VIVIT: CURVATURE ACCESS THROUGH THE GENERALIZED GAUSS-NEWTON'S LOW-RANK STRUCTURE

SUPPLEMENTARY MATERIAL

A MATHEMATICAL DETAILS

A.1 REDUCING THE GGN EIGENVALUE PROBLEM TO THE GRAM MATRIX

For Equation (4), consider the left hand side of the GGN's characteristic polynomial $\det(\boldsymbol{G}-\lambda\boldsymbol{I}_D) = 0$. Inserting the VIVIT factorization (Equation (3)) and using the matrix determinant lemma yields

$$
\begin{aligned}
\det(-\lambda\boldsymbol{I}_D + \boldsymbol{G}) &= \det\left(-\lambda\boldsymbol{I}_D + \boldsymbol{V}\boldsymbol{V}^\top\right) && \text{(Low-rank structure (3))} \\
&= \det\left(\boldsymbol{I}_{NC} + \boldsymbol{V}^\top(-\lambda\boldsymbol{I}_D)^{-1}\boldsymbol{V}\right)\det(-\lambda\boldsymbol{I}_D) && \text{(Matrix determinant lemma)} \\
&= \det\left(\boldsymbol{I}_{NC} - \frac{1}{\lambda}\boldsymbol{V}^\top\boldsymbol{V}\right)(-\lambda)^D \\
&= \left(-\frac{1}{\lambda}\right)^{NC}\det\left(\boldsymbol{V}^\top\boldsymbol{V} - \lambda\boldsymbol{I}_{NC}\right)(-\lambda)^D \\
&= (-\lambda)^{D-NC}\det\left(\tilde{\mathbf{G}} - \lambda\boldsymbol{I}_{NC}\right). && \text{(Gram matrix)}
\end{aligned}
$$

Setting the above expression to zero reveals that the GGN's spectrum decomposes into $D - NC$ zero eigenvalues and the Gram matrix spectrum obtained from $\det(\tilde{\mathbf{G}} - \lambda\boldsymbol{I}_{NC}) = 0$.

A.2 RELATION BETWEEN GGN AND GRAM MATRIX EIGENVECTORS

Assume the nontrivial Gram matrix spectrum $\tilde{\mathbb{S}}_+ = \{(\lambda_k, \tilde{\mathbf{e}}_k) \mid \lambda_k \neq 0, \tilde{\mathbf{G}}\tilde{\mathbf{e}}_k = \lambda_k\tilde{\mathbf{e}}_k\}_{k=1}^K$ with orthonormal eigenvectors $\tilde{\mathbf{e}}_j^\top\tilde{\mathbf{e}}_k = \delta_{jk}$ (δ represents the Kronecker delta) and $K = \operatorname{rank}(\boldsymbol{G})$. We now show that $\boldsymbol{e}_k = {}^1/\sqrt{\lambda_k}\boldsymbol{V}\tilde{\mathbf{e}}_k$ are normalized eigenvectors of $\boldsymbol{G}$ and inherit orthogonality from $\tilde{\mathbf{e}}_k$.

To see the first, consider right-multiplication of the GGN with $\boldsymbol{e}_k$, then expand the low-rank structure,

$$
\begin{aligned}
\boldsymbol{G}\boldsymbol{e}_k &= \frac{1}{\sqrt{\lambda_k}} \boldsymbol{V}\boldsymbol{V}^\top \boldsymbol{V} \tilde{\mathbf{e}}_k && \text{(Equation (3) and definition of } \boldsymbol{e}_k\text{)} \\
&= \frac{1}{\sqrt{\lambda_k}} \boldsymbol{V} \tilde{\mathbf{G}} \tilde{\mathbf{e}}_k && \text{(Gram matrix)} \\
&= \lambda_k \frac{1}{\sqrt{\lambda_k}} \boldsymbol{V} \tilde{\mathbf{e}}_k && \text{(Eigenvector property of } \tilde{\mathbf{e}}_k\text{)} \\
&= \lambda_k \boldsymbol{e}_k \,.
\end{aligned}
$$

Orthonormality of the $\boldsymbol{e}_k$ results from the Gram matrix eigenvector orthonormality,

$$
\begin{aligned}
\boldsymbol{e}_j^\top \boldsymbol{e}_k &= \left(\frac{1}{\sqrt{\lambda_j}} \tilde{\mathbf{e}}_j^\top \boldsymbol{V}^\top \right) \left(\frac{1}{\sqrt{\lambda_k}} \boldsymbol{V} \tilde{\mathbf{e}}_k \right) && \text{(Definition of } \boldsymbol{e}_j, \boldsymbol{e}_k\text{)} \\
&= \frac{1}{\sqrt{\lambda_j \lambda_k}} \tilde{\mathbf{e}}_j^\top \tilde{\mathbf{G}} \tilde{\mathbf{e}}_k && \text{(Gram matrix)} \\
&= \frac{\lambda_k}{\sqrt{\lambda_j \lambda_k}} \tilde{\mathbf{e}}_j^\top \tilde{\mathbf{e}}_k && \text{(Eigenvector property of } \tilde{\mathbf{e}}_k\text{)} \\
&= \delta_{jk} \,. && \text{(Orthonormality)}
\end{aligned}
$$

