Sensitivity-Aware Finetuning for Accuracy Recovery on Deep Learning Hardware

Lakshmi Nair
Lightmatter
100 Summer Street
Boston, MA 02110
`lakshmi@lightmatter.co`

Darius Bunandar
Lightmatter
100 Summer Street
Boston, MA 02110
`darius@lightmatter.co`

Abstract

Existing methods to recover model accuracy on analog-digital hardware in the presence of quantization and analog noise include noise-injection training. However, it can be slow in practice, incurring high computational costs, even when starting from pretrained models. We introduce the Sensitivity-Aware Finetuning (SAFT) approach that identifies noise sensitive layers in a model, and uses the information to freeze specific layers for noise-injection training. Our results show that SAFT achieves comparable accuracy to noise-injection training and is $2\times$ to $8\times$ faster.

1 Introduction

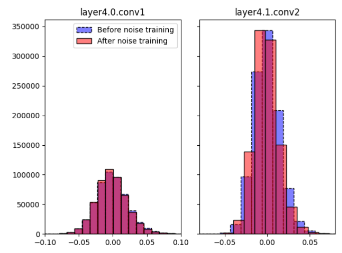
Recent advances in analog-digital hardware is motivated by improving energy and speed efficiency for deep learning applications. However, such devices are often susceptible to effects of analog noise and reduced precision (quantization) which impacts the final model accuracy. One of the commonly used approaches for tackling this issue includes noise-injection training. Here, the model is subjected to the perturbations caused by quantization and/or analog noise, by injecting some representative noise into the model's layers during training, to recover accuracy [1, 2, 3, 4]. Prior work has shown that loss of precision due to quantization can also be treated as "noise" and that models can be made resilient to this loss by retraining with noise injection, where the injected noise is proportional to the precision loss [4, 3]. However, noise-injection training can incur significant training time, even when starting from pretrained models. The speed of training can be potentially improved by training only a subset of the layers that are highly sensitive to noise, while freezing (i.e., disable weight updates) the rest. As shown in Figure 1a for ResNet50, some weights change substantially during noise-injection training, while others hardly change and could potentially be frozen (conceptually similar to transfer learning [5]). While prior work has looked at similar methods for speeding up training, they either focus on BERT-like models [6, 7], or rely on domain knowledge to identify noise sensitive layers [8].

Motivated by these observations, we seek to answer the question: "*Starting from a pretrained model, can we identify which layers are the most sensitive to noise, and retrain just those?*". Prior work in quantization has used KL-divergence to identify layers that are sensitive to quantization [9, 10, 11]. We present an alternate metric for measuring layer sensitivity to noise, by computing the *standard deviation* of the output differences between the noisy/quantized model and the noise-free/unquantized model at each layer. We then introduce the Sensitivity-Aware Finetuning (SAFT) approach based on noise-injection training, that selects specific layers for training based on the layer sensitivity analysis.

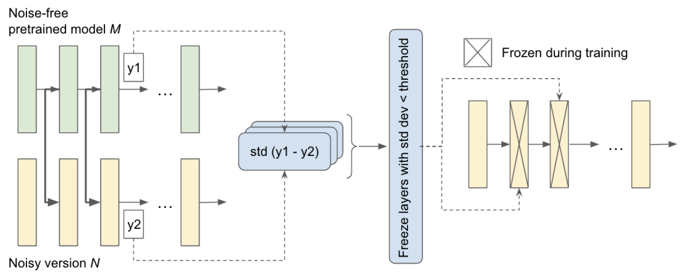
2 Sensitivity-Aware Finetuning

The motivation behind SAFT is the observation that after noise-injection training, the parameters of a model change significantly only for specific layers, e.g., Figure 1a shows this for ResNet50. Keeping

Preprint. Under review.



(a) Weights of ResNet50, before and after noise-injection training from a pretrained model. For some layers, the weights do not change much post-training.


Noise-free pretrained model M
y1
y2
Noisy version N
std (y1 - y2)
Freeze layers with std dev < threshold
Frozen during training

(b) SAFT begins by passing the inputs of each layer in the noise-free model through the corresponding layers of the noisy model. The outputs for both models at each layer is used to compute a standard deviation over their differences. Only layers with standard deviations above a certain threshold are trained using noise-injection training while the remaining are frozen.

