
Efficient and Approximate Per-Example Gradient Norms for Gradient Noise Scale

Anonymous Author(s)

Affiliation

Address

email

Abstract

1 The gradient noise scale is valuable to compute because it provides a suggestion
2 for a compute efficient batch size when training a deep learning model. However,
3 computing it can be awkward or expensive depending on the approach taken due to
4 difficulty obtaining small batch gradient norm estimates. “Efficient” per-example
5 gradient norms provide accurate small batch gradient norms but are inefficient
6 in transformer or convolutional models. By assuming activations are normally
7 distributed, we compute an approximate per-example gradient norm that tracks the
8 true per-example gradient norm in practical settings. Using this approximation,
9 we construct a Scaled Output Gradient Noise Scale (SOGNS) that is generally
10 applicable at negligible cost and provides additional feedback to the practitioner
11 during training.

12 1 Introduction

13 Gradient Noise Scale (GNS) correlates with the “critical batch size”, which prescribes a batch size at
14 which the model will require “twice as many steps as an optimally data-efficient (small-batch) run
15 would take, and twice as many optimization steps as an optimally time-efficient (large-batch) run
16 would take” [McCandlish et al., 2018]. For this reason, the batch size prescribed by GNS has been
17 demonstrated to be useful while training GPT3 [Brown et al., 2020].

18 Computing the GNS requires gradient norms from small and large batches (described in Section 2).
19 However, in settings where we desire high performance compute, batch sizes typically need to be
20 large, making it difficult or costly to sample small batch gradients. Goodfellow [2015] introduces
21 a trick to access per-example gradient norms efficiently, but this trick cannot be applied in settings
22 with tensor rank larger than 2. In particular, transformer language models have rank-3 tensor with
23 batch, sequence and hidden dimensions. To address this problem, we construct an approximation
24 that assumes normally distributed activations at layer inputs, which allows us to access per-example
25 norms efficiently (described in Section 3.1).

26 2 Background

27 McCandlish et al. [2018] suggest using the “simple” GNS, $\mathcal{B}_{\text{simple}}^1$, as a metric to inform the
28 practitioner while training a model,

$$\mathcal{B}_{\text{simple}} = \frac{\text{tr}(\Sigma)}{G^T G}$$

¹This approximation is denoted as “simple” because it assumes that the Hessian is diagonal in the Taylor expansion of the loss.

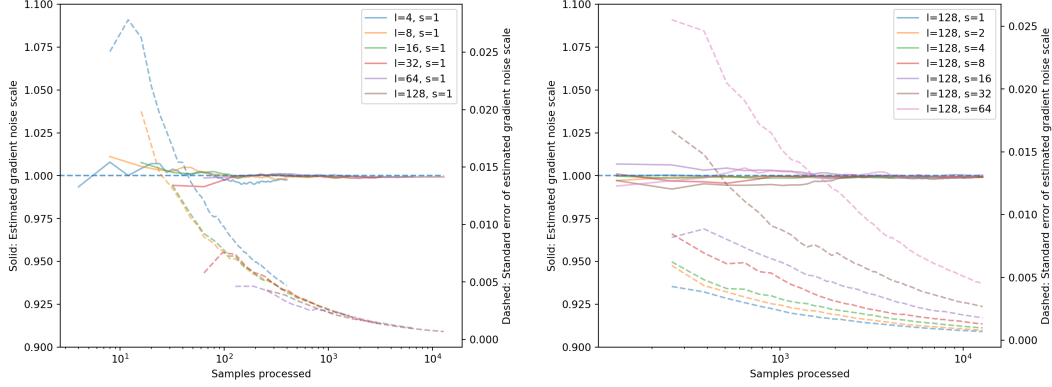


Figure 1: The variance of the GNS estimator for different B_{big} (left) and B_{small} (right) sizes. $B_{\text{big}} = l$ and $B_{\text{small}} = s$ in legends.