B Experimental details

B.1 Performance evaluation

Hardware information: Results presented in this section, as well as Section 2, were generated on a workstation with the following hardware:

- **CPU:** Intel Core i7-8700K CPU @ 3.70 GHz × 12 (32 GB)
- **GPU:** NVIDIA GeForce RTX 2080 Ti (11 GB)

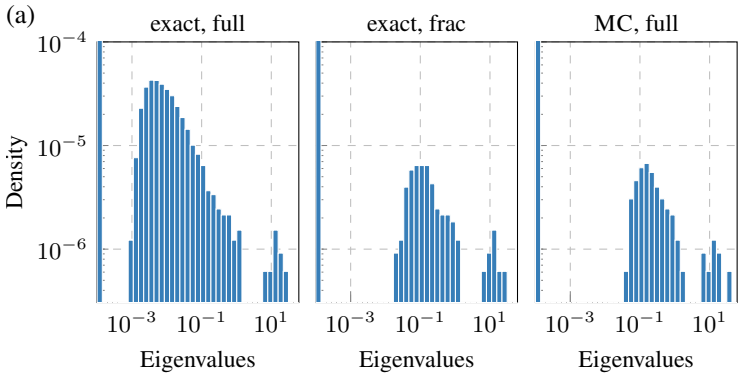
We will use their shorthands to indicate the device that executed the computation.

Settings: Performance is evaluated with different GGN approximations, parameterized by the used mini-batch samples (full, frac), and the backpropagated loss Hessian representation (exact, MC):

- **exact, full:** Backpropagate the exact loss Hessian representation for all mini-batch samples (NC vectors).
- **MC, full:** Backpropagate an MC approximation of the loss Hessian (using a single MC sample) for all mini-batch samples (N vectors).
- **exact, frac:** Backpropagate the exact loss Hessian representation for a fraction ($1/8$, as in Zhang et al. (2017)) of mini-batch samples ($N/8C$ vectors).
- **exact, MC:** Backpropagate an MC approximation of the loss Hessian (using a single MC sample) for a fraction ($1/8$, as in Zhang et al. (2017)) of mini-batch samples ($N/8$ vectors).

In addition to computing the target quantity (GGN spectrum, damped Newton step) with BackPACK, a standard gradient backpropagation on the full mini-batch is always performed in PyTorch's backward pass. Performance is evaluated on convolutional neural nets from DeepOBS (Schneider et al., 2019): 3c3d on CIFAR-10, 2c2d on Fashion-MNIST, and All-CNN-C on CIFAR-100.

GGN spectra: To obtain the spectra of Figure 1, Figure S.4, and Figure S.5, we initialize the respective architecture, then draw a mini-batch and evaluate the GGN eigenvalues under the described approximations, clipping the Gram matrix eigenvalues at 10^{-4}. Mini-batch sizes correspond to the default value for training where possible (CIFAR-10 3c3d: $N = 128$, Fashion-MNIST 2c2d: $N = 128$). Only on CIFAR-100 All-CNN-C (trained with $N = 256$), we reduce the batch size to $N = 64$ to fit the exact computation on the full mini-batch, used as baseline, into memory.



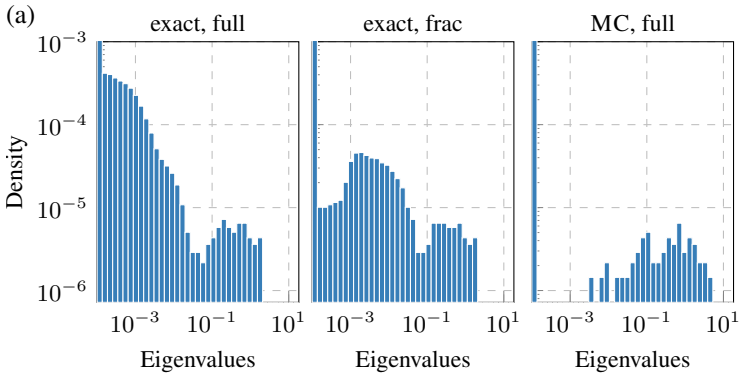
(b) N_{crit} (eigenvalues)

$_{\text{GGN}}^{\text{Batch}}$	full	frac
exact	76	207
MC	718	1740

(c) N_{crit} (block Newton)

$_{\text{GGN}}^{\text{Batch}}$	full	frac
exact	68	159
MC	368	528

Figure S.4: **Scalability evaluation on FASHION-MNIST 2C2D ($D = 3{,}274{,}634$, $C = 10$).** (a) GGN eigenvalue distribution with different costs on a mini-batch of size $N = 128$. From left to right: Exact GGN on the full batch, exact GGN on a batch fraction ($1/8$, as in Zhang et al. (2017)), MC-approximation of the GGN on the full batch. (b) Maximum batch size N_{crit} (GPU) for a standard gradient computation and the GGN spectrum and (c) computing exact Newton steps with layer-wise parameter groups.