Figure 1: Sensitivity-Aware Finetuning (SAFT): motivation (left) and description (right)

such layers noise-free in the model results in larger accuracy improvements. This possibly indicates that the layers whose parameters change the most have higher noise sensitivity. Hence, we seek to retrain only the most noise sensitive layers to validate this hypothesis.

Our approach takes a pretrained model M, its noisy version N, and a sample batch of inputs X from the training data. Note that N refers to the model used during standard noise-injection training [1], where we inject noise into the weights during the forward pass to perturb the outputs. The input data is first passed through M and the inputs and outputs at every layer are stored. Then, the inputs at every layer of the *original model* M are passed through the *corresponding layers of* N (See Appendix Algorithm 1). The layer outputs for N are also saved, and the standard deviation[1] of the differences between the outputs of M and N are computed per-layer. The process flow is shown in 1b.

Once the standard deviations are computed, SAFT involves: a) Identifying the top k layers with the highest standard deviations; b) Selectively training only the top k layers of N while freezing the parameters of the remaining layers. We note that k is an additional hyperparameter. The value of k can be determined based on visualizing the standard deviation values in a plot, or it can be treated as the other hyperparameters and set using tools such as Tune [13]. Another consideration here is the batch size used for computing the statistics. The batch size should be sufficiently large to obtain a reasonable estimate of the noise sensitivity[2]. When large batches cannot be processed, data samples can be processed individually and stacked. The statistics can then be accumulated over the stack. For training, we use the procedure in [1], and apply the backward gradient updates to noise-free weights.

3 Experiments

We evaluate SAFT on eight different models. In all cases, similar to prior work in [14] we only apply noise to the matrix-multiplication layers (such as Convolutions, Linear etc.), and leave other layers such as batchnorm or activation layers as noise-free. We also evaluate the use of KL-divergence as an alternate metric to standard deviation in our experiments. Similar to prior work, we evaluate SAFT with simulated hardware noise using both multiplicative and additive noise, wherein noise is injected into the weights [1, 2, 15]. We sample the noise N from both a Gaussian distribution with zero mean as in prior work [2], $N \sim \mathcal{N}(0, \sigma)$, and from a Uniform distribution $N \sim U[-r_1, r_1]$. Our baseline noise-injection is implemented similar to the approach in [1]. The parameters of the noise distributions for the different models are shown in Appendix Table 5. The specific noise parameters were chosen so as to result in a drop in the performance of all the models, which can then be recovered through training. Note that we set a fixed seed for all our training runs to ensure fair comparison.

For SAFT, we compute the standard deviation values on a single batch of training data. We freeze $total - k$ layers in a model during training, retraining only k. We determined k empirically by visualizing the standard deviation plots and checking the number of layers that have a relatively high

[1]Noise mean is typically zero based on hardware models [12, 1, 3]

[2]We find that using the batch sizes typically used for training, works well in most cases

Table 1: Results comparing SAFT with baseline noise-injection training with Gaussian noise shows similar performances. Here "Untrained" denotes performance before training, when noise is injected. Note that SAFT achieves accuracy close to noise-injection training while being 2× to 8× faster.

		Multiplicative Gaussian			**Additive Gaussian**			**SAFT**
	FP32	**Untrained**	**Noise-inj**	**SAFT**	**Untrained**	**Noise-inj**	**SAFT**	**Speed ↑**
ResNet18	69.8	68.7	69.1	69.0	66.0	67.4	67.8	2×
ResNet34	73.3	72.1	72.9	72.9	69.0	70.0	70.0	4×
ResNet50	76.1	74.9	75.8	75.6	70.7	73.1	73.1	8×
ResNeXt50	77.6	72.2	74.4	74.0	71.1	73.9	74.2	8×
MobileNet v3	74.0	70.8	71.7	71.6	70.9	72.7	72.6	5×
Faster RCNN	59.0	56.5	58.9	58.7	52.2	54.4	54.8	3×
Mask RCNN	56.0	52.0	55.3	55.6	48.5	53.6	54.9	3×
Bert Base	74.7	73.1	74.4	74.6	72.4	74.4	74.2	2×

Table 2: Results for finetuning with Gaussian injected noise using KL-divergence to freeze layers.

