BACKWARD ANALYSIS FOR CUMULANT ESTIMATION

Team Afronaut
ARC White-Box Estimation Challenge 2026, Phase 2
Algorithmic-contribution write-up

ABSTRACT

Estimating the expected activations of a random ReLU network from its weights requires an account of how non-Gaussian structure develops with depth. Existing methods address this problem through forward analysis, in which a compressed representation of the third-order cumulants is propagated from layer to layer and the fourth-order cumulants are regenerated from data. We show that such an approach is limited by the quality of the propagated representation. We develop a backward analysis in which only the mean and covariance are propagated, and the higher-order corrections required at each layer are derived as needed using gradient information. The resulting method yields a parameter-free correction for the effect of a ReLU on a neuron's fourth cumulant. It reduces the final-layer error by a factor of 2.6 relative to forward analysis, and it can be optimized to run using less than 40% of the compute budget. We further show that two fourth-order terms identified by the backward analysis improve forward analysis when incorporated, and that most of the remaining error comes from a family of fourth-order terms whose evaluation scales polynomially with the width.

1 INTRODUCTION

Random neural networks are a standard object of study in deep learning theory, and the expected activation of each neuron under Gaussian inputs is a common quantity to compute. Computing it can be quite challenging. The pre-activations of the first layer are Gaussian, but every activation function introduces higher-order cumulants that gradually warp the Gaussian structure of the distribution. An estimator that propagates only the mean and covariance accumulates several orders of magnitude of error over sixteen layers. This challenge explores the problem using networks with 16 layers, each of width 1024. It scores each estimator using its final-layer mean squared error, weighted by the fraction of the compute budget it uses. The challenge is to estimate the final-layer activations with the lowest possible mean squared error and compute usage.

The prevailing approach is forward analysis by cumulant propagation. The mean and covariance are propagated together with a compressed representation of the third-order cumulants, and a per-neuron fourth cumulant is regenerated at each layer from data. The published method of team 504aldo established this approach, and most entries near the top of the leaderboard build on it. The variants we examined differ in cost engineering and learned correctors, but not in the underlying propagation approach.

In this work, we investigate where forward analysis loses accuracy and whether a different organization of the computation can recover it. We show that forward analysis is limited by the quality of the propagated representation. When exact information about per-neuron variance, as well as third- and fourth-order cumulant information, is provided, the method becomes 11.7 times more accurate. This may explain why learned per-neuron correctors struggle to recover the error. It also motivates a backward analysis in which only the Gaussian part of the state is propagated and the higher-order corrections are derived at each layer, as needed, using gradient information from the earlier layers. We make the following contributions.

- **Backward analysis.** We introduce the look-back design, which propagates each layer's mean and covariance statistics. It derives the third-order cumulants, a fourth-order covariance correction, and each neuron's fourth cumulant using gradient information. The fourth cumulant is obtained using a parameter-free correction for the effects of the ReLU function.

- **Cost reduction.** We reduce the cost of the method from roughly 80 budgets to under 40% of one through exact rewrites of its most expensive terms, learned selections of operator products, and a sequence of composed reductions. We also characterize the amount of room left for further optimizations.
- **Integration with forward analysis.** We incorporate two fourth-order terms identified by the backward analysis into forward analysis and show that they improve its graded score.
- **Error attribution.** We show that two thirds of the remaining error of the backward analysis comes from its third-order inputs, and that most of this is due to a family of fourth-order terms. These terms involve three neurons, and the exact evaluation of this family scales polynomially with the width of the network.

2 Notation and Problem Formulation

We write μ and C for the mean and covariance of the pre-activations at a layer. At third order, we distinguish each neuron's own cumulant K_{iii} from the pair slice K_{iik}. At fourth order, we distinguish the diagonal K^4_{iiii}, the $(3,1)$ slice K^4_{iiij}, the $(2,2)$ slice K^4_{iijj}, and the three-neuron entries K^4_{iijk}. For a neuron with standardized mean $a_i = \mu_i/\sqrt{C_{ii}}$, $\Phi(a_i)$ is the expected slope of its ReLU under the Gaussian approximation.