29 where G are the gradients and Σ is their associated covariance matrix. To compute $\mathcal{B}_{\text{simple}}$ McCandlish
 30 et al. [2018] further define the unbiased estimators \mathcal{S} and $|\mathcal{G}|^2$ shown in Equations 1 and 2, where
 31 B_{big} and B_{small} are the batch sizes used to compute the gradients.

$$|\mathcal{G}|^2 := \frac{1}{B_{\text{big}} - B_{\text{small}}} B_{\text{big}} |G_{B_{\text{big}}}|^2 - B_{\text{small}} |G_{B_{\text{small}}}|^2 \approx G^T G \quad (1)$$

$$\mathcal{S} := \frac{1}{1/B_{\text{small}} - 1/B_{\text{big}}} |G_{B_{\text{small}}}|^2 - |G_{B_{\text{big}}}|^2 \approx \text{tr}(\Sigma). \quad (2)$$

32 We can easily compute $|G_{B_{\text{big}}}|$ using the accumulated gradients immediately after the backward pass.
 33 However, the challenge in computing $|G_{B_{\text{small}}}|$ is that it requires the gradients for a batch size that
 34 is smaller than the batch size used for the optimizer step. McCandlish et al. [2018] propose using
 35 the gradients communicated between Distributed Data Parallel (DDP) nodes but this means that the
 36 variance of the resulting GNS estimate is tied to that DDP configuration. A taxonomy of the options
 37 for computing $|G_{B_{\text{small}}}|$ is presented in Appendix A.

38 As the estimate of the small batch gradient norm may be the mean over samples within the minibatch,
 39 in accordance with the law of large numbers, the variance of the estimate decreases with the number
 40 of observations of the gradient norm. As shown in Figure 1, this implies the small batch size should
 41 be as small as possible to obtain an estimate of $|G_{B_{\text{small}}}|$, and thus the GNS, with minimal variance.
 42 Further discussion of this result may be found in Appendix B and code in Appendix B.1.

43 3 Efficient Per-example Norms

44 Goodfellow [2015] proposes a trick to compute gradient norms for individual examples in a minibatch,
 45 which would provide the minimum variance estimate of the GNS as described in Section 2. He
 46 observes that the squared norm of the gradient is a sum of elements in an outer product that can be
 47 factored into two smaller sums on the input vectors, eliminating the need to calculate the full outer
 48 product. It may be stated as follows using Einstein and Lagrange notation,

$$n_b^2 = (w^\theta)_{bik}^2 = x_{bi} x_{bi} y_{bk}^\theta y_{bk}^\theta,$$

49 where x are the activations prior to a linear layer, y^θ are the gradients of the loss with respect to the
 50 outputs of the linear layer and w^θ are the gradients of the loss with respect to the weights of the linear
 51 layer. Further explanation of this notation may be found in Appendix C.

52 For networks of only linear layers acting on 2D inputs, this trick is sufficient to provide accurate
 53 GNS estimates. However, for networks with convolutional or 3D inputs to linear layers, such
 54 as transformers, this trick is no longer efficient. For three dimensions, $\mathbf{X} \in \mathbb{R}^{B \times T \times I}$ and $\mathbf{Y} \in$
 55 $\mathbb{R}^{B \times T \times K}$ [Li et al., 2022],

$$n_b^2 = (w^\theta)_{bik}^2 = \left(\prod_t x_{bti} y_{btkt}^\theta \right)^2 = x_{bti} y_{btkt}^\theta x_{bui} y_{buk}^\theta$$

56 has $O(T^2)$ complexity in the sequence length T . In these cases computing the norms explicitly, as
 57 the per-example gradient trick avoids, is more efficient. More details on this case are provided in
 58 Appendix C.1.

