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 enables faster computation in fields such as machine learning, data analytics, and vision. Such approximations automatically transform a program into 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 to measure how much each value in the program to produce a faster, yet accurate, approximate program.

We propose *NAP* (Noise-based Analyzer of Programs) which provides a novel sensitivity analysis of each operator and variable in a program. NAP performs sensitivity analysis by introducing independent Gaussian noise to each value in a program (e.g., arithmetic operator and variable reference), producing a stochastic semantics of the program.

NAP then jointly maximizes the variances of the noise distributions subject to a bound on the stochastic program's expected error. NAP poses the maximization process as the solution to a novel constrained optimization problem and solves the problem using stochastic gradient descent (SGD).

Each program value's resulting variance denotes its sensitivity. If a value is less sensitive to perturbations, then the optimal variance of the value's Gaussian will be large. Likewise, if a program value is more sensitive, then its variance will be small. Together, these variances describe a distribution over potentially valid approximations, which we term the program's *noise envelope.*

NAP's design explores a new area for sensitivity analysis in that its noise-based approach computes a *region-based* estimate of sensitivity that computes the expected error with respect to perturbations within the program's entire noise envelope. This approach can more accurately characterize sensitivity than a *point-based* estimate, such as the derivative, because while the derivative at a point may be large in magnitude, the total variation in expected error over the point's local region may be small.

In this paper, we validate NAP's sensitivities by using them to generate mixed-precision approximate programs for a neural network as well as for a set of scientific computing benchmarks. We demonstrate the value of NAP's noise-based approach as well as validate the relationship between sensitivity analysis for expected error versus that for worst-case error.

[*]These authors contributed equally to the paper.

2 NAP (by Example)

NAP takes 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 it produces a sensitivity analysis. In this example, NAP's pipeline for sensitivity analysis is:

1. Add 0-mean Gaussian noise parametrized by variance to every variable and operator.
2. Solve an optimization problem to maximize these variances subject to an expected error constraint.

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

$$f(x, \{\times_0, +_1, \times_2, +_3\}) = x \times_0 x +_1 2 \times_2 x +_3 1.$$

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 a new function g defined as

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

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

$$\max_{\vec{\sigma}} \Big(\underbrace{\sum_i \log(\sigma_i)}_{\text{noise envelope}} - \lambda \underbrace{\mathop{\mathbb{E}}_{x \sim \mathcal{D}, \vec{\epsilon} \sim \mathcal{N}(\vec{0}, \vec{\sigma}^2)} \mathcal{L}(x, \vec{\epsilon})}_{\text{minimize error}} \Big) \tag{1}$$

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

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

The first term in the objective specifies that solutions with larger volume noise envelops are more desirable. The second term ensures that the approximate result $g(x; \vec{\epsilon})$ is close to the exact result $g(x; \vec{0})$ by estimating the expected error, computed by averaging over samples from the data distribution.

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 variance of roughly $\sigma_x x$ added to the output. For small δ, $\sigma_x x$ is small allowing for σ_x to increase. Verifying this experimentally, NAP assigns a standard deviation σ_x of four times the magnitude when $\delta = 0.1$ versus $\delta = 1$.

3 Case Studies

NAP's sensitivity analysis is a new point in the trade-off space of approaches in that it measures the sensitivity of a local region around the original values of the program than at the original values themselves. In the following case studies we validate the behavior of NAP's sensitivity analysis by using its sensitivities to perform mixed-precision selection for several programs, including neural networks and a benchmark suite of standard, numerical programs.

3.1 Mixed-Precision Allocation

To validate NAP, we used it to perform mixed-precision quantization and pruning of deep neural networks. For this case study we only add (and train) noises on the neural network's weights and not the operators in its computation.[1] We set the loss $\mathcal{L}$ used in Equation 1 to be the categorical cross-entropy loss, which is consistent with the loss used to train the original network.