We consider a bias-free ReLU network $z^0 = W_0 x$, $h^l = \mathrm{ReLU}(z^l)$, $z^{l+1} = W_{l+1} h^l$ for $l = 0, \ldots, 15$, with $x \sim \mathcal{N}(0, I)$ and width $n = 1024$. An estimator returns $\mathbb{E}[h^l]$ for every layer, and we care about the mean squared error at the final layer. The score for a network is its final-layer mean squared error, which we refer to as the raw error, multiplied by $\max(0.1, C/B)$. Here C is the number of floating-point operations the challenge attributes to the estimator and $B = 2^{41}$ is the budget. Additional computation below 10% of the budget is free, while cost and accuracy trade off linearly otherwise. A network whose cost exceeds the budget, or whose wall-clock time outside counted arithmetic exceeds 0.4 s, receives a score of zero. For the purpose of exposition, we express cost in units of one $n \times n \times n$ matrix product, so the budget is 1024 units.

3 Related Work

The reference method for this problem is the factorized third-order chain of 504aldo, released under the MIT licence during the competition. It propagates the mean and covariance along with a compressed representation of the third cumulant. The third cumulant is stored as a sum over sources, one per layer. Each source is a pair of $n \times n$ factors that later weight matrices transport forward, and sources far from the current layer are compressed into shared low-rank bases. A per-neuron fourth cumulant is regenerated at each layer from the off-diagonal covariance using coefficients fit to data, and is fed back into the third-order terms. Later entries by other teams add pruning of inactive neurons, fast matrix multiplication, and learned correctors trained on public Monte Carlo data. Several teams report that giving the method more accurate fourth-order inputs tends to make it worse. The likely reason is that the fitted coefficients compensate for omitted terms, and our analysis in Section 4 points the same way. An exact fourth-order chain run outside the budget on a large GPU reaches a raw error six times below ours, but no team has brought its cost near the budget. We are not aware of a published method that evaluates cross-layer third-order corrections for less than one matrix product per pair of layers.

Backward analysis is well established in reachability analysis, where it complements forward analysis by characterizing the states from which a target is reachable rather than the states reachable from an initial set. We believe a similar approach can be applied here. Forward cumulant propagation computes the statistics at a layer from those of the previous layer. Our method computes them from every earlier layer using the sensitivity of the current layer to each of the previous layers.

4 Backward Analysis

We refer to the resulting method as the look-back design. The guiding principle is to avoid propagating quantities that can be derived. The forward pass propagates only the mean and covariance, which uses about 2% of the budget. At each layer, the non-Gaussian corrections required by the next

step are derived from the state of every earlier layer. A target layer l has l earlier source layers, for a total of 120 layer pairs. With this approach, the main question becomes whether these pairs can be evaluated accurately without exceeding the computation budget.

Forward Pass. Given the mean, the covariance, and the cumulants at a layer, a degree-12 Hermite expansion can be used to compute the mean and covariance of the post-activations. The mean depends on K_{iii} and K^4_{iiii}, and the covariance depends on K_{iik}, K^4_{iiij}, and K^4_{iijj}. With all corrections disabled, the forward pass has a raw error of 3.4×10^{-6} on our test networks. The Gaussian part is cheap to propagate, and nearly all of the attainable accuracy comes from the higher-order corrections.

Third-Order Cumulants Using Gradient Information. The third cumulant at layer l is a sum over sources, one per earlier layer b. Each source is the image of the cumulant generated at that layer's activation under the linearized map from h^b to z^l. We evaluate this map exactly as a demand matrix $Y^{l,b}$, which is the gradient of the target pre-activations with respect to the source activations. It is a product of the intervening weight matrices and the diagonal Gaussian slopes $\Phi(a)$. At the target, we extract only the entries the Hermite needs. Because we only retain the mean and covariance between layers, there is nothing to compress or truncate. Each source is recomputed from its own layer whenever a target needs it. With fourth-order inputs taken from 504aldo's regeneration law evaluated on the method's own state, this reaches a raw error of 1.9×10^{-8} on networks 8 and 12, which is below forward analysis at every layer.

