BitFit: Bitstream-Aware Training for Stochastic Neural Networks

Nitya Joshi
University of Wisconsin-Madison
Madison, Wisconsin, USA
njoshi26@wisc.edu

Kyle Daruwalla
Cold Spring Harbor Laboratory
Cold Spring Harbor, New York, USA
daruwal@cshl.edu

Mikko Lipasti
University of Wisconsin-Madison
Madison, Wisconsin, USA
mikko@engr.wisc.edu

Abstract

Stochastic computing (SC) is an unconventional computing paradigm with randomized streams of bits as its data representation. SC offers higher power and energy efficiency over traditional binary systems, making it an attractive option for compute-heavy workloads such as deep learning. Unfortunately, mapping floating-point algorithms to SC imposes specific constraints on numerical values, and a conventionally trained neural network often violates these constraints. The end result is a more efficient, but inaccurate SC network. Our work presents BitFit: a framework for training convolutional neural networks that minimizes performance degradation when moving from floating-point training to SC-based inference. To the best of our knowledge, BitFit enables the first demonstration of a deep neural network deployed in a streaming, end-to-end SC substrate that performs well with relatively short bitstreams.

1 Introduction

Stochastic computing (SC) is an unconventional computing paradigm with randomized streams of bits as its data representation [9]. Any hardware operation on stochastic bitstreams needs to process only a single bit at a time, which can often be done with simple circuits operating in a streaming fashion. This eliminates the need for much of the sequential logic associated with von Neumann computing, offering higher power and energy efficiency over traditional binary systems. Applications well-suited to SC must be robust to the approximate nature of randomized bitstreams—as such, neural networks are a natural fit for SC implementation.

An unexpected challenge faced by stochastic implementations of neural networks is the adjustment of model parameters trained in floating-point for the SC realm. Prior work focuses on converting the floating-point model to bitstreams by scaling parameters and network outputs after training, or by clipping them while training to force the optimizer to keep them in a permissible range. Instead, a training scheme that always takes SC constraints into consideration can significantly improve the performance of neural networks when moving from floating-point to SC. In this work, we present BitFit: a training scheme that creates models tailor-made for fully end-to-end SC deployment.

2 Motivation

The simplest strategy for utilizing the low energy benefits of SC while also having an accurate network has been to design hybrid systems operating with both stochastic and binary/floating-point numbers [3, 15]. But converting in and out of SC consumes significant energy, and any non-SC operators require orders of magnitude more power [4, 5]. Furthermore, while Li et al [15] implement a slightly deeper network than LeNet5, a major gap in the space of SC-CNN exploration has been a demonstration of an end-to-end SC-CNN that retains performance for deeper networks.

Stochastic bitstreams are samples from a Bernoulli distribution whose mean encodes an underlying real-valued number. As a result, any number in an SC circuit must remain in the $[-1, +1]$ interval. Values that try to exit this interval will saturate at the endpoints. Yu et al. [17] set out to create a SC-CNN by scaling parameters into the non-saturating range—a floating-point model's parameters can be scaled down on a per-layer basis to avoid saturation with a final upscaling at the end to preserve model behavior. While this strategy can be effective for shallow networks, the effects of scaling degrade accuracy as the number of layers increases. After scaling down a given layer's parameters, its output is now a scaled version of the original output, forcing the subsequent layer's parameters to also be scaled accordingly. Additionally, the subsequent layer will induce its own scaling based on its out-of-range parameters. As a result, layers deep within a network are scaled by an extremely large factor. The net effect is a post-scaled network with parameters whose magnitudes are extremely small. Unfortunately, the number of timesteps required to represent a number as a stochastic bitstream increases as the magnitude decreases [5]. Shortening the bitstream length to an acceptable value greatly impacts the final accuracy. Conversely, using a sufficiently long bitstream length would end up hurting any energy benefits obtained when moving to SC. Finally, due to a final step of output upscaling at the end, any small errors introduced due to approximate nature of SC can severely impede accurate inference.

