PLDI: G: NAP: Noise-Based Sensitivity Analysis for Programs

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 the effect of a change to a value on the output, creating an opportunity for safe and principled approximations.

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).

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 and compare it to standard point-based approaches.

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 the quality of the approximate outputs, and it produces a sensitivity analysis. NAP's pipeline for sensitivity analysis is:

1. Insert Gaussian noise parametrized by variance to the targets of approximation.
2. Solve an optimization problem to maximize these variances subject to an expected error constraint, which is the difference between the true output and the output generated from the noise-inserted program. The variances become the sensitivities.

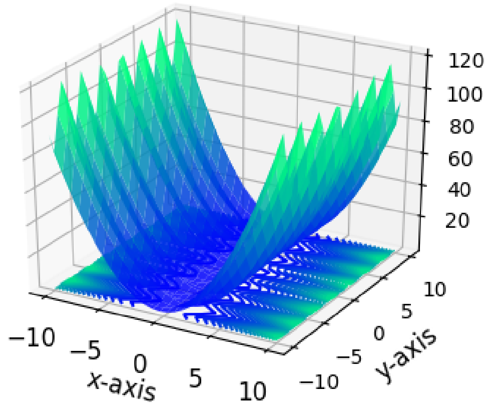

Figure 1. The figure depicts the loss surface $(x + \sin(1000y))^2$. Locally about (0,0), the quartic growth of y dominates the quadratic growth of x, so y should be more sensitive than x. Globally, since y is nearly constant while x grows quadratically, x should be more sensitive than y. A region-based sensitivity analysis can model this, while a point-based approach cannot.

For example, consider the function

$$f(x, y) = x + \sin(1000y) \tag{1}$$

where the inputs x, y are uniformly distributed following $x \sim \mathcal{U}(-100, 100)$ and $y \sim \mathcal{U}(-1, 1)$.

Inserting noise NAP introduces noise variables $\vec{\epsilon}$ into the program producing a *noisy program*

$$f_{\vec{\epsilon}}(x, y) = [x\epsilon_x + \sin((1000 \times [y\epsilon_y])\epsilon_\times)\epsilon_{\sin}]\epsilon_+, \tag{2}$$

where $\vec{\epsilon} \sim \mathcal{N}(\vec{1}, \vec{\sigma}^2)$. Note that $\mathcal{N}(\vec{1}, \vec{\sigma}^2)$ denotes $\mathcal{N}(\vec{1}, \Sigma)$ where the covariance matrix is nonzero only at $\Sigma_i i = \vec{\sigma}_i^2$. Each of the Gaussian-distributed noise variables in the noisy program $\tilde{f}$ corresponds to a single variable or operator in f. For example, $\epsilon_\times$ models the error introduced in the multiplication between 1000 and y. Constants, such as 1000, have no noise variable paired with them. Notice that $f(x, y) = f_{\vec{1}}(x, y)$.

Optimizing variances NAP uses the loss function

$$\max_{\vec{\sigma}} \Big(\underbrace{\sum_i \log(\sigma_i)}_{\text{noise envelope}} - \lambda \underbrace{\mathop{\mathbb{E}}_{\vec{x}\sim\mathcal{D},\vec{\epsilon}\sim\mathcal{N}(\vec{1},\vec{\sigma}^2)} \mathcal{L}(f, f_{\vec{\epsilon}})}_{\text{minimize error}} \Big) \quad (3)$$

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

$$\mathcal{L}(f, f_{\vec{\epsilon}}) = (f - f_{\vec{\epsilon}})^2 . \quad (4)$$

The first term in the objective specifies that solutions with larger noise envelopes are more desirable. The second term ensures that the approximate result is close to the exact result by estimating the expected error, computed by averaging over samples from the data distribution. The regularization parameter λ, controls the trade-off between the volume of the noise envelope and the expected error. NAP uses SGD to solve this optimization problem.

Analysis Expanding upon the caption of Figure 1, in the low-noise limit (λ large), the optimization will make y more sensitive than x – modeling the *local* curvature. In the high-noise limit (λ small), the optimization will make x more sensitive than y – modeling the *nonlocal* behavior. These sensitivities may be used to guide approximation, resulting in a coarse approximation of x in the first case and a coarse approximation of y in the second. Furthermore, our approach is *data-dependent*, meaning that the noisy envelope will change in response to changes in the data distribution $\mathcal{D}$.