59 3.1 Proposed Additional Approximation

60 Assuming all entries of \mathbf{X} are IID Gaussian with a batch-dependent standard deviation σ_b and mean
 61 zero allows us to compute the following expectation in closed form:

$$E\left[\prod_i x_{bi}x_{bi}\right] = \prod_i E[x_{bi}x_{bi}] = \prod_i \sigma^2(x_{bi}) = I\sigma^2(x_{bi}).$$

62 Applying this in the 3D case,

$$E[n_b^2] = E\left[y_{bt_k}^0 y_{bu_k}^0 x_{bti} x_{bui}\right] = y_{bt_k}^0 y_{bu_k}^0 E\left[x_{bti} x_{bui}\right] = \prod_{t,k} y_{bt_k}^0 y_{bu_k}^0 \prod_i \sigma_b^2 = I\sigma_b^2 \prod_{t,k} y_{bt_k}^0 y_{bu_k}^0$$

63 and we know $\sigma_b^2 = \frac{1}{T} \prod_{t,i} x_{bti} x_{bti}$ in line with our assumptions above, assuming x_{bti} is zero-mean.

64 Factorizing the quadratic in the t, u dimension produces

$$E[n_b^2] = I\sigma_b^2 \prod_k \prod_t y_{bt_k}^0 \quad !^2$$

65 In practice, this says we can approximate n_b in the 3D case by summing the activations over the T
 66 dimension, squaring the result, and multiplying by the squared norm of \mathbf{X} , divided by T :

$$n_b^2 \approx \eta_b^2 = I\sigma_b^2 \prod_k \prod_t y_{bt_k}^0 \quad !^2 \quad \circ \quad \frac{1}{T} \prod_{t,i} x_{bti} x_{bti} \quad \times \quad \prod_k \prod_t y_{bt_k}^0 \quad !^2$$

67 and we can see that this is equal to the exact per-example gradient when $T = 1$:

$$n_b^2 \approx \eta_b^2 = I\sigma_b^2 \prod_k \prod_t y_{bt_k}^0 \quad !^2 = I \frac{1}{I} \prod_i x_{bi} x_{bi} \quad \times \quad \prod_k (y_{bk}^0)^2 = x_{bi} x_{bi} y_{bk}^0 y_{bk}^0$$

68 Experiments in Section 4, along with simulations in Appendix D, confirm that this approximation is
 69 accurate. This approximation may also be extended to apply to $|G_{B_{\text{big}}}|$ as described in Appendix E
 70 but this observation is unnecessary for the results presented here, as we assume the exact $|G_{B_{\text{big}}}|$ is
 71 easy to access.

72 Substituting η_b^2 into Equations 1 and 2 yields $\mathcal{B}_{\text{SOsimple}}$, the Scaled Output Gradient Noise Scale
 73 (SOGNS). The analogous metric using the exact per-example norm is $\mathcal{B}_{\text{PEPsimple}}$ the Per-Example
 74 Parameter Gradient Noise Scale (PEPGNS).

75 4 Experiments

76 4.1 Approximate Per-Example Gradient Noise Scale

77 We investigate how well SOGNS from Section 3.1 correlates with the observed GNS by training a
 78 1M parameter Convolutional Neural Network (CNN) on MNIST. Figure 2a shows the overall fit of
 79 SOGNS to PEPGNS at all points throughout training for only the convolutional layers (the remaining
 80 linear layers only process 2D tensors so the estimate is exact). Throughout training, the relationship
 81 between the SOGNS and PEPGNS is extremely regular over several orders of magnitude.

82 We also demonstrate the overall performance of the approximation by comparing the relationship
 83 between observed GNS and training loss. In Figure 2b, we replicate McCandlish et al. [2018] and
 84 draw $\mathcal{B}_{\text{crit}}$ as the authors measured. We see that the correlation to the critical batch size is similar for
 85 both SOGNS and PEPGNS.

(a) Scatter plot comparing the exact and approximate GNS estimators $B_{PEPsimple}$ and $B_{SOsimple}$ (b) Replication of GNS vs. loss plot from McCandlish et al. [2018], including their results for B_{crit} and both $B_{PEPsimple}$ and $B_{SOsimple}$

Figure 2: Investigation of the accuracy of the approximation from Section 3.1 on MNIST.