After training the variance for each weight w, we compute the standard deviation σ and define a function f with

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

for a constant c. If $f(\sigma) \leq 0$ we prune w, and otherwise we quantize w to $f(\sigma)$ mantissa bits for use with MPFR's multiple-precision floating-point computation representation, which supports specifying an arbitrary number of bits of precision in the mantissa. For this usage, c denotes a parameter that we can vary to control the average bitwidth as shown in Figure 1.

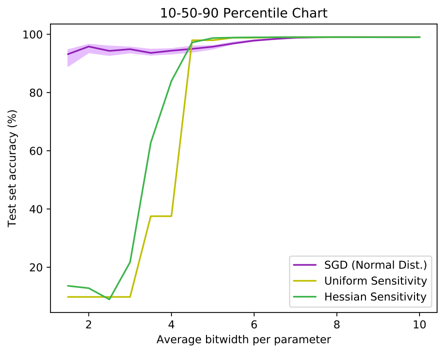

Figure 1. The accuracy of a quantized (fixed-point quantized) model as a function of average bitwidth. Because NAP results (purple) are stochastic, we ran it 30 times to show the distribution of results.

Figure 1 presents our results on a LeNet-style architecture for MNIST [7]. Our LeNet-style architecture involves arithmetic over 27K weights [6]. We train the noise variances for 50 epochs (50 full sweeps) of the MNIST dataset.

Figure 1 shows that even for low average bitwidths, much of the accuracy of the model remains. For example, with an average of about 2 bits we achieve 95.3% test accuracy versus a 99.5% baseline accuracy of the unquantized model. Moreover, NAP outperforms uniform quantization (quantizing the entire neural network to a fixed bitwidth) at these low average bitwidths. This demonstrates that NAP's sensitivity analysis is able to distinguish between weights that are more sensitive to perturbation and weights that are less sensitive (or not necessary at all) and therefore allocate them more or fewer (or no) bits, respectively.

3.2 Region-based Sensitivity

As a comparison to a point-based approach (versus our noise-based approach), the Hessian Sensitivity line in Figure 1 shows the result of calculating the second-order derivatives of each parameter, which is a standard analytical technique.

The line for Hessian sensitivity in Figure 1 computes the sensitivity of each parameter p as $\log(\frac{\partial^2 \mathcal{L}}{\partial p^2})$, which is a standard, point-based sensitivity technique [2]. Comparing figures 2 and 3 we see that the Hessian sensitivities follow a unimodal distribution, while NAP's sensitivities follow a bimodal distribution.

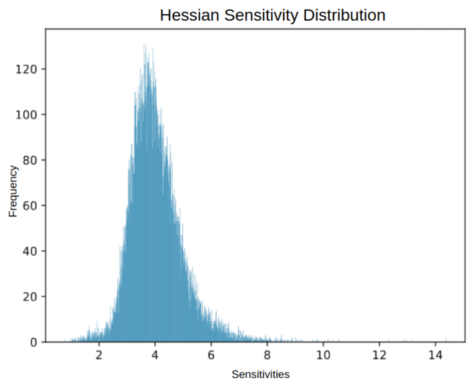

Figure 2. The Hessian-based sensitivities have a single mode.

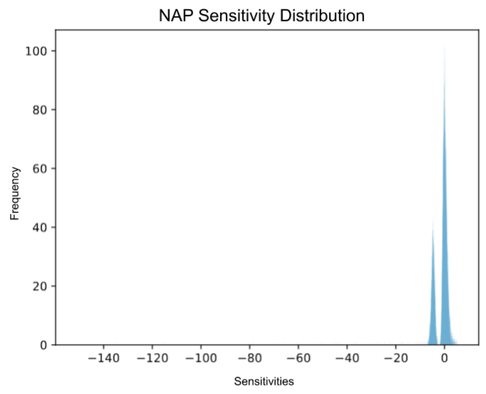

Figure 3. Bimodal noise distribution of NAP with a heavy left tail i.e. a small collection of weights that are very sensitive to perturbations.