***Precision tuning guided by* NAP** In the analysis, we showed that at different scales there are different optimal noise configurations and that the proposed optimization problem in Equation 3 captures this phenomenon. We experimentally validate that NAP can model f (Equation 1) with the noisy program $f_{\vec{\epsilon}}$ (Equation 2) by optimizing with SGD. Training consists of sampling $x, y \sim \mathcal{D}$ and $\epsilon \in \mathcal{N}(\vec{1}, \vec{\sigma}^2)$ in batches and using a well-known technique called the reparameterization trick to train the variances using samples [7].

Using NAP, we empirically confirmed that for large λ (low-noise), y is more sensitive than x and that for small λ (high-noise), the opposite is the case. Indeed, if we assign all variables to `double` except for y which we assign to `long double`, the expected error is $1.3 \cdot 10^{-13}$ over the input distribution. In contrast, if we assign all variables to `double` except for x which we assign to `long double`, the expected error is $7.9 \cdot 10^{-14}$.

In Section 4, we develop a methodology for automatic selection of λ and assignment of precisions. Using this methodology and NAP's sensitivities, we automatically generate the latter precision assignment for an appropriate expected error threshold.

3 Region-based sensitivity analysis

We present a formal treatment of noisy programs and our optimization problem. The formal framework provides the scaffolding necessary to prove that in the low-noise limit (i.e. with a sufficiently low volume noisy envelope), our optimization problem follows a point-based method – specifically, the Hessian. Because it is straightforward to prove results about this point-based method using the Taylor Series, our optimization problem inherits these guarantees in the case of low noise. A number of these point-based optimality guarantees are provided in [8].

3.1 Syntax and Semantics

We formally define syntax and semantics for noisy programs. We introduce the following syntactic classes:

$$\begin{array}{rcll} \text{Constants} & c & \in & \mathbb{R} \\ \text{Variables} & x & \in & \text{Strings} \\ \text{Distributions} & \mathcal{D} & \in & \text{Probability Distributions} \\ \text{Environments} & \gamma & ::= & [] \mid \gamma[x \mapsto v] \\ \text{Contexts} & \Gamma & ::= & \cdot \mid x \leftarrow \mathcal{D}, \Gamma \mid x \overset{\sim}{\leftarrow} \mathcal{D}, \Gamma \\ \text{Expressions} & e & ::= & c \mid x \mid N(e) \mid e \odot e \end{array}$$

Environments store maps from variable names to values and are generally a single sample of all of the noisy expressions in a program from a context. Contexts hold collections of (noiseless) samples or noisy samples. Expressions are arithmetic expressions that are either noisy or not noisy. Certain expressions are the targets of approximation and to denote this for expression e we write $N(e)$. Thus, $N(e)$ is a target of approximation, which means that the expression will become a noisy expression in the noisy program. Expressions may also be composed with any arithmetic expression $\odot$, which for the sake of our application are just $+, -, \times$, and $\div$.

Contexts and environments We use the common interpretation of the notation $x \leftarrow \mathcal{D}$, which is the sampling of x from a data distribution $\mathcal{D}$. For example, the input to a noisy program might be an image and $x \leftarrow \mathcal{D}$ will assign x to one of the pixels in the input image. We then define noisy sampling of inputs to be $x \overset{\sim}{\leftarrow} \mathcal{D}$, which first samples from the data distribution and then introduces noise into the input.

$$\begin{aligned}
[\![\cdot]\!] &= \text{return } [] \\
[\![x \leftarrow \mathcal{D}, \Gamma]\!] &= \lambda\vec{\sigma}. \\
&\quad \gamma \leftarrow [\![\Gamma]\!](\vec{\sigma}) \\
&\quad c \leftarrow \mathcal{D} \\
&\quad \text{return } \gamma[x \mapsto c] \\
[\![x \tilde{\leftarrow} \mathcal{D}, \Gamma]\!] &= \lambda(\vec{\sigma}_1, \sigma). \\
&\quad \gamma \leftarrow [\![\Gamma]\!](\vec{\sigma}_1) \\
&\quad c \leftarrow \mathcal{D} \\
&\quad c' \leftarrow [\![N(c)]\!](\sigma) \\
&\quad \text{return } \gamma[x \mapsto c']
\end{aligned}$$

Noisy expressions For each noisy expression $N(e)$, we introduce a fresh positive real variable. This variable is the standard deviation σ of a normal distribution that we will optimize over and will eventually become the sensitivity of that expression. We denote a generic binary operation (such as $+, -, \times, \div$) with the symbol $\odot$.