Frasser et al. propose a training scheme in [7] that adds discrete steps in the training loop to quantize, normalize and clip the weights, then perform an additional weight update based on these transforms in order to stay within the target parameter range for SC. Frasser's evaluation [8] applies this approach to a shallow LeNet5 network, so the issue of saturated activations is not as evident, since that becomes a much bigger issue with deeper networks. Similar quantization-aware training methods, first introduced in 1990 [6], and implemented in frameworks such as TensorFlow-Lite [13, 16] can

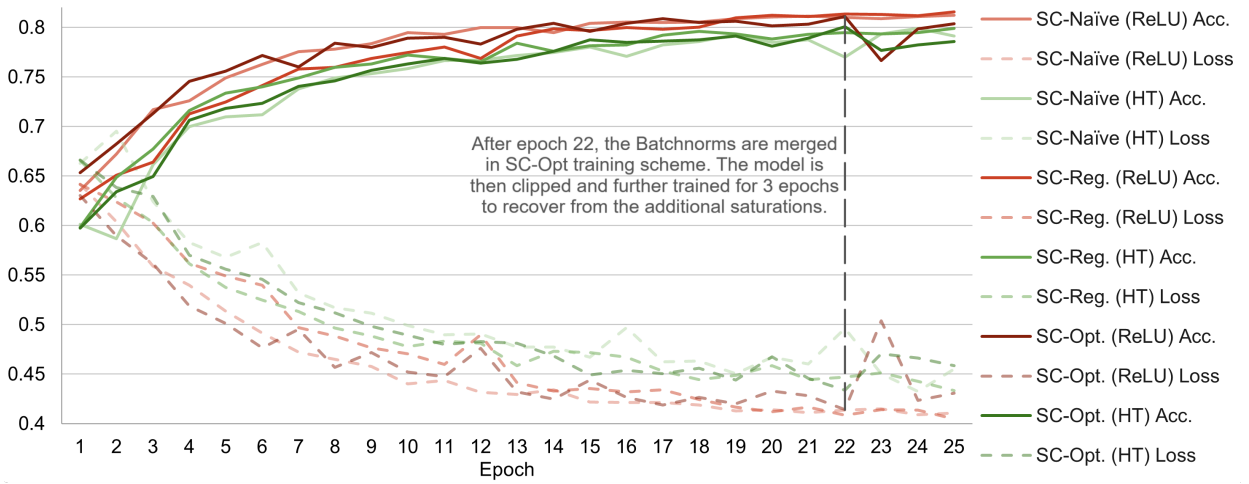

Figure 1. Validation accuracy and loss for SC- Naive, SC-regularized (SC-Reg.), and SC-Optimized(SC-Opt.) networks with ReLU and hardtanh (HT) as activation functions for training in floating point.

improve performance of quantized models when compared to just using post training quantization. While the latter is simpler to implement, the former enables models to attain better accuracy within the restrictions of quantization. BitFit achieves the best of both for stochastic constraints, providing a bitstream-aware training scheme that preserves model accuracy, while also being fairly straightforward to implement, and avoiding the memory overhead of a duplicated model (forward-discretized, backwards-continuous) needed for prior approaches [6, 13].

3 BitFit Training Scheme

In bipolar SC, model parameters can belong to the range [-1, 1] and anything beyond that saturates to -1 or 1. While clipping the parameters restricts them to the appropriate range, it can often result in much slower training—the weight updates seek to increment the weights only to get clipped at the end of an epoch. In our experiments, the non-ideal choice of weights in this process can also result in the model converging erratically or sometimes failing to converge, making the process both inefficient and ineffective. In BitFit, we penalize larger coefficients in the following way during training itself, to obtain a model directly compatible with this range, without needing any post-training interventions:

1. Custom regularization

Generally, adding a L1 or L2 norm to the training objective helps avoid overfitting. We utilize a custom regularization component to instead promote fitting to SC range constraint. We include the following regularization component in the loss function in the training scheme:

$$\text{Regularization(model)} = \sum_i \max(|\theta_i| - 1, 0)$$
$$\forall \theta \in \text{model parameters}$$

This function nudges the model into maintaining accuracy while taking our target range into account. Unlike Frasser et al [8], instead of having an explicit step after weight updates to take care of quantization, this regularization enables a pull toward the limited range to be directly integrated into the weight updates. With only the saturated parameters being penalized in the loss function, accuracy is retained when moving from floating point training to SC model inference. Even for scaled SC-CNNs, we will show that this step helps decrease the scaling required significantly.