Figure 3: Results of a 111M parameter language model experiment measured on a fixed checkpoint. On the left, the approximate small batch gradient norm is compared to the exact and on right, the approximate SOGNS is compared to the exact PEPGNS.

86 4.2 Large Scale Gradient Noise Scale

87 To verify that this method is useful in practice, a checkpoint from a 111M parameter language
 88 model [Dey et al., 2023] was tested. In Figure 3, SOGNS and PEPGNS are compared, showing that
 89 the approximation tracks the exact case but diverges for some layers in the network. McCandlish et al.
 90 [2018] observes that the GNS may diverge by an order of magnitude from the measured “critical
 91 batch size” so the relationship we observe is within the margin of error.

92 5 Conclusion

93 Choosing a batch size is often achieved with reference to previous experiments or by hyperparameter
 94 search, which can be especially onerous in novel settings where a reasonable choice for batch size
 95 is not obvious. The GNS is a useful metric to navigate in such circumstances. In this paper, we
 96 observe that the per-example gradient norm trick [Goodfellow, 2015] could provide a useful shortcut
 97 for a minimal variance estimate of the GNS but it is inefficient in practical settings involving large
 98 transformer models [Li et al., 2022], requiring $O(T^2)$ operations in sequence length. To address
 99 this, we propose SOGNS, an approximation that operates on $O(T)$, while correlating closely with the
 100 exact GNS. As practitioners now know that it is critical to log the gradient norms during training, we
 101 hope that this work can make GNS an accessible metric for large scale experiments.

102 **References**

103 Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal,
 104 Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel
 105 Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler,
 106 Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott
 107 Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya
 108 Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

109 Denis Choquet, Pierre L'Ecuyer, and Christian Léger. Bootstrap confidence intervals for ratios of
 110 expectations ACM Trans. Model. Comput. Simul. 4(4):326–348, oct 1999. ISSN 1049-3301. doi:
 111 10.1145/352222.352224. URL <https://doi.org/10.1145/352222.352224>.

112 Nolan Dey, Gurpreet Gosal, Zhiming, Chen, Hemant Khachane, William Marshall, Ribhu Pathria,
 113 Marvin Tom, and Joel Hestness. Cerebras-GPT: Open compute-optimal language models trained
 114 on the Cerebras wafer-scale cluster, 2023.

115 Ian Goodfellow. Efficient per-example gradient computations, 2015.

116 John Graunt. Natural and Political Observations Mentioned in a following index made upon the
 117 Bills of Mortality. London: Printed by John Martyn, Printer to the Royal Society, at the Bell in St.
 118 Paul's Church-yard, 5th edition, 1676.

119 Xuechen Li, Florian Tramèr, Percy Liang, and Tatsunori Hashimoto. Large language models can be
 120 strong differentially private learners, 2022.

121 Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of
 122 large-batch training, 2018.

123 Gaspar Rochette, Andre Manoel, and Eric W. Tramel. Efficient per-example gradient computations
 124 in convolutional neural networks, 2019.

125 **A Taxonomy**

126 The following taxonomy describes the different methods available to compute GNS . Each computes
 127 $\text{J}_{\text{G}_{\text{small}}}$ in a different way:

- 128 • Microbatch: multiple G_{small} are computed over a set of microbatches
 - 129 – DDP: Each G_{small} are gradients communicated between DDP nodes [McCandlish et al.,
 130 2018]
 - 131 – Sequential: Each G_{small} are computed sequentially during gradient accumulation
- 132 • Subbatch: During gradient accumulation, select G_{small} partway through
- 133 • Per-example:
 - 134 – Exact: $\text{J}_{\text{G}_{\text{small}}}$ is computed directly the per-example gradient trick [Goodfellow,
 135 2015, Li et al., 2022]
 - 136 – Approximation: $\text{J}_{\text{G}_{\text{small}}}$ is approximated by assuming input activations are normally
 137 distributed with mean zero

138 The choice of which method to use may be dictated by the hardware available.

139 **B Variance of GNS Measurements**

140 The GNS is a ratio estimator [Graunt, 1676], it is of the form $\frac{x}{y}$, where x and y are the sample
 141 means of two random variables, in this case $\text{J}_{\text{G}_{\text{small}}}$ and S .