$$\begin{aligned}
\mathcal{G}[c] &= * \\
\mathcal{G}[x] &= * \\
\mathcal{G}[N(e)] &= \mathcal{G}[e] \times \mathbb{R}^+ \\
\mathcal{G}[e_1 \odot e_2] &\triangleq \mathcal{G}[e_1] \times \mathcal{G}[e_2]
\end{aligned}$$

Computation with noise Now that we have defined noisy expressions, we need to specify how to optimize the standard deviations $\vec{\sigma}$ that parameterize noisy expressions. In the case of an expression with no added noise, we return a distribution completely concentrated at the value specified in the environment. When we compose two noisy expressions, the parametrized standard deviation of each subexpression remains the same and we concatenate the vectors of the standard deviations.

$$\begin{aligned}
[\![c]\!] &= \text{return } c \\
[\![e]\!]_\gamma &= \text{return } \gamma(x) \\
[\![e_1 \odot e_2]\!]_\gamma &= \lambda(\vec{\sigma}_1, \vec{\sigma}_2). \\
&\quad x_1 \leftarrow [\![e_1]\!]_\gamma(\vec{\sigma}_1) \\
&\quad x_2 \leftarrow [\![e_2]\!]_\gamma(\vec{\sigma}_2) \\
&\quad \text{return } x_1 \odot x_2
\end{aligned}$$

We introduce a scale factor ϵ that we sample from a normal distribution with mean 1 and variance σ^2. The mean is 1 because we want the noisy expression to be equivalent to a noiseless expression when $\sigma^2 = 0$. The computation proceeds by recursively sampling to obtain a single output value and then scaling the output by ϵ.

$$\begin{aligned}
[\![N(e)]\!]_\gamma &= \lambda(\vec{\sigma}_1, \sigma). \\
&\quad v \leftarrow [\![e]\!]_\gamma(\vec{\sigma_1}) \\
&\quad \epsilon \leftarrow \mathcal{N}(1, \sigma^2) \\
&\quad \text{return } v\epsilon
\end{aligned}$$

3.2 Optimization Problem

In terms of our defined semantics, the optimization problem for the program $f(\Gamma) \triangleq e$ is

$$\max_{\substack{\vec{\sigma} \in \mathcal{G}[\Gamma], \\ \vec{\sigma}' \in \mathcal{G}[e]}} \sum_{\sigma_i \in [\vec{\sigma}; \vec{\sigma}']} \log(\sigma_i) - \lambda \mathop{\mathbb{E}}_{\substack{\gamma \leftarrow [\![\Gamma]\!](\vec{\sigma}), \\ y \leftarrow [\![e]\!]_\gamma(\vec{\sigma}')}} \mathcal{L}(\gamma, y), \tag{5}$$

where $\lambda > 0$ is the regularization parameter and the loss $\mathcal{L}$ is the domain specific loss that measures the performance on a given task.

Coherence Thus far, we have motivated region-based sensitivities by showing how they can be nonlocal. However, when we add an infinitesimal amount of noise to the program, we would hope that the resulting sensitivities would coincide with the point-based estimations. We prove that for a second-order approximation of the mean-squared-error loss, the constrained optimization problem defined in terms of Equation 5 approaches these Hessian sensitivities. Consider

$$\max_{\substack{\vec{\sigma} \in \mathcal{G}[\Gamma], \\ \vec{\sigma}' \in \mathcal{G}[e]}} \sum_{\sigma_i \in [\vec{\sigma}; \vec{\sigma}']} \log(\sigma_i) \tag{6}$$

$$\text{s.t.} \mathop{\mathbb{E}}_{\vec{\epsilon} \leftarrow \mathcal{N}(\vec{0}, \vec{\sigma}'^2)} \vec{\epsilon}^T H \vec{\epsilon} \leq \delta, \tag{7}$$

where $\delta > 0$ corresponds to the regularization parameter λ, controlling the magnitude of noise tolerated, and H is the Hessian of the loss. Notice that if we were to try to solve this constrained optimization problem with Lagrange multipliers, then we would obtain the objective in Equation 5 except with the loss instead of the Hessian approximation. With regard to the constrained optimization, we present the following theorem:

Theorem 1. *If H is an $n \times n$ positive semidefinite Hermitian matrix and $\vec{\sigma}$ is the diagonal of the covariance matrix, then the solution to the optimization problem*

$$\max_{\substack{\vec{\sigma} \in \mathcal{G}[\Gamma], \\ \vec{\sigma}' \in \mathcal{G}[e]}} \sum_{\sigma_i \in [\vec{\sigma}; \vec{\sigma}']} \log(\sigma_i)$$

$$s.t. \mathop{\mathbb{E}}_{\vec{\epsilon} \leftarrow \mathcal{N}(\vec{0}, \vec{\sigma}'^2)} \vec{\epsilon}^T H \vec{\epsilon} \leq \delta,$$

is $\vec{\sigma'}^2 \propto diag(H^{-1})$.