2. Clipping before training without batchnorms

While this encourages all parameters to stay within $[-1, +1]$ it does not totally eliminate saturation within the model. During the conversion to SC, a model's batch normalization layers are merged into the previous convolution layer. At this stage, a few hidden saturated parameters can surface. While one can just clip these saturated parameters in a non-scaled SC-CNN, accuracy can suffer quite a bit even with a few of such saturated values. Instead of taking the accuracy hit upon merging, fine tuning the model for a few more epochs without the batch normalization layers present helps it recover from the hidden saturated parameters. As the custom loss function implemented can explode at this stage and sabotage model training, one time clipping of model parameters is done after merging the batch normalization layers. This is followed by fine tuning: training with the regularization for a few more epochs, just enough to recover from the saturations injected into the model on merging batchnorms.

3. Hardtanh Activation function

While ReLU has been a popular choice for an activation function and is compatible with scaling the network (as scaling by a factor retains its original effects in the model), it allows saturated positive values to continue to flow through the network. Replacing ReLU with hardtanh in the network creates a network that mimics the saturating behavior of bitstreams not only in the parameters but also in the neuron outputs to some extent. Allowing bitstream activations to saturate implicitly induces a non-linearity, so training with hardtanh allows the floating point model to accommodate this range inherent "activation function."

4 Methodology

For all the experiments, we used the TinyMLPerf Visual-WakeWords Dataset with 73,518 images in the training set and 35,901 images in the test set [1]. The model was trained in Julia and Flux.jl [2, 11, 12] and SC simulation done in BitSAD.jl [5]. Basic augmentations like scaling, rotating and zooming were applied to the training set. The loss function used was logit binary cross entropy. In the cases with custom regularization, it was added to the loss function without any extra scaling factor. The models were all trained for a total of 25 epochs, using Adam optimizer [14] with a learning rate of 0.001.

The model used was MobileNetV1 [10] (with a model scaling of 0.25) (Total parameter count 226, 849) with ReLU or hardtanh as activation functions as specified. The models were trained without any training enhancements (SC-naive), with just the regularization active (SC-regularized), and with both the regularization and fine-tuning without batch normalization (SC-optimized). In case of the SC-optimized models, training with batch normalization was done for 22 epochs followed by 3 epochs of training without batch normalization to maintain a total training duration of 25 epochs.

To evaluate the performance of the model in SC, the batch normalization layers are merged and then the network with unscaled model parameters are simulated for a bitstream of length 1000. These bitstream approximated coefficients are then used in the model along with activation functions clipped to $[-1, +1]$ to evaluate the validation set. All these changes help isolate the effects of bitstream saturation on SC-CNN performance over the floating-point accuracy during training. Further comparison was performed between performance of scaled SC-naive and SC-regularized ReLU models, against SC-optimized hardtanh model for varying bitstream cycle lengths.

Acronym	Description
SC-Naive	Naive floating-point training regime with no changes
SC-Reg.	Adding the custom regularizing term in the training scheme, but not merging the batchnorms during training
SC-Opt.	All the training changes enabled, creating models with all parameters in the $[-1, +1]$ range.

Table 1. Description of various training scheme acronyms

5 Results

In the validation accuracy over the epochs (Figure 1), it can be seen that ReLU always outperformed hardtanh as the activation function by a margin of around 2% in floating-point. There was an expected dip in accuracy on merging the batchnorm layers after epoch 22 in our SC-Opt. scheme, but the model is successfully able to recover from it quickly. The decrease in floating point accuracy between SC-naive and SC-Opt. was between 0.5-1%, but SC-Opt. scheme was able to successfully get rid of all over-saturated model parameters.

On tracking the amount of saturated parameters at the end of training (Figure 2), it can be seen that even a small amount of saturation can severely decrease accuracy, and

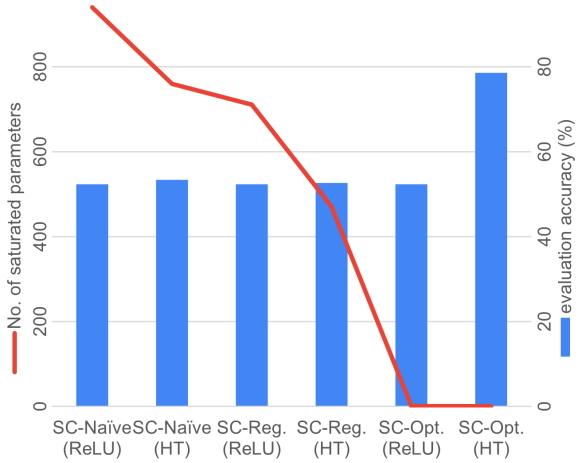

Figure 2. Accuracy and Saturation count for SC-Naive, SC-Reg., and SC-Opt. with ReLU and hardtanh (HT) as activation functions with capped activation functions.