Fourth-Order Covariance Correction. The $(3, 1)$ slice of the fourth cumulant is used for the covariance update. Forward analysis models it as a fitted multiple of the off-diagonal covariance. We derive a backward equivalent using the same demand matrices, evaluating each source's contribution at its originating activation. This reduces the error to 1.55×10^{-8} on both test networks.

Per-Neuron Fourth Cumulant. The remaining input is each neuron's own fourth cumulant, which is used to compute the mean. If we use its true value from Monte Carlo labels, the error drops to 8.5×10^{-9} and 9.0×10^{-9}, so the diagonal accounts for most of the remaining gap. We compute it without labels using a parameter-free correction for the effect of the ReLU on the fourth cumulant it receives, which we refer to as dressing. With the dressed diagonal, the method reaches 9.04×10^{-9} and 8.81×10^{-9} on networks 8 and 12. This is within 6% of the labeled ceiling and 2.6 times below forward analysis.

Theoretical Basis. The design relies on two results. First, statistical linearization is equivalent to Gaussian moment closure. An affine surrogate of a ReLU network is exactly Gaussian and carries no non-Gaussian content, and the degree-2 Hermite residual it leaves behind is the second-order generation term of the source-form third-order chain. We verified this to machine precision on small networks. Third-order cumulants must therefore be generated at the activations and transported, and the demand matrices can be used to transport them. Second, a fourth-order closure can be derived at fixed weights without fitted coefficients, and it agrees with exact quadrature to 2.4×10^{-11}. Its benefit within forward analysis is small, however, because the fitted coefficients already compensate for the omitted terms.

5 Reducing the Cost of the Look-Back Design

Exact Rewrites. The first implementation of the look-back design required roughly 81,700 units, or about 80 budgets. Almost all of this cost came from one term of the $(3, 1)$ slice, which we refer to as the 3-core. For every (origin, relay, target) triple, this term reads the full third cumulant of the relay layer, which costs n^4 operations per triple. We remove the relay from this computation using two identities. The first two terms of the 3-core do not depend on the relay. They are intermediate states of the forward recursion that produces the pair slice of the third cumulant, so the third-order computation already provides them. The third term sums over relays before it reaches the source, so it can be written as a directional derivative of the same evaluation along one accumulated factor per target. Together, these rewrites reduce the 3-core from 72 budgets to 2.4 without changing the result. Sharing products that appear more than once reduces the full slice from 76.5 budgets to 5.1.

Table 1: Cost reduction of the look-back design. The estimated score is the mean of raw error times budget fraction over networks 0-15.

Stage	Budget fraction	Estimated score
Composed single-precision implementation	0.99	1.18×10^{-8}
+ cheaper fourth-order correction, inactive targets removed	0.73	8.8×10^{-9}
+ distant pairs compressed, readout retrained, inactive sources removed	0.59	6.9×10^{-9}
+ relaxed removal threshold, three-layer fourth-order read, Strassen depth 3	0.47	6.0×10^{-9}
+ learned adjoint representations, Strassen depth 4	0.40	5.3×10^{-9}

Learned Selections. The remaining reductions come from selecting which products to keep. We drop operator products that contribute less than 0.2% of the error for every source. We replace the five separate operators of each pair with three learned combinations of them. We compress sources that are more than four layers from their target to rank 448, using a basis predicted from the source's own Gaussian state. We also replace the learned per-neuron estimate, whose inputs cost 161 units, with a blended estimate that costs fewer than two. Each of these choices was made using public training networks or two of the test networks, and validated on the remaining test networks. The resulting single-precision implementation uses 97.7% of the budget and reaches 1.04×10^{-8} and 1.17×10^{-8} on networks 8 and 12, which is about 1.2 times the error of the double-precision design. We believe this loss comes from the single-precision implementation rather than from the method.

