NAP: Noise-Based Sensitivity Analysis for Programs

Jesse Michel[*] MIT
Sahil Verma[*] IIT Kanpur
Benjamin Sherman MIT
Michael Carbin MIT

1 Introduction

Low-precision approximation of programs allows for faster computation especially in fields such as machine learning, data analytics, and vision. Such approximations automatically transform a program to produce one that approximates the original output but executes much faster. At the heart of this approximation is sensitivity analysis – understanding the program's robustness to various perturbations. Sensitivity analysis provides a metric for importance that may be used to select how much each parameter may be modified in the search for fast, accurate approximate programs.

We propose *NAP* (noise-based analyzer of programs) which provides a novel sensitivity analysis to model perturbations to each operator and variable in an expression. NAP performs sensitivity analysis by introducing independent Gaussian noise to each of the parameters to be approximated and then optimizing the parametrized variances over the stochastically sampled model (see section 2 for an example). If a parameter p is less sensitive to perturbations then p admits more noise, i.e., the trained variance of the Gaussian for p will be large. Likewise, a parameter that is more sensitive to perturbations will admit less noise.

NAP uses stochastic gradient descent (SGD) to optimize variances to maximize the loss to meet a given quality-of-service. We validate NAP's sensitivities by using them to generate mixed-precision approximate programs for a set of scientific computing benchmarks and for the weights of a neural network. We compare our results with those produced by FPTuner, a state-of-the-art tool for mixed-precision analysis, and with Hessian-based and uniform neural network quantization [1, 2].

2 NAP (by Example)

We will first give an overview of our approach and then show how this applies to a simple example. NAP takes in as input a program f, a distribution $\mathcal{D}$ over inputs to f, and a loss function $\mathcal{L}$ that describes how good approximate outputs are, and produces a sensitivity analysis. The following is the pipeline for sensitivity analysis (using NAP) and approximation:

1. Add 0-mean Gaussian noise parametrized by variance to every variable and operator.
2. Solve the optimization problem to obtain variances.
3. Use the variances to find an (mixed-precision) allocation for variables and operators.

[*]These authors contributed equally to the paper.

For example, consider the quadratic function $x \times x + 2x + 1$ or, parametrizing the operators,

$$f(x, \{\times_0, \times_1, +_2, +_3\}) = x \times_0 x +_2 2 \times_1 x +_3 1.$$

Suppose a distribution on inputs $x \sim \mathcal{D}$. NAP provides sensitivity analysis by explicitly inserting variables $\epsilon_k \sim \mathcal{N}(0, \sigma_k^2)$ for each operator $\odot_k$ and $\epsilon_y \sim \mathcal{N}(0, \sigma_y^2)$ for each variable y. Inserting these variables gives

$$g(x; \vec{\epsilon}) = (x + \epsilon_x)(x + \epsilon_x) + \epsilon_0 + 2(x + \epsilon_x) + \epsilon_1 + \epsilon_2 + 1 + \epsilon_3.$$

Letting $\vec{\sigma}$ be the vector of standard deviations, NAP uses SGD to solve the optimization problem

$$\max_{\vec{\sigma}} \Big(\sum_i \log(\sigma_i) - \lambda \mathop{\mathbb{E}}_{x \sim \mathcal{D}, \vec{\epsilon} \sim \mathcal{N}(\vec{0}, \vec{\sigma}^2)} \mathcal{L}(x, g(x; \vec{\epsilon})) \Big) \qquad (1)$$

where $\lambda > 0$ is a regularization parameter and in this example the loss $\mathcal{L}$ is the squared error

$$\mathcal{L}(x, \hat{y}) = \left(g(x; \vec{0}) - \hat{y} \right)^2.$$

The first term in the objective specifies that larger variances are more desirable solutions, while the second term ensures that the approximate result $g(x; \vec{\epsilon})$ is close to the exact result $g(x; \vec{0})$.