142 To estimate the variance of this estimator we chose a Jackknife estimator [Choquet et al., 1999],

$$\text{var}(r) = \frac{1}{n} \sum_{i=1}^n (r_i - r_J)^2;$$

143 wher \hat{r}_i is the ratio estimator computed with the sample removed and \hat{r} is the jackknife estimate
 144 of the ratio. Performing a simulation with this estimator it is possible to estimate the effect of the
 145 B_{small} and B_{big} on the variance of the estimator. These two cases are illustrated in Figures 1. We
 146 can see that the size of B_{big} is not important because the decrease in the variance as the number
 147 of samples increases is constant for B_{big} . However, the size of B_{small} is important because the
 148 variance decreases as B_{small} increases, regardless of the samples processed.

149 This reinforces the intuition that the lowest variance estimate of GNS should use the smallest
 150 B_{small} possible. The smallest choice is $B_{\text{small}} = 1$, therefore obtaining the per-example gradient
 151 norm is valuable. In the following Section 3 the per-example gradient norm trick provides this norm
 152 efficiently using gradients that are already computed in the backward pass.

153 B.1 Variance of the Gradient Noise Scale

154 The following code was used to produce Figure 1.

```

155 import numpy as np
156 import matplotlib.pyplot as plt
157 import hashlib
158
159 from dataclasses import dataclass
160
161
162 N = 1000
163 scale = 1.
164 # use explicit random state, but set it to be random by default
165 rng = np.random.RandomState(np.random.randint(1))
166 true_G = rng.randn(N)
167 true_G = np.sqrt(N) * (true_G / np.linalg.norm(true_G)) # normalise to have exactly norm N
168
169 def draw_G(B):
170     return (scale/np.sqrt(B)) * rng.randn(N) + true_G
171
172 def mean_of_microbatches(small_batch, large_batch):
173     # this is the normal setting, where you have a large batch and you split it
174     # into small batches, computing the norm of each and the norm of the whole
175     assert large_batch % small_batch == 0
176     r = large_batch // small_batch
177     G = np.array([draw_G(small_batch) for _ in range(r)])
178     return np.mean(np.linalg.norm(G, axis=1)**2), np.linalg.norm(G.mean(0))**2
179
180 funcs = {'mean_of_microbatches': mean_of_microbatches}
181
182 def jackknife(x, y):
183     n = len(x)
184     if n == 1:
185         return x[0] / y[0], np.nan
186     x, y = np.array(x), np.array(y)
187     r = np.mean(x) / np.mean(y)
188     x = x.reshape(1, 1).repeat(n, axis=1) - np.eye(n, dtype=bool)
189     y = y.reshape(1, 1).repeat(n, axis=1) - np.eye(n, dtype=bool)
190     r_i = np.mean(x, axis=0) / np.mean(y, axis=0) # vectorised jackknife
191     r_j = n * r * (((n - 1) / n) * r_i.sum())
192     # variance
193     var_r = ((n - 1) / n) * np.sum((r_i - r_j)**2)
194     return r_j, np.sqrt(var_r)
195
196 def run_replicates(large_batch, small_batch, replicates, func_type='simple_norms'):
197     for _ in range(replicates):
198         G_small, G_large = funcs[func_type](small_batch, large_batch)
199         G_est = (large_batch * G_large - small_batch * G_small) / (large_batch - small_batch)
200         S_est = (G_small - G_large) / (1./small_batch - 1./large_batch)
201         yield S_est, G_est
202
203 @dataclass
204 class Experiment:
205     samples_processed: list
206     B_est: list
207     sigmaB: list
208     S_est: list
209     G_est: list
210
211 @staticmethod
212 def mean(experiments):
213     samples_processed = experiments[0].samples_processed
214     B_est = np.mean([e.B_est for e in experiments], axis=0)
215     sigmaB = np.mean([e.sigmaB for e in experiments], axis=0)
216     S_est = np.mean([e.S_est for e in experiments], axis=0)
217     G_est = np.mean([e.G_est for e in experiments], axis=0)
218     return Experiment(samples_processed, B_est, sigmaB, S_est, G_est)
219
220 def gather_data(large_batch, small_batch):
221     S_est, G_est = [], []
222     samples_processed, B_est, sigmaB = [], [], []
223     for i, (s, g) in enumerate(run_replicates(