accuracy can only be preserved if both model parameters and layer outputs are not saturated. In case of the SC-naive and SC-regularized models, around 0.2-0.4% of parameters were saturated yet all of them had an SC-evaluated accuracy of close to 50%, even though the floating-point accuracy prior to considering SC constraints was around 80%. Even the SC-optimized model with ReLU activation suffers a similar drop in accuracy despite reducing the number of saturated parameters to zero. This is due to the saturating positive activations. The SC-optimized scheme with hardtanh was successfully able to eliminate all saturated parameters with only a 1% decrease in floating-point accuracy. Due to the saturating nature of ReLU for positive numbers, SC-optimized ReLU suffered in bitstream evaluation due to the saturation in its activations. The effectiveness of SC-optimized scheme at getting rid of activation clipping can also be extended to any other similarly range restricted setups, as it provides a clear advantage over parameter clipping alone.

When comparing the performance of scaled ReLU (with SC-naive and SC-regularized training schemes) versus SC-optimized hardtanh for various cycle lengths, we find that the latter can preserve accuracy at much shorter bitstream lengths. For models trained without hardtanh, activation saturation can degrade performance even when no parameter saturation occurs. This can be compensated for by further increasing the scaling factor. We perform such an analysis in Figure 3. For ReLU-based networks, we recover performance by aggressive scaling but very long cycle counts. In contrast, SC-optimized hardtanh reaches close to its floating-point performance in a much shorter cycle length. This allows the early termination of SC to be fully utilizable. But if the number of cycles is not an issue, then scaled SC-regularized ReLU does provide higher accuracy at higher energy consumption. As its floating-point accuracy was higher than that of the SC-optimized hardtanh network, increasing bitstream length helps it reach better performance eventually.

Even when comparing SC-Naive and SC-Regularized, we see that our custom loss term greatly reduces the required scaling factor, allowing the SC-Regularized network to recover its performance faster as the bitstream length increases.

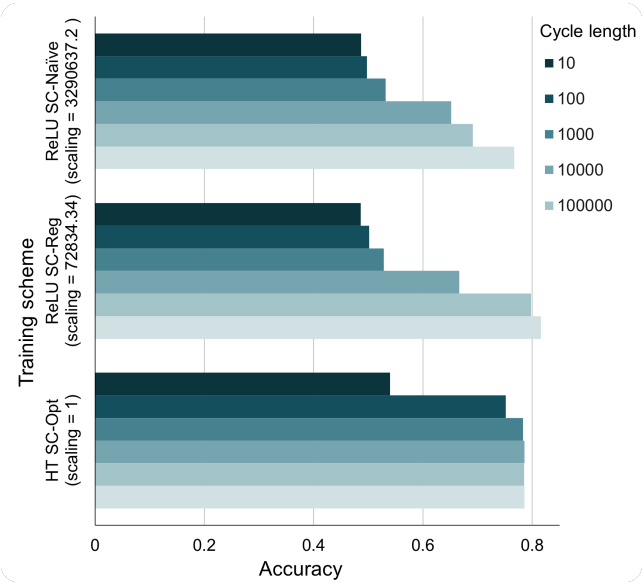

Figure 3. Accuracy for varying cycle lengths for scaled SC-Naive and SC-Reg with ReLU and SC-Opt with hardtanh.

6 Conclusion

In this work, we develop BitFit—a training scheme to produce models that do not need parameter scaling to deploy on SC substrates. We propose a novel regularization term that avoids saturating parameters, which coupled with additional clipped batch normalization-free fine-tuning completely eliminates saturation in model parameters and activations. Our regularization term could be applied in other range restricted applications, providing definite advantages over forced clamping during training. Our results show how eliminating over-saturated parameters and activations can produce accurate SC models without parameter scaling. The resulting SC-Optimized (with hardtanh) model is almost as accurate as the floating-point model, and it maintains that accuracy in the SC domain with relatively short bitstreams. The training scheme also helps scaled models by pushing the scaling factor down, and decreasing the necessary bitstream length for ReLU-based scaled networks as well. To further improve the training scheme, exploring more fine-grained interaction of saturating inputs with the model parameters and with each other could help obtain even higher performing SC models.