Composed Reductions. We then composed a further sequence of reductions, which brought the cost from 99% to under 40% of the budget while increasing the raw error by about 10%. These include a cheaper form of the fourth-order covariance correction, removing neurons that are almost always inactive from every product, compressing the gradients of distant pairs using a shared basis, retraining the learned correction used in the readout, replacing the adjoint computations of distant pairs with learned representations, and one additional level of Strassen multiplication. The final implementation uses 415 units on network 8 at a raw error of 1.19×10^{-8}, and its estimated score over sixteen networks is 5.3×10^{-9} (Table 1).

Cost Floor. Three quarters of the final cost is spent on the layer pairs. The 54 pairs that are within four layers of their target account for more than half of this, at about seven full-width products each. If every pair could be evaluated with a single product, the pairs would cost 120 units in total, so these nearby pairs alone cost more than an entry at 10% of the budget can afford. We evaluated several ways of reducing them, including fewer forward or adjoint products, lower rank, fewer output directions, dropping sources, and deeper Strassen recursion, and none of them succeeded. Deeper Strassen recursion in particular reaches the grader's memory limit at depth 4 and doubles the server time at depth 5. Approaches that avoid evaluating the nearby pairs one at a time would need to propagate their statistics between layers, and each of the ones we tried fails for the same reason. The dominant product for each pair is the Gaussian cross-layer covariance, which has full rank between nearby layers. We believe more can be done here, but we were unable to explore additional ideas because we were constrained by this author's other responsibilities, which includes finishing a dissertation.

6 Integrating Forward and Backward Analysis

Grader Outcome. The look-back design did not receive a grade. We submitted five packages on the final day, all of which were within the budget. One failed the grader's smoke test on a network width that it had not been run on before. The other four were rejected on the 0.4 s limit on wall-clock time outside counted arithmetic. The grader applies this limit to the whole submission, whereas the local scoring library only zeroes the network that exceeds it, so none of our local tests detected the problem. When we measured the look-back design side by side with our graded forward package on matched server threads, its uncounted time per network was 1.2 to 1.6 times larger at a comparable number of requests. This suggests that the excess is per-request overhead in the implementation rather than a property of the method.

Table 2: Graded entries. Budget fraction is the adjusted score divided by the raw error.

Submission	Contents	Adjusted	Raw	Budget fraction
335354	504aldo's method with our garbage-collection policy	5.42×10^{-9}	2.13×10^{-8}	0.25
335404	+ acco-cyber's pruning, our learned corrector	4.51×10^{-9}	2.00×10^{-8}	0.23
336771	+ cost-only rewrite, corrector retrained on public data	4.27×10^{-9}	1.93×10^{-8}	0.22
337407	+ published cost settings, corrector retrained	3.30×10^{-9}	2.01×10^{-8}	0.16
337583	+ fourth-order read and per-neuron term from the backward analysis	3.14×10^{-9}	1.93×10^{-8}	0.16
337637	+ carried-state corrector, fewer calls	$\mathbf{3.09 \times 10^{-9}}$	1.92×10^{-8}	0.16
336639	Our own forward method with its learned corrections	6.75×10^{-9}	1.76×10^{-8}	0.38

Graded Entries. The backward analysis still contributed to our graded entries through forward analysis. Our graded submissions use 504aldo's method with acco-cyber's pruning and our own modifications. These include a garbage-collection policy that keeps the uncounted time within its limit, a learned corrector for the inputs of the readout that is trained end to end on keenanpepper's public data, a rewrite that reduces cost without changing predictions, and the cost settings used by two other teams. Together, these reached a score of 3.30×10^{-9} (submission 337407). We then used two terms from the backward analysis to improve this entry. The first is a fourth-order read of the three-neuron family on the third-order factors the method already propagates, with a rank-4 correction for coincident indices. It reduces the raw error on all sixteen test networks and increases the cost by less than 1%. The second is the per-neuron fourth cumulant from Section 4, with the component that the regeneration law already accounts for removed. Together, they improve the graded score to 3.14×10^{-9} (submission 337583). Adding a corrector that propagates a learned state across layers, following a published design, and two rounds of call reduction gives our best graded entry at 3.09×10^{-9} (submission 337637).

