$$

$$= \frac{1}{20} - 1 = -0.95 \tag{5}$$

The fast surrogate gradient has a partial derivative $\frac{\partial S}{\partial U}$ is given in Equation 6.

$$\frac{\partial S}{\partial U} = \frac{1}{(1+\beta|U|)^2} \tag{6}$$

With representative values $U = -4.35$ and $\beta = 0.923$ $\frac{\partial L}{\partial W}$ becomes -2.69×10^{-6}. This gradient value rounds to zero in FP8 if the exponent has 5, 4, or 3 bits.

	posit<8, 2>	posit<8, 3>	FP8 (e=5, m=2)	FP8 (e=4, m=3)
Range Extent	± 16777216	± 281474976710656	± 57344	± 240
Total Number	255	255	247	239

TABLE II: A table comparing the total number of representable values for four arithmetic systems. The count includes only one value of zero, and excludes all NaN and NaR.

For posit arithmetic the interaction between loss function and es are equally intriguing. One possible explanation for these observations is best understood by first examining the count, range and distribution of values in both posit and FP representations. Any 8-bit posit configuration represents $2^8 - 1 = 255$ values with the smallest positive representable value in posit<8, 2> being 5.96×10^{-8}, and 3.55×10^{-15} for posit<8, 3>. In contrast, FP8 with e=5 and m=2 can represent just 247 values with the smallest positive representable value 1.53×10^{-5} and the largest representable value being 57344. The reason FP8 represents fewer values is because it has many more NaN, although these can be minimised by reducing the number of mantissa bits. Table II shows the range of representable values for four arithmetic systems posit<8, 2>, posit<8, 3>, FP8 (e=5, m=2) and FP8 (e=4, m=3) and by excluding NaN, -0 and NaR, it also shows the number of values that each system can actually represent. Table II also shows that posits have much larger dymanic range. However, the distribution of these values is markedly different between arithmetics, as shown in Table III and Figure 5.

Table III categorises the count of representable values by interval, indicating that within the range of ±50, a posit configuration can represent a greater number of values than FP8. Indeed, only outside the interval of $\pm \geq 71$ does FP8 (e=4, m=3) encompass more of these exact representations, albeit marginally. Figure 5 shows this visually, demonstrating that in the interval (-1×10^{-6} and 1×10^{-6}) there are 5 representable values using posit<8, 2>, 25 values using posit<8, 3>, but only 1 value when using FP8.

This may have significant implications during the update step of training. For example, in the SHD task, after one batch of 128 samples, the gradients (g) of the weights in the classification layer have an approximate minimum value of -0.00004327 a maximum of 0.00000528 (mean=-0.00000132, s.d.=0.00000394). The weights in this layer have a minimum value of -0.03124 a maximum of 0.03125 (mean=0.00002, s.d.= 0.01798). In the best case scenario, using the ADAM optimiser with $\beta_1 = 0.9$ and $\beta_2 = 0.999$ then the first moment (m_1) will be 0.000000528, which quantises to 9.54×10^{-7} in posit<8, 2>. The second moment relies on the square of the gradient (g^2) and this saturates the arithmetic to its lowest possible representable value 5.96×10^{-8}. However, using posit<8, 3>this becomes 1.46×10^{-11}, with two further values below that if required. For comparison, FP8 underflows to 0.0 even for m_1. Therefore, these intermediate values will likely have a large impact on such a system and may explain the interdependence of es and loss function choice.

Range	posit<8, 2>	posit<8, 3>	FP8 (e=5, m=2)	FP8 (e=4, m=3)
± 0.000001	5	25	1	1
± 0.001	25	57	41	1
± 0.1	73	101	93	58
± 1	129	129	121	114
± 2	145	137	129	130
± 10	181	155	147	166
± 50	205	173	165	201
±100	213	181	173	217

TABLE III: A table showing the count of values that are exactly representable for four arithmetic configurations.

This highlights two important aspects of how posit arithmetic functions. Firstly, posits avoid underflow by saturating to their lowest representable value. This allows calculations to continue without introducing a potentially catastrophic 0.0. Although, in the posit<8, 2>above, avoiding underflow was not sufficient to allow training to continue. The second aspect of posit arithmetic is the tapered precision which stems from the way values are added geometrically.

V. Conclusion

Here we showed that posit arithmetic is able to train a SNN with only 8-bits. Although some variability was noted, posit arithmetic performed best when $es = 3$ for the SHD dataset but $es = 2$ for the MNIST dataset. The posit standard dictates $es = 2$ for all n [5] but some implementations allow runtime switching between the two levels Tiwari et al. [13]. We also demonstrated the interplay between arithmetic and loss function providing some insight into the values required

This article has been accepted for publication in a future proceedings of this conference, but has not been fully edited. Content may change prior to final publication.
Citation information: DOI10.1109/COINS61597.2024.10622154.10222486, 2024 IEEE International Conference on Omni-layer Intelligent Systems (COINS)

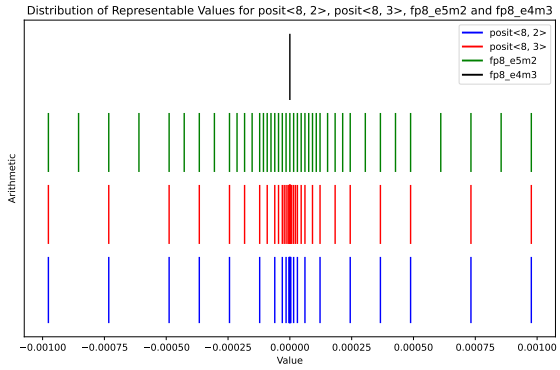

Fig. 5: The distribution of exactly representable values for four 8-bit arithmetics, posit<8, 2>, posit<8, 3>, FP8(e=4, m=3) and FP8(e=5, m=2) exponent and 2-bit mantissa.

for successful training and why some arithmetic configurations work better than others.