	Untrained (Multiplicative / Additive)	**Multiplicative (KL-d)**	**Additive (KL-d)**
ResNet34	72.1 / 69.0	71.9	68.2
ResNet50	74.9 / 70.7	74.2	69.4
Bert base	73.1 / 72.4	74.2	74.1

noise standard deviation. We seek to standardize this procedure in our future work. Table 4 in the Appendix shows the batch size, and k values (#Frozen $=$ #Total $- k$) used in our experiments. Some models require more layers to be trained than others owing to higher noise in more layers.

Our experiments evaluate: *Given the exact same training parameters, does SAFT perform similar to baseline noise-injection training?* Note that our research question compares our approach to noise-injection training, rather than to obtain a predefined target performance, which noise-injection training has already been shown to achieve with sufficient epochs [1, 4, 14]. In our training experiments, we only train for a few epochs (1-5 epochs) to see if the performances of the two approaches match, whereas achieving close to the baseline noise-free FP32 performance takes many more epochs [14].

4 Results

Noise standard deviation plots for four models are shown in Figure 2. Stars $\star$ indicate the layers that are trained, while the remaining are frozen. Similar to prior findings, the first and last set of layers in vision models exhibit high sensitivity [16, 17, 1]. We also see a "sawtooth" pattern in the vision models like ResNet, corresponding to the repeating blocks in the network, consistent with observations in prior work [12]. For MobileNet v3, quite a few of the convolution layers have a higher noise standard deviation compared to ResNet50. For Faster RCNN, we see that several layers in the "head" of the model, responsible for predicting the bounding box locations, are particularly sensitive. For Bert base, the sensitivity is quite spread out across the model with several layers exhibiting high noise sensitivity. Specifically, 10 of the trained 20 layers are self-attention layers, with the remaining

Table 3: Results comparing SAFT with baseline noise-injection training for Uniform noise shows similar performances. Here "Untrained" denotes performance before training, when noise is injected.

		Multiplicative Uniform			**Additive Uniform**			**SAFT**
	FP32	**Untrained**	**Noise-inj**	**SAFT**	**Untrained**	**Noise-inj**	**SAFT**	**Speed ↑**
ResNet18	69.8	68.2	69.3	69.0	64.8	66.9	66.2	2×
ResNet34	73.3	71.8	72.7	72.6	67.1	68.5	68.0	4×
ResNet50	76.1	74.5	75.1	75.3	67.9	70.5	70.1	8×
ResNeXt50	77.6	71.3	73.0	73.1	73.0	75.7	75.3	8×
MobileNet v3	74.0	71.4	72.1	72.2	72.9	73.9	73.8	5×
Faster RCNN	59.0	57.1	58.2	58.2	56.2	57.0	57.2	3×
Mask RCNN	56.0	53.4	54.5	54.8	49.5	51.5	51.6	3×
Bert Base	74.7	68.9	72.0	72.4	62.2	72.1	72.7	2×

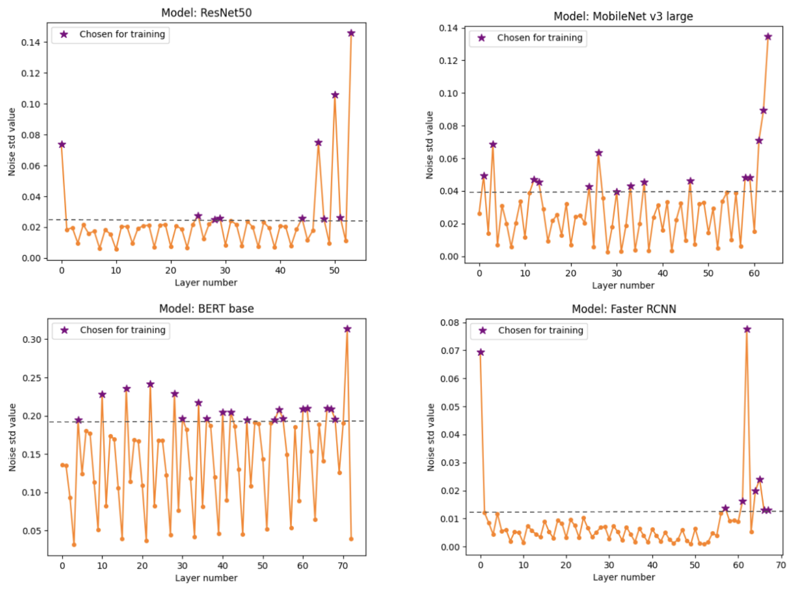


Figure 2: Plots of the per-layer standard deviations for four models: ResNet50, MobileNet v3 large, Bert base, and Faster RCNN. The purple stars $\star$ denote the layers that were selected for training while the rest were frozen. First and last layers of vision models have high standard deviations.