7 Error Attribution

Third-Order Inputs. To understand where the remaining error of the look-back design comes from, we replaced the third-order entries that the readout uses with values estimated from 2^{25} to 2^{26} Monte Carlo samples, at every layer, and left everything else unchanged. The error dropped from 9.0×10^{-9} to 3.2×10^{-9} on network 8 and from 8.8×10^{-9} to 2.8×10^{-9} on network 12. About a quarter of what remains is the sampling noise of the labels themselves. This means that the third-order inputs account for about two thirds of the error. The error is concentrated in the deeper layers, where the inputs are biased low. Using more accurate fourth-order inputs does not help. After one layer of propagation, the third-order error of the method is almost the same as that of a variant that uses exact fourth-order statistics. What all of these variants share is the way they model the three-neuron entries K^4_{iijk}, which they all build from the same fixed form.

The Three-Neuron Family. We can compute every cumulant exactly on small networks. There, using the exact three-neuron entries recovers 88% of the improvement obtained from exact third-order inputs, and no other candidate recovers a measurable share. At full width, we built a tangent map of the Gaussian sweep that predicts eleven end-to-end runs to within 1.1%. It places the exact family at 80 to 86% of the improvement. Deriving the dominant class of this family backward reduces the error of the method by 20 to 24%, but it costs roughly one thousand budgets. We believe this cost is structural. The two expensive terms of the family are loops that close through the neuron being predicted, and their index graph has treewidth 3. Every contraction order therefore costs n^4 per (source, relay, target) triple. We evaluated several cheaper forms, including derived, learned, randomized, and factored approximations, and none of them kept more than a fifth of the effect. Corrections learned offline from public data with billion-sample moments recover about a tenth of

the improvement. The remaining accuracy in this problem therefore lies in a term that we understand well but cannot yet evaluate cheaply.

8 Conclusions and Future Work

We investigated where cumulant propagation loses accuracy on deep random ReLU networks and showed that forward analysis is limited by the quality of the propagated representation. We developed a backward analysis that propagates only the mean and covariance and derives the higher-order corrections from earlier layers as needed, using gradient information. To our knowledge, it is the most accurate analytic estimator at this scale. We reduced its cost from roughly 80 budgets to a floor near a third of one, and we used two of its fourth-order terms to improve forward analysis, which improved our graded score. Finally, we attributed most of the remaining error to a family of fourth-order terms whose evaluation scales polynomially with the width.

Limitations. The look-back design has not been graded. The grader rejected its packages on the limit on uncounted wall-clock time, which our implementation exceeds by a factor of 1.2 to 1.6 relative to our graded forward package, and we have no local test that reproduces the grader's behavior on this limit. Even at its measured accuracy, its cost floor of about a third of the budget places it above the leading entries in adjusted score. Several of the learned selections were made using two of the eight test networks, and where this is the case we report results on the remaining six.

Future work. Reducing the per-request overhead of the implementation would allow the method to be graded as it stands. Beyond that, the analysis points to one target, which is an affordable evaluation of the three-neuron fourth-order family. It accounts for most of the remaining error, and none of the approximations we tried captures it well. Tree-level expressions for fourth-order generation in the later layers, which other teams have derived for forward analysis, are a natural candidate for a method that propagates the exact mean and covariance, and we intend to evaluate them. Finally, this participant would like to revisit this problem after wrapping up their academic responsibilities.

9 Statement on the Use of Language Models

The code of this project was written by large language model agents (Claude, Anthropic) operating as coding agents under the direction of the author. The agents wrote the research code and every submitted package, ran the experiments, and drafted the study reports. The author set the research direction and the rules under which the agents operated, reviewed and ruled on results, and decided what to submit. This document was written collaboratively, with the agents producing drafts from the study reports and the author revising and editing them. Of the 344 commits in the project repository at the time the record was assembled, 309 carry a language-model co-author line. Claims were checked against exact quadrature, dense computation on small networks, independent re-implementations, and Monte Carlo labels drawn in two halves, and each study's report was reviewed before it was accepted.