NAP uses the optimal variances to determine the approximate program by assigning each operator $\odot_k$ and variable y an allocation of mantissa bits. Imagine $x \sim \mathcal{U}[-1-\delta, -1+\delta]$ in our running example. If δ is large, NAP will make σ_x small since there will be a large noise contribution of roughly $\sigma_x x$ to the final answer. For small δ, $\sigma_x x$ is small allowing for σ_x to increase. Verifying this experimentally, when $\delta = 1$, NAP assigns a quarter of the noise to σ_x as for $\delta = 0.1$. Thus, different data distributions result in different sensitivities and thus different variable and operator precisions.

3 Case Study: Accuracy

We used NAP to generate mixed-precision approximate versions of numerical programs for scientific computing from the FPBench benchmark set [1]. Program inputs for FPBench programs have a specified range. We generated distributions over inputs by assuming each input is uniformly distributed over its range, and that inputs are independent. We then ran NAP to create mixed-precision approximate programs and evaluated their root-mean-square error (RMSE).

FPTuner [1] provides upper bounds on the error for double-precision versions of these FPBench programs for *any* possible inputs within range, so we set λ to yield RMSEs just lower than that bound. Table 1 shows these RMSEs in comparison to the FPTuner error bounds, together with the mean number of mantissa bits used in our approximate programs to achieve those RMSEs. This gives some measure of how weakening from worst-case bounds to average-case bounds allows further approximation.

Benchmarks	FPTuner	RMSE	Mean Bits
verlhulst	3.79e-16	3.72e-16	50
sineOrder3	1.17e-15	7.90e-16	50
predPrey	1.99e-16	1.73e-16	50
sine	8.73e-16	8.34e-17	51
doppler1	1.82e-13	7.75e-14	51
doppler2	3.20e-13	1.07e-13	51
doppler3	1.02e-13	5.10e-14	51
rigidbody1	3.86e-13	1.37e-13	51
sqroot	7.45e-16	4.00e-16	50
rigidbody2	5.23e-11	6.08e-12	51
turbine2	4.13e-14	2.35e-14	50
carbon gas	1.51e-08	3.01e-09	49
turbine1	3.16e-14	1.13e-14	51
turbine3	1.73e-14	1.40e-14	50
jet	2.68e-11	1.07e-11	50

Table 1. We compare FPTuner's maximum error bound against NAP's empirical root mean squared error. Mean bits is the average number of bits in the mantissa of the approximate program (vs. FPTuner's 52-bit mantissa).

This relaxed requirement gives NAP the flexibility to generate tighter expected error bounds using fewer bits than FPTuner for all of the benchmarks.

4 Case Study: Scalability

An additional challenge for FPTuner (and other solver-based techniques) is that they do not scale to larger programs. To evaluate the FPTuner's scalability or larger numerical programs, we evaluated FPTuner on two programs. The first computes the sum of elements of the matrix obtained by multiplication of two 5 by 5 input matrices. The second computes the result of applying a sigmoid function to the dot product of a input vector with a weight vector each of size 50. These programs model the computation of a neuron in a neural network. We applied FPTuner to both programs with an error bound of 1e-15 and FPTuner time-out given a threshold of 15 hours.

To validate NAP's scalability, we used it to perform mixed-precision quantization and pruning of deep neural networks. In case the of neural network quantization we only add (and train) noises on model weights and not the operators. We update noise values for 50 epochs for the MNIST dataset.

Figure 1 presents our results on a LeNet-style architecture for MNIST [7]. Our LeNet-style architecture involves arithmetic over 27K parameters [6].

After training the noises, for each weight w we have a tuned standard deviation σ and we define

$$f(\sigma) = \lfloor c - \log(\sigma) \rfloor$$

for a constant c that is varied in order to tune the average bitwidth as shown in Figure 1. If $f(\sigma) \leq 0$ we prune w and otherwise we quantize w to $f(\sigma)$ bits. We set the loss $\mathcal{L}$ used in equation 1 to be the categorical cross-entropy, which is consistent with the loss used to train the original network.

One can see in Figure 1 that even for extremely low bitwidths, much of the accuracy of the model on MNIST remains. For example, with an average of about 2 bits we achieve 95.3% test accuracy. With 8-bits, all of the approaches lose no more than 1% of the test accuracy.