10 being intermediate and output dense layers. For models like Bert, identifying the most sensitive layers can be tricky and a larger proportion of layers have to be trained compared to other models.

The corresponding training speed improvements are shown in Tables 1 and 3, where up to $8\times$ speed improvements in training can be observed. In the case of a few models such as ResNet18 and RCNN, speed up of about $2\times$ and $3\times$ is observed. The actual amount of speedup depends on the processing time of each layer, which in turn depends on the size of the layer (i.e., # of parameters). Hence, a direct correlation between size of the model (i.e., # of layers) and the speedup is difficult to establish.

The final performance of SAFT in terms of the model metrics is shown in Tables 1 and 3 for Gaussian and Uniform noise respectively. We see that SAFT (with k frozen layers) closely matches[3] the performance of the full noise-injection training approach for all noise models, leading to improvements in terms of the metrics. An interesting finding here is that specific layers that do not form a continuous sequence, can be independently trained. Typically in transfer learning a *continuous* sequence of the last few layers, such as the last few convolutional and fully-connected layers, are often retrained [5].

Lastly, we finetune a few models with Gaussian noise injection, using KL-divergence for selecting the layers to freeze as opposed to using standard deviation (See Table 2). Interestingly, using KL-divergence did not improve performance on vision models, although it did perform well on Bert base. It is possible that since most layers in Bert base have high noise sensitivity (See Figure 2), KL-divergence chose and trained some of the noisiest layers, whereas the noisiest layers are much more specific and localized in the case of the vision models. These differences need further investigation.

5 Conclusions and Future Work

We introduced Sensitivity-Aware Finetuning (SAFT) for fast finetuning of pretrained models to deal with noise. SAFT computes layer sensitivity using standard deviations to freeze some layers. SAFT performs comparably to noise-injection training in terms of accuracy, while being faster at training. In the future, we will investigate additional metrics for SAFT, including combinations of metrics like standard deviation and KL-divergence. We will investigate techniques for easily identifying the k hyperparameter used in SAFT and investigate the reasons as to why standard deviation works better than KL-divergence in some cases and vice-versa. We believe the layer sensitivity analysis can also be used for performing Partial Quantization and Quantization-Aware Training [14] in future work.

[3]Our results were confirmed with the Wilcoxon Signed Rank test ($\alpha = 0.05$)

References

[1] V. Joshi, M. Le Gallo, S. Haefeli, I. Boybat, S. R. Nandakumar, C. Piveteau, M. Dazzi, B. Rajendran, A. Sebastian, and E. Eleftheriou, "Accurate deep neural network inference using computational phase-change memory," *Nature communications*, vol. 11, no. 1, pp. 1–13, 2020.

[2] C. Zhou, P. Kadambi, M. Mattina, and P. N. Whatmough, "Noisy machines: Understanding noisy neural networks and enhancing robustness to analog hardware errors using distillation," *arXiv preprint arXiv:2001.04974*, 2020.

[3] A. Basumallik, D. Bunandar, N. Dronen, N. Harris, L. Levkova, C. McCarter, L. Nair, D. Walter, and D. Widemann, "Adaptive block floating-point for analog deep learning hardware," *arXiv preprint arXiv:2205.06287*, 2022.

[4] C. Baskin, N. Liss, E. Schwartz, E. Zheltonozhskii, R. Giryes, A. M. Bronstein, and A. Mendelson, "Uniq: Uniform noise injection for non-uniform quantization of neural networks," *ACM Transactions on Computer Systems (TOCS)*, vol. 37, no. 1-4, pp. 1–15, 2021.

[5] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.

[6] D. Vucetic, M. Tayaranian, M. Ziaeefard, J. J. Clark, B. H. Meyer, and W. J. Gross, "Efficient fine-tuning of bert models on the edge," *arXiv preprint arXiv:2205.01541*, 2022.

[7] E. B. Zaken, S. Ravfogel, and Y. Goldberg, "Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models," *arXiv preprint arXiv:2106.10199*, 2021.