```

```

224         large_batch, small_batch, 100, func_type='mean_of_microbatches'
225     )):
226     S_est.append(s)
227     G_est.append(g)
228     b, sigma = jackknife(S_est, G_est)
229     samples_processed.append((i+1)large_batch)
230     B_est.append(b)
231     sigmaB.append(sigma)
232     return Experiment(samples_processed, B_est, sigmaB, S_est, G_est)
233
234 def gather_cached_data(large_batch, small_batch):
235     def generate_hash(large_batch, small_batch):
236         batch_str = str(large_batch) + "_" + str(small_batch)
237         hash_obj = hashlib.sha256(batch_str.encode())
238         small_hash = hash_obj.hexdigest()[:8]
239         return small_hash
240     # repeatedly call gather_data and cache the results to file
241     from pathlib import Path
242     import pickle
243     # check if cache_dir exists
244     cache_dir = Path('gns_var_cache')
245     cache_dir.mkdir(exist_ok=True)
246     gns_var_fpath = cache_dir / f'gns_var_cache_{generate_hash(large_batch, small_batch)}.pkl'
247     # load data if we have any
248     if gns_var_fpath.exists():
249         with open(gns_var_fpath, 'rb') as f:
250             cached_experiments = pickle.load(f)
251     else:
252         cached_experiments = []
253     # and then compute more anyway
254     experiment = gather_data(large_batch, small_batch)
255     # append this to the data we have
256     cached_experiments.append(experiment)
257     # save the data
258     with open(gns_var_fpath, 'wb') as f:
259         pickle.dump(cached_experiments, f)
260     return Experiment.mean(cached_experiments)
261
262 def plot_gns_var(large_batches, small_batches):
263     # this function can be run repeatedly to improve the estimate of the stderr
264     prop_cycle = plt.rcParams['axes.prop_cycle']
265     colors = prop_cycle.by_key()['color']
266     fig, ax1a = plt.subplots(1, 1)
267     fig.set_figheight(6)
268     ax1b = ax1a.twinx()
269     for i, (large_batch, small_batch) in enumerate(zip(large_batches, small_batches)):
270         e = gather_cached_data(large_batch, small_batch)
271         color = colors[i]
272         ax1a.plot(e.samples_processed, e.B_est,
273                 label=f'l={large_batch}, s={small_batch}', alpha=0.5, color=color)
274         ax1b.plot(e.samples_processed, e.sigmaB,
275                 label=f'l={large_batch}, s={small_batch}', alpha=0.5, color=color, linestyle='dashed')
276     ax1a.hlines(1.0, 0, e.samples_processed[1], linestyles='dashed', alpha=0.7)
277     ax1a.set_ylim(0.9, 1.1)
278     ax1a.set_xlabel('Samples processed')
279     ax1a.set_ylabel('Solid: Estimated gradient noise scale')
280     ax1b.set_ylabel('Dashed: Standard error of estimated gradient noise scale')
281     ax1a.set_xscale('log')
282     ax1a.legend()
283     plt.show()
284
285 # example usage
286 plot_gns_var([4, 8, 16, 32, 64, 128], [1] * 6)

```

287 C Efficient Per-Example Gradient Norm Notation

288 This is a description of the trick proposed by Goodfellow [2015] using Einstein and Lagrange
289 notation.

290 For the weights $W \in \mathbb{R}^{I \times K}$ of a linear layer, with input $X \in \mathbb{R}^{B \times I}$ and output $Y \in \mathbb{R}^{B \times K}$, the
291 gradient of the loss is