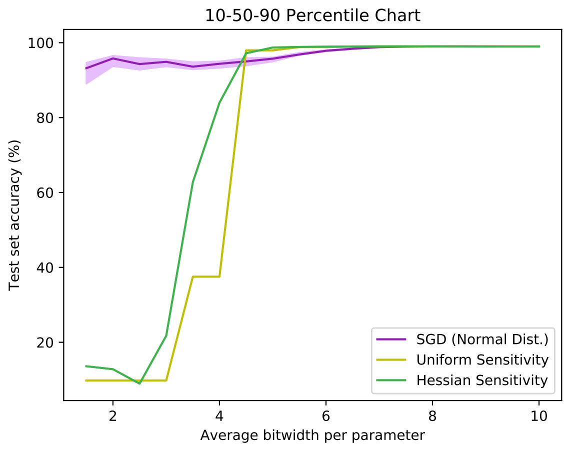

Figure 1. We show the space-accuracy tradeoff after quantizing weights. At low bitwidths, NAP outperforms uniform and Hessian-based quantization. Because results generated from NAP (purple) are randomized, we ran it 30 times to show the distribution of results.

The Hessian sensitivity approach in Figure 1 computes the sensitivity of each parameter p as $\log(\frac{\partial^2 \mathcal{L}}{\partial p^2})$, which is a standard analytical technique [2].

Approximating tens of thousands of weights demonstrates how NAP's sensitivity analysis scales well to large numerical programs in a way not possible by other approaches.

5 Related Work

Researchers have approached program approximation by performing loop perforations, function substitutions, and quantization [3, 4, 8]. Other numerical tuning approaches provide maximum instead of expected error bounds [1, 3]. This approach scales poorly and is less well suited to modern applications such as machine learning.

Other researchers introduced their own sensitivity analysis techniques to produce annotations that identify operations requiring high precision [9, 10].

We compare our neural network to a Hessian-based approach and show that at low bitwidths, it produces worse results than our approach [2]. Our sensitivity analysis is similar to the noise model to compute generalization bounds on neural networks [5].

References

[1] Wei-Fan Chiang, Mark Baranowski, Ian Briggs, Alexey Solovyev, Ganesh Gopalakrishnan, and Zvonimir Rakamarić. 2017. Rigorous floating-point mixed-precision tuning. *ACM SIGPLAN Notices* 52, 1 (2017), 300–315. https://doi.org/10.1145/3093333.3009846

[2] Yann Le Cun, John S. Denker, and Sara A. Solla. 1990. Optimal Brain Damage. In *Advances in Neural Information Processing Systems*. Morgan Kaufmann, 598–605.

[3] Eva Darulova and Viktor Kuncak. 2014. Sound compilation of reals. *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL 14* (2014). https://doi.org/10.1145/2535838.2535874

[4] S. Han, H. Mao, and W. J. Dally. 2015. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *ArXiv e-prints* (Oct. 2015). arXiv:cs.CV/1510.00149

[5] G. Karolina Dziugaite and D. M. Roy. 2017. Computing Nonvacuous Generalization Bounds for Deep (Stochastic) Neural Networks with Many More Parameters than Training Data. *ArXiv e-prints* (March 2017). arXiv:1703.11008

[6] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*. 2278–2324.

[7] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/. (2010). http://yann.lecun.com/exdb/mnist/

[8] Sasa Misailovic, Michael Carbin, Sara Achour, Zichao Qi, and Martin C. Rinard. 2014. Chisel: Reliability- and Accuracy-Aware Optimization of Approximate Computational Kernels. *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages and Applications - OOPSLA 14* (2014). https://doi.org/10.1145/2660193.2660231

[9] B. Nongpoh, R. Ray, S. Dutta, and A. Banerjee. 2017. AutoSense: A Framework for Automated Sensitivity Analysis of Program Data. *IEEE Transactions on Software Engineering* 43, 12 (Dec 2017), 1110–1124. https://doi.org/10.1109/TSE.2017.2654251

[10] Pooja Roy, Rajarshi Ray, Chundong Wang, and Weng Fai Wong. 2014. ASAC: Automatic Sensitivity Analysis for Approximate Computing. *Proceedings of the 2014 SIGPLAN/SIGBED conference on Languages, compilers and tools for embedded systems - LCTES 14* (2014). https://doi.org/10.1145/2597809.2597812