(b) N_{crit} (eigenvalues)

$_{\text{GGN}}^{\text{Batch}}$	full	frac
exact	35	140
MC	1280	1952

(c) N_{crit} (block Newton)

$_{\text{GGN}}^{\text{Batch}}$	full	frac
exact	35	135
MC	1079	1536

Figure S.5: **Scalability evaluation on CIFAR-100 ALL-CNN-C ($D = 1{,}387{,}108, C = 100$).** (a) GGN eigenvalue distribution with different costs on a mini-batch of size $N = 64$. From left to right: Exact GGN on the full batch, exact GGN on a batch fraction ($1/8$, as in Zhang et al. (2017)), MC-approximation of the GGN on the full batch. (b) Maximum batch size N_{crit} (GPU) for a standard gradient computation and the GGN spectrum and (c) computing exact Newton steps with layer-wise parameter groups.

Critical batch sizes: Similar to the GGN spectra, we repeat their computation and vary the mini-batch size until the device runs out of memory. The largest mini-batch size that can be handled is denoted as N_{crit}, the critical batch size. Figure 1b, Figure S.4b, Figure S.5b, and Table S.1a present additional results.

The critical batch sizes in Figure 1c, Figure S.4c, Figure S.5b, and Table S.1b, employ a block-diagonal GGN approximation with groups consisting of weights and bias terms in each layer (see Appendix C). For each block we compute a damped Newton step (first term in Equation (9), using Gram matrix eigenvalues larger than 10^{-4}) with constant damping $\delta = 1$.

As explained in Section 2, the GGN eigenvalues only require the Gram matrix. Newton steps from a block-diagonal approximation additionally require the directional derivatives (Equation (8)), which involve individual gradients. Second-order directional derivatives λ_{nk} (Equation (8b)) are evaluated on the same samples as the GGN eigenvectors, but we *always* use all mini-batch samples to compute the directional gradients γ_{nk} (Equation (8a)). As gradients are cheaper to compute, this suggests

Table S.1: **Critical batch sizes for eigenvalues and Newton steps with different approximations.** Additional results that complement Figure 1b,c, Figure S.4b,c, and Figure S.5 are shown column-wise for each architecture. From top to bottom, we report the critical batch sizes for computing (a) the GGN eigenvalue spectrum on CPU, (b) damped Newton steps with a block-diagonal GGN approximation corresponding to individual layers on CPU, and (c,d) damped Newton steps with the full GGN matrix on CPU and GPU. Interpretations and procedure details are provided in the text.

	CIFAR-10 3C3D		**FASHION-MNIST 2C2D**		**CIFAR-100 ALL-CNN-C**	
(a)	N_{crit} (eigenvalues, CPU)		N_{crit} (eigenvalues, CPU)		N_{crit} (eigenvalues, CPU)	
$\text{GGN}^{\text{Batch}}$	full	frac	full	frac	full	frac
exact	1166	3004	231	615	94	519
MC	8430	14908	2208	5495	3984	5859
(b)	N_{crit} (block Newton, CPU)		N_{crit} (block Newton, CPU)		N_{crit} (block Newton, CPU)	
$\text{GGN}^{\text{Batch}}$	full	frac	full	frac	full	frac
exact	1046	2423	210	487	95	504
MC	4997	6838	1137	1643	3360	3920
(c)	N_{crit} (full Newton, CPU)		N_{crit} (full Newton, CPU)		N_{crit} (full Newton, CPU)	
$\text{GGN}^{\text{Batch}}$	full	frac	full	frac	full	frac
exact	667	2215	202	487	43	309
MC	3473	5632	1107	1639	2015	2865
(d)	N_{crit} (full Newton, GPU)		N_{crit} (full Newton, GPU)		N_{crit} (full Newton, GPU)	
$\text{GGN}^{\text{Batch}}$	full	frac	full	frac	full	frac
exact	208	727	66	159	13	87
MC	1055	1816	362	528	640	959

evaluating them on more samples compared to curvature, see e.g. Zhang et al. (2017). The overhead thus leads to smaller critical batch sizes in comparison to computing the GGN spectrum.