References

[1] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, et al. 2021. MLPerf Tiny Benchmark. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks* (2021).

[2] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. 2017. Julia: A fresh approach to numerical computing. *SIAM Review* 59, 1 (2017), 65–98. https://doi.org/10.1137/141000671

[3] Zhiyuan Chen, Yufei Ma, and Zhongfeng Wang. 2022. Hybrid stochastic-binary computing for low-latency and high-precision inference of cnns. *IEEE Transactions on Circuits and Systems I: Regular Papers* 69, 7 (2022), 2707–2720.

[4] Kyle Daruwalla, Heng Zhuo, and Mikko Lipasti. 2019. BitSAD: A Domain-Specific Language for Bitstream Processing. In *First ISCA Workshop on Unary Computing - June 2019*. Phoenix, AZ, USA.

[5] Kyle Daruwalla, Heng Zhuo, Rohit Shukla, and Mikko Lipasti. 2019. BitSAD v2: Compiler Optimization and Analysis for Bitstream Computing. *ACM Trans. Archit. Code Optim.* 16, 4, Article 43 (Nov. 2019), 25 pages. https://doi.org/10.1145/3364999

[6] Emile Fiesler, Amar Choudry, and H. John Caulfield. 1990. Weight discretization paradigm for optical neural networks. In *Optical Interconnections and Networks*, Hartmut Bartelt (Ed.), Vol. 1281. International Society for Optics and Photonics, SPIE, 164 – 173. https://doi.org/10.1117/12.20700

[7] Christiam F. Frasser, Pablo Linares-Serrano, Iván Díez de los Ríos, Alejandro Morán, Erik S. Skibinsky-Gitlin, Joan Font-Rosselló, Vincent Canals, Miquel Roca, Teresa Serrano-Gotarredona, and Josep L. Rosselló. 2023. Fully Parallel Stochastic Computing Hardware Implementation of Convolutional Neural Networks for Edge Computing Applications. *IEEE Transactions on Neural Networks and Learning Systems* 34, 12 (2023), 10408–10418. https://doi.org/10.1109/TNNLS.2022.3166799

[8] Christiam F Frasser, Alejandro Morán, Vincent Canals, Joan Font, Eugeni Isern, Miquel Roca, and Josep L Rosselló. 2023. Approximate arithmetic aware training for stochastic computing neural networks. In *2023 38th Conference on Design of Circuits and Integrated Systems (DCIS)*. IEEE, 1–6.

[9] Brian R Gaines. 1969. Stochastic computing systems. *Advances in Information Systems Science: Volume 2* (1969), 37–172.

[10] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR* abs/1704.04861 (2017). arXiv:1704.04861 http://arxiv.org/abs/1704.04861

[11] Mike Innes. 2018. Flux: Elegant Machine Learning with Julia. *Journal of Open Source Software* (2018). https://doi.org/10.21105/joss.00602

[12] Michael Innes, Elliot Saba, Keno Fischer, Dhairya Gandhi, Marco Concetto Rudilosso, Neethu Mariya Joy, Tejan Karmali, Avik Pal, and Viral Shah. 2018. Fashionable Modelling with Flux. *CoRR* abs/1811.01457 (2018). arXiv:1811.01457 https://arxiv.org/abs/1811.01457

[13] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. 2017. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. *CoRR* abs/1712.05877 (2017). arXiv:1712.05877 http://arxiv.org/abs/1712.05877

[14] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[15] Tianmu Li, Wojciech Romaszkan, Sudhakar Pamarti, and Puneet Gupta. 2021. GEO: Generation and execution Optimized stochastic computing accelerator for neural networks. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 689–694.

[16] TensorFlow Model Optimization team. [n. d.]. Quantization Aware Training with TensorFlow Model Optimization Toolkit - Performance with Accuracy. https://blog.tensorflow.org/2020/04/quantization-aware-training-with-tensorflow-model-optimization-toolkit.html.

[17] Joonsang Yu, Kyounghoon Kim, Jongeun Lee, and Kiyoung Choi. 2017. Accurate and efficient stochastic computing hardware for convolutional neural networks. In *2017 IEEE International Conference on Computer Design (ICCD)*. IEEE, 105–112.