Proof Sketch. Simplify the optimization problem using concavity of the log and linearity of expectation to

$$\max \prod_i \sigma_i'^2$$
$$\text{s.t. } \sum_i \sigma_i'^2 h_{ii} \leq \delta.$$

Using the AM-GM inequality gives

$$n \sqrt[n]{\prod_i \sigma_i'^2 h_{ii}} \leq \sum_i \sigma_i'^2 h_{ii}$$

The left expression is a scalar times the objective, by monotonicity of square-root. A maximum is achieved when the constraint is tight. From AM-GM, this is the case exactly when all of the terms are equal, yielding

$$\sigma_i'^2 h_{ii} = \sigma_j'^2 h_{jj} \quad \forall i, j$$

Thus, to respect the constraint, each of these terms is equal to a constant $c = \delta/n$. Now we see that

$$\sigma_i'^2 = c/h_{ii},$$

which shows the desired proportionality. □

Thus, we find that to a second-order approximation, larger variances (entries of $\vec{\epsilon}$) come from smaller entries on the diagonal of the Hessian. As a result, $\vec{\epsilon}$ will be large in directions in which there would be a small change in the loss. This optimization problem bears a close resemblance to the well-known Hessian loss used in some of the early literature on neural network quantization [2]. The Hessian approach is point-based and therefore only local. In this case, the approximation becomes worse as the variation of the second derivative increases around the target point of approximation. For example, in a quadratic function, the Hessian perfectly models the curvature everywhere, whereas for a higher-order function (such as a cubic) it does not.

4 NAP for Precision Tuning

We compare NAP's region-based sensitivity analysis with first- and second-order point-based approaches from ADAPT and the Hessian approach – a generalization of ADAPT consistent with the model used in Theorem 1 [8]. We compare with the published results from ADAPT and our results generated using the Hessian.

Techniques ADAPT uses derivatives and approximations of down-cast errors to estimate output error and uses the generated sensitivities as an ordering on variables. Less sensitive variables are cast down until the error threshold is violated. The output is an assignment of variables to `double` and `long double` variables.

NAP has a number of key differences from ADAPT:

- NAP generates precisions for program variables and operators, while ADAPT generates precisions only for variables.
- NAP generates an approximate program by optimizing over a distribution of inputs, whereas ADAPT optimizes for a single input.
- NAP generates *multi-precision* (positive integer mantissa) bitwidths. ADAPT selects between two precisions.

We compute the Hessian sensitivities by averaging the squares of the derivative values over multiple input samples. There is a mathematical equivalency between this and taking the second-order derivative of the mean-squared error loss with respect to each of the noise variables $\vec{\epsilon}$ that NAP introduces (e.g. see Equation 2).

NAP converts the sensitivity σ^2 of each variable to an assignment of n mantissa bits by following the equation $n = -\lfloor \log_2(\sigma) \rfloor$.

Methodology We compare approximations guided by sensitivities from NAP and the Hessian approach on a subset of FPBench benchmarks [3]. NAP and the Hessian approach both sample inputs from a uniform distribution over each input interval specified in FPBench. We use the error bounds from the FPTuner paper [1].

The Hessian approach and NAP generate multi-precision assignments. To obtain precisions from Hessian sensitivities, we take logarithm of the sensitivities, shift them until they are all positive, and make them the bitwidths of the corresponding variables. We then increment or decrement bitwidths until the error bound is satisfied as tightly as possible.

We address the differences between ADAPT's and NAP's designs by:

- Computing average bitwidths over only the program variables for NAP.
- Providing experiments with NAP for both single-input and multi-input cases. Single-inputs are assigned to be the mean of the range that the input was defined over (unless the mean is 0 where we use the mean of the positive segment). For the multi-input case, inputs are uniformly distributed over the appropriate intervals. The arclength benchmark is only specified at a point, so we omit multi-input results.
- Calculating the mean over the bitwidths of the variables that are multi-precision, mixed-precision, and `double` and `long double` (D & L). Mixed-precision assignments are IEEE 754 standard floats, assigned by incrementing or decrementing precisions until the error bound is satisfied as tightly as possible. We use a similar methodology for D & L configurations.