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Y} \frac{\partial Y}{\partial W} = X^T \frac{\partial L}{\partial Y}$$

292 which can be expressed in Einstein and Lagrange notation for a batch (left) or per-example (right) as

$$w_{ik}^0 = x_{bi} y_{bk}^0 \quad w_{bij}^0 = x_{bi} y_{bk}^0$$

293 with the squared norm in either case being

$$n^2 = (w^0)_{ik}^2 = w_{ik}^0 w_{ik}^0 \quad n_b^2 = (w^0)_{bik}^2 = w_{bik}^0 w_{bik}^0$$

294 and the per-example case factorizing as

$$n_b^2 = (w^0)_{bik}^2 = x_{bi} x_{bi} y_{bk}^0 y_{bk}^0$$

Figure 4: The naive approximation is compared to an exact computation of the per-example norm, with the ratio of the two shown on the y-axis.

So, it is sufficient to compute the squared norm of \mathbf{Y}^0 for each example to obtain exact per-example gradient norms of linear layer weights.

C.1 Per-Example Gradient Norms in 3D

For three dimensions $\mathbf{X} \in \mathbb{R}^{B \times T \times I}$ and $\mathbf{Y} \in \mathbb{R}^{B \times T \times K}$, the sums do not factorize because the per-example gradient must be reduced over the dimension:

$$w_{ij}^0 = \sum_t x_{bti} y_{btk}^0 \quad w_{bij}^0 = \sum_t x_{bti} y_{btk}^0 :$$

In this case the resulting per-example norm is [Li et al., 2022]

$$n_b^2 = (\sum_{t,i} x_{bti} y_{btk}^0)^2 = \sum_{t,i} x_{bti} y_{btk}^0 \sum_{t,i} x_{bui} y_{buk}^0 :$$

The contraction order is vital to the efficiency of this computation as

$$n_b^2 = \sum_{t,u} \sum_i x_{bti} h_{btu} \sum_k y_{btk}^0 y_{buk}^0$$

has quadratic complexity over $\sum_{t,u} i$ where T is typically sequence length in language modeling. In these cases, specifically when $n^2 > IK$ [Li et al., 2022], computing the per-example gradients explicitly before reduction is preferred:

$$n_b^2 = \sum_{i;k} \sum_t x_{bti} y_{btk}^0 \quad !^2 :$$

This operation can also be performed as a grouped convolution [Rochette et al., 2019], but the overall contractions hit the same complexity limits. In our experiments we unfold \mathbf{Y}^0 and then apply the method above when computing exact or approximate gradient norms of convolutional layers.

D Simulation Results

As discussed in Section 3.1, the approximation in Section 3.1 may either use the unit Gaussian assumption or assume the standard deviation of the activations is known; these are referred to here as the naive or relaxed assumptions, respectively. The results of a simulation are shown in Figure 5 and Figure 4 for the relaxed and naive approximations. It can be seen that the relaxed approximation is more accurate than the naive approximation.

E Approximation of Large Batch Gradient Norms

The approximation presented in Section 3.1 may be interpreted as using a scaled version of the output gradient in place of the gradient with respect to the weights, specifically we can define

$$n_b^2 = \sum_k \sum_t x_{bti} y_{btk}^0 \quad !^2 = \sum_k !_{bk}^0 \quad \text{where } !_{bk}^0 = \sum_t x_{bti} y_{btk}^0 :$$

Figure 5: The relaxed approximation is compared to an exact computation of the per-example norm, with the ratio of the two shown on the y-axis.

318 The approximation can then also be applied to compute

$$|G_{B_{\text{big}}}|^2 = \frac{\sum_k \sum_b |g_{bk}|^2}{b}$$

319 The accuracy of this approximation is illustrated in Figure 6b.

320 F MNIST Approximation Fit

321 For the remaining quantities not discussed in Section 4.1, Figure 6 describes the small batch squared
 322 gradient norm, the large batch squared gradient norm, the unbiased squared gradient norm and trace
 323 estimators of Equation 2.