[8] T. Piao, I. Cho, and U. Kang, "Sensimix: Sensitivity-aware 8-bit index & 1-bit value mixed precision quantization for bert compression," *PloS one*, vol. 17, no. 4, p. e0265621, 2022.

[9] S. Migacz, "8-bit inference with tensorrt," https://on-demand.gputechconf.com/gtc/2017/presentation/s7310-8-bit-inference-with-tensorrt.pdf, May 2017.

[10] L. Kummer, K. Sidak, T. Reichmann, and W. Gansterer, "Adaptive precision training (adapt): A dynamic fixed point quantized training approach for dnns," *arXiv preprint arXiv:2107.13490*, 2021.

[11] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," *arXiv preprint arXiv:2103.13630*, 2021.

[12] S. Garg, J. Lou, A. Jain, and M. Nahmias, "Dynamic precision analog computing for neural networks," *arXiv preprint arXiv:2102.06365*, 2021.

[13] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," *arXiv preprint arXiv:1807.05118*, 2018.

[14] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius, "Integer quantization for deep learning inference: Principles and empirical evaluation," *arXiv preprint arXiv:2004.09602*, 2020.

[15] L.-H. Tsai, S.-C. Chang, Y.-T. Chen, J.-Y. Pan, W. Wei, and D.-C. Juan, "Robust processing-in-memory neural networks via noise-aware normalization," *arXiv preprint arXiv:2007.03230*, 2020.

[16] J. L. McKinstry, S. K. Esser, R. Appuswamy, D. Bablani, J. V. Arthur, I. B. Yildiz, and D. S. Modha, "Discovering low-precision networks close to full-precision networks for efficient embedded inference," *arXiv preprint arXiv:1809.04191*, 2018.

[17] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, 2016, pp. 525–542.

A Appendix

Table 4: # of total layers vs. frozen layers, and the speed increase for SAFT over baseline noise-injection training (*Lower k values could potentially be obtained through a more rigorous search).

	Batch size	**k**	**# Total**	**# Frozen**
ResNet18	256	10	21	11
ResNet34	256	10	37	27
ResNet50	256	10	54	44
ResNeXt50	256	10	54	44
MobileNet v3	256	15	64	49
Faster RCNN	20	8	68	60
Mask RCNN	20	8	74	66
Bert base	80	20	73	53

Table 5: Noise distribution parameters for the different models. The specific noise parameters were chosen so as to result in a drop in the performance of all the models, which can then be recovered through training.

	Gaussian Noise σ		**Uniform noise r_1**		
	Mult	**Add**	**Mult**	**Add**	**Epochs (training)**
ResNet18	0.05	0.005	0.1	0.01	3
ResNet34	0.05	0.005	0.3	0.01	3
ResNet50	0.05	0.005	0.3	0.01	3
ResNeXt50	0.08	0.005	0.15	0.008	3
MobileNet v3 large	0.02	0.005	0.03	0.008	5
Mask RCNN	0.01	0.005	0.01	0.008	1
Faster RCNN	0.05	0.005	0.05	0.005	1
Bert base	0.02	0.002	0.3	0.01	1

Algorithm 1: Computing layer sensitivity

```
Input: Original model $M$, noisy model $N$
Output: Layer output standard deviations $S$
Data: Sample batch of inputs $X$
Function save_data($M$, $X$):
    // Save input-output of each layer
    $I_{model}$ = { }
    $O_{model}$ = { }
    for $i$, $l$ in enumerate(M.layers) do
        if $i$ = 0 then
            $Y = l(X)$
            $I_{model}[l] = X$
            $O_{model}[l] = Y$
        else
            $I_{model}[l] = Y$
            $Y = l(Y)$
            $O_{model}[l] = Y$
        end
    end
    return $I_{model}, O_{model}$

Function compute_stats($M$, $N$, $X$):
    $I_{clean}, O_{clean}$ = save_data($M$, $X$)
    $S$ = { }
    for $l$ in N.layers do
        // Pass inputs of M through N
        $Y_{noisy} = l(I_{clean}[l])$
        $\hat{Y} = Y_{noisy} - O_{clean}[l]$
        $S[l] = std(\hat{Y})$
    end
    // Sort in decreasing std values
    return $sort(S, \downarrow)$
```