However, this study suffered from some limitations, firstly our SHD network architecture was likely not deep enough and was also too wide as demonstrated by the FP32 over fitting. Additionally, we only investigated one optimiser algorithm, which may also have a significant impact on the results. Furthermore, we did not explore using gradient and loss scaling which constitute an inexpensive bit-shift if kept to shift $= 2^n$ and have been used successfully in other studies Ho et al. [7].

Nonetheless, posit arithmetic is an intriguing and powerful new arithmetic that clearly warrants further investigation.

Acknowledgements

JK is funded by EPSRC grant numbers EP/V052241/1 and EP/S030964/1.

References

[1] IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84, July 2019. doi: 10.1109/IEEESTD.2019.8766229.

[2] B. Cramer, Y. Stradmann, J. Schemmel, and F. Zenke. The Heidelberg Spiking Data Sets for the Systematic Evaluation of Spiking Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 33: 2744–2757, 2019. doi: 10.1109/TNNLS.2020.3044364.

[3] J. Eshraghian, M. Ward, E. Neftci, X. Wang, G. Lenz, G. Dwivedi, Bennamoun, D. S. Jeong, and W. Lu. Training Spiking Neural Networks Using Lessons From Deep Learning. *Proceedings of the IEEE*, 111:1016–1054, 2021. doi: 10.1109/JPROC.2023.3308088.

[4] J. K. Eshraghian and W. D. Lu. The fine line between dead neurons and sparsity in binarized spiking neural networks, Jan. 2022.

[5] J. Gustafson, G. Bohlender, S. Y. Chung, and V. Dimitrov. Standard for Posit™ Arithmetic (2022). Available online at https://posithub.org/docs/posit_standard-2.pdf, Mar. 2022. Accessed: 01/02/2023.

[6] J. L. Gustafson and I. T. Yonemoto. Beating Floating Point at its Own Game: Posit Arithmetic. *Supercomputing Frontiers and Innovations*, 4(2):71–86, Apr. 2017. doi: 10.14529/jsfi170206.

[7] N.-M. Ho, D.-T. Nguyen, H. D. Silva, J. L. Gustafson, W.-F. Wong, and I. J. Chang. Posit Arithmetic for the Training and Deployment of Generative Adversarial Networks. In *2021 Design, Automation & Test in Europe Conference & Exhibition*, pages 1350–1355, Feb. 2021. doi: 10.23919/DATE51398.2021.9473933.

[8] N.-M. Ho, H. De Silva, J. L. Gustafson, and W.-F. Wong. Qtorch+: Next Generation Arithmetic for Pytorch Machine Learning. In *Next Generation Arithmetic*, volume 13253, pages 31–49, Cham, 2022. Springer International Publishing.

[9] J. Hou, Y. Zhu, S. Du, and S. Song. Enhancing Accuracy and Dynamic Range of Scientific Data Analytics by Implementing Posit Arithmetic on FPGA. *J Sign Process Syst*, 91(10):1137–1148, Oct. 2019. doi: 10.1007/s11265-018-1420-5.

[10] H. W. Lui and E. Neftci. Hessian Aware Quantization of Spiking Neural Networks, Aug. 2021. URL http://arxiv.org/abs/2104.14117. arXiv:2104.14117 [cs].

[11] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank. A Survey of Neuromorphic Computing and Neural Networks in Hardware. *arXiv:1705.06963 [cs]*, May 2017. URL http://arxiv.org/abs/1705.06963. arXiv: 1705.06963.

[12] A. Shymyrbay, M. E. Fouda, and A. Eltawil. Low Precision Quantization-aware Training in Spiking Neural Networks with Differentiable Quantization Function, May 2023. URL http://arxiv.org/abs/2305.19295. arXiv:2305.19295 [cs].

[13] S. Tiwari, N. Gala, C. Rebeiro, and V. Kamakoti. PERI: A Configurable Posit Enabled RISC-V Core. *ACM Transactions on Architecture and Code Optimization*, 18 (3):25:1–25:26, Apr. 2021. doi: 10.1145/3446210.

[14] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan. Training Deep Neural Networks with 8-bit Floating Point Numbers. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[15] F. Zenke and S. Ganguli. SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks. *Neural Computation*, 30(6):1514–1541, June 2018. ISSN 0899-7667. doi: 10.1162/neco_a_01086.

[16] T. Zhang, Z. Lin, G. Yang, and C. De Sa. QPyTorch: A Low-Precision Arithmetic Simulation Framework, Oct. 2019.

[17] G. Zhuang, Z. Bing, Z. Zhou, X. Yao, Y. Huang, K. Huang, and A. Knoll. An Energy-Efficient Lane-Keeping System Using 3D LiDAR Based on Spiking Neural Network. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages

4763–4769, Detroit, MI, USA, Oct. 2023. IEEE. doi: 10.1109/IROS55552.2023.10342044.