Benchmarks	Ops	Error Thresholds	ADAPT	Single Input NAP			Multiple Input NAP		
				Multi	Mixed	D & L	Multi	Mixed	D & L
carbonGas	15	1e-10	58.4	47	48.4	55.6	52	53.2	61.6
jetEngine	28	1e-13	58	47.9	53.4	53.4	55	62.3	64
arclength	33	1e-12	52	23.9	30.4	52	–	–	–
simpsons	43	1e-12	53.2	43.7	52	52	50.2	61	61

Table 1. We compare ADAPT's `double` and `long double` (*D & L*) assignments with the single-input and multiple-input version of NAP's multi-precision (*Multi*), mixed-precision (*Mixed*), and D & L precisions. All but the first two columns show the average number of bits in the mantissa for the targets of approximation. The multi-input task is harder than the single input one since NAP fits to a distribution over inputs rather than fitting to a specific input.

Benchmarks	Ops	Threshold	Hessian	NAP
carbonGas	15	1e-08	52.9	48.4
doppler1	11	1e-13	54.6	49.9
doppler2	11	1e-13	55.6	51.2
doppler3	11	1e-13	52.9	50.7
jetEngine	28	1e-11	50.9	50.5
predprey	7	1e-16	56.3	50.7
rigidbody1	11	1e-13	53	50.7
rigidbody2	13	2e-11	53	49.3
sine	11	2e-16	53	47.8
sineOrder3	6	1e-15	50	50
sqroot	12	2e-16	56.2	48.8
turbine1	16	1e-14	64.5	49.1
turbine2	13	1e-14	53.2	49.8
turbine3	16	1e-14	64.6	49.8
verlhulst	5	1e-16	56	51.2

Table 2. We compare multi-precision assignment from the Hessian approach and NAP. The two rightmost columns show the average number of bits in the mantissa for the targets of approximation.

Results Table 2 shows that NAP's precision assignments save an average of 5.2 bits versus those from the Hessian approach.

In Table 1 for the single-input case, there is an average savings of 14.8 bits in multi-precision mode, 9.3 bits in mixed-precision mode, and 2.1 bits in the comparison using `double` and `long double` (the same as ADAPT). Evaluating over multiple inputs, which is a more practical and challenging case, NAP can still save an average of 4.1 bits in multi-precision mode, but requires additional bits for more coarse precisions.

5 Related Work

Researchers have approached program approximation by performing loop perforations, function substitutions, and quantization [4, 6, 9]. Other approaches provide maximum instead of expected error bounds, which can be valuable in safety-critical systems, but perform poorly in the average case [1, 4]. Still others produce annotations that identify operations requiring high precision [10, 11].

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 [5]. We hope to extend our work to provide generalization bounds on families of approximations.

References

[1] Chiang, W.-F., Baranowski, M., Briggs, I., Solovyev, A., Gopalakrishnan, G., and Rakamarić, Z. Rigorous floating-point mixed-precision tuning. *SIGPLAN Not.* (2017).

[2] Cun, Y. L., Denker, J. S., and Solla, S. A. Optimal brain damage. In *NIPS* (1990).

[3] Damouche, N., Martel, M., Panchekha, P., Qiu, J., Sanchez-Stern, A., and Tatlock, Z. Toward a standard benchmark format and suite for floating-point analysis.

[4] Darulova, E., and Kuncak, V. Sound compilation of reals. In *POPL* (2014).

[5] Dziugaite, G. K., and Roy, D. M. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. In *UAI* (2017).

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

[7] Kingma, D. P., and Welling, M. Auto-encoding variational bayes, 2013.

[8] Menon, H., Lam, M. O., Osei-Kuffuor, D., Schordan, M., Lloyd, S., Mohror, K., and Hittinger, J. Adapt: Algorithmic differentiation applied to floating-point precision tuning. *International Conference for High Performance Computing, Networking, Storage and Analysis* (2018).

[9] Misailovic, S., Carbin, M., Achour, S., Qi, Z., and Rinard, M. C. Chisel: Reliability- and accuracy-aware optimization of approximate computational kernels. In *OOPSLA* (2014).

[10] Nongpoh, B., Ray, R., Dutta, S., and Banerjee, A. Autosense: A framework for automated sensitivity analysis of program data. *IEEE Transactions on Soft. Eng.* (2017).

[11] Roy, P., Ray, R., Wang, C., and Wong, W. F. Asac: Automatic sensitivity analysis for approximate computing. In *LCTES* (2014).