[1]In Section 3.3 we also consider noise of variances

3.3 Expected versus Worse-case Error

NAP optimizes the expected error of the program, where the expectation quantifies over both the input distribution as well as the noise distribution. To evaluate the relationship between this expected error and a more standard worst-case error model, we compare mixed-precision allocations guided by NAP versus those from FPTuner [1] on a suite of numerical programs for scientific computing from FPBench.

FPTuner. FPTuner [1] provides upper bounds on the error for double-precision versions of these FPBench programs for *any* possible inputs within the range.

Although we compare NAP to FPTuner, they have fundamental differences [1]. For input variables from a given range, FPTuner satisfies a desired absolute error bound. To achieve this, FPTuner creates an appropriate mixed-precision allocation, for example, it may set operators and variables to either 64-bit or 128-bits as needed. The second column of Table (1) shows the bound for an allocation where all variables and operators are 64-bits.

An additional challenge for FPTuner (and other solver-based techniques) is that they do not scale to larger programs. To evaluate FPTuner's scalability on 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 an 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 10^{-15} and FPTuner timed out given a threshold of 15 hours.

Methodology. We used NAP to generate a mixed-precision approximate program that satisfies a given expected error constraint where we measure error as root-mean-square error (RMSE). We generate an input distribution by sampling. We assume each input is uniformly distributed over the range that FPBench specifies. For programs with multiple inputs, we generate inputs independently. We then ran NAP to generate sensitivities. To achieve a given expected error, we set the λ parameter for NAP's optimization problem using binary search.

To generate mixed-precision programs, we take the log of the variances assigned to each operator and variable and use it to assign a corresponding allocation of mantissa bits for use with MPFR's multiple-precision floating-point computation representation. As noted in Section 3.1, using MPFR enables us compute with an arbitrary number of bits of precision.

We compare expected error produced by NAP to maximum error bounds produced by FPTuner.

Results. 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. These results illustrate that for a fixed number of bits the expected error (from NAP) is lower than maximum error (from FPTuner) and for an expected error equalling the maximum error, fewer bits should be needed.

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

4 Related Work

Researchers have approached program approximation by performing loop perforations, function substitutions, and quantization [3, 5, 8]. Other approaches provide maximum instead of expected error bounds, which scales poorly and is less well-suited for modern applications such as machine learning [1, 3]. Still others produce annotations that identify operations requiring high precision [9, 10].

We compare a Hessian-based quantization approach to ours and show that at low bitwidths, it produces worse results than our approach [2]. Our sensitivity analysis is similar to the noise model used to compute generalization bounds on neural networks [4]. We hope to extend our work to provide generalization bounds on families of approximations.

References

[1] Wei-Fan Chiang, Mark Baranowski, Ian Briggs, Alexey Solovyev, Ganesh Gopalakrishnan, and Zvonimir Rakamarić. 2017. Rigorous Floating-point Mixed-precision Tuning. In *POPL*.

[2] Yann Le Cun, John S. Denker, and Sara A. Solla. 1990. Optimal Brain Damage. In *NIPS*.

[3] Eva Darulova and Viktor Kuncak. 2014. Sound compilation of reals. In *POPL*.

[4] Gintare Karolina Dziugaite and Daniel M. Roy. 2017. Computing Nonvacuous Generalization Bounds for Deep (Stochastic) Neural Networks with Many More Parameters than Training Data. In *UAI*.

[5] S. Han, H. Mao, and W. J. Dally. 2015. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *ICLR*.

[6] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* (Nov. 1998).

[7] Yann LeCun and Corinna Cortes. 1998. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/. (1998).

[8] Sasa Misailovic, Michael Carbin, Sara Achour, Zichao Qi, and Martin C. Rinard. 2014. Chisel: Reliability- and Accuracy-Aware Optimization of Approximate Computational Kernels. In *OOPSLA*.

[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).

[10] Pooja Roy, Rajarshi Ray, Chundong Wang, and Weng Fai Wong. 2014. ASAC: Automatic Sensitivity Analysis for Approximate Computing. In *LCTES*.