For completeness, Table S.1c,d shows critical batch sizes when the block-diagonal GGN is replaced by its full representation (full Newton). In contrast to Newton steps with a block-diagonal matrix that can discard the stage-wise matrices $\boldsymbol{V}^{(i)}$ during backpropagation, $\boldsymbol{V}$ must now be stored until all parameters have been traversed. This leads to higher memory consumption, and hence smaller critical batch sizes, but avoids multiple Gram matrix inversions (one per parameter group).

In summary, we find that there always exists a combination of approximations which allows for critical batch sizes larger than the traditional size used for training (some architectures even permit exact computation). Different accuracy-cost trade-offs may be preferred, depending on the application and the computational budget. By the presented approximations, VIVIT's representation is capable to adapt over a wide range.

B.2 NOISE ANALYSIS DURING TRAINING

Procedure: We train the following DEEPOBS (Schneider et al., 2019) architectures with SGD and ADAM: 3C3D on CIFAR-10 ($N = 128$), 2C2D on FASHION-MNIST ($N = 128$), and ALL-CNN-C on CIFAR-100 ($N = 256$). To assert successful training, we use the hyperparameters from Dangel

Table S.2: **Hyperparameter settings for training runs to analyze noise.** For both SGD and ADAM, we report their learning rates α (taken from the baselines in Dangel et al. (2020)) and link to their visualization. Momentum for SGD was fixed to 0.9, and ADAM uses the default parameters $(\beta_1, \beta_2) = (0.99, 0.999)$.

Problem	SGD	ADAM
CIFAR-10 3C3D	$\alpha \approx 3.79 \cdot 10^{-3}$ (Figure 2)	$\alpha \approx 2.98 \cdot 10^{-4}$ (Figure S.6)
FASHION-MNIST 2C2D	$\alpha \approx 2.07 \cdot 10^{-2}$ (Figure S.7)	$\alpha \approx 1.27 \cdot 10^{-4}$ (Figure S.8)
CIFAR-100 ALL-CNN-C	$\alpha \approx 4.83 \cdot 10^{-1}$ (Figure S.9)	$\alpha \approx 6.95 \cdot 10^{-4}$ (Figure S.10)

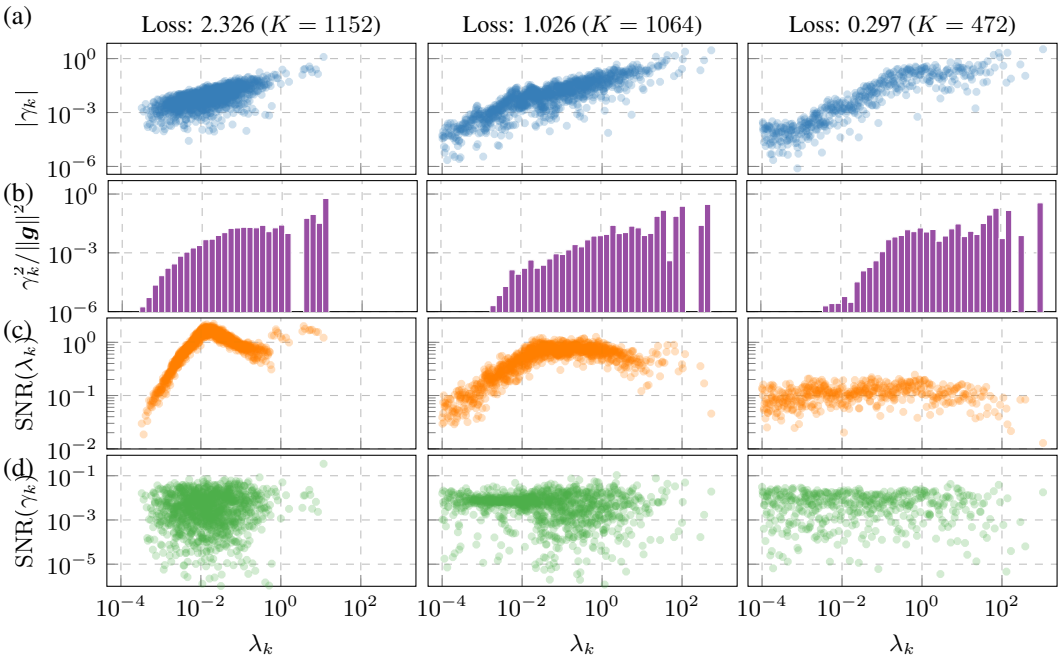


Figure S.6: **Gradient, curvature and noise during training of CIFAR-10 3C3D with ADAM.** Individual columns show the architecture's state at initialization (*left*), an early (epoch 5, *center*), and advanced (epoch 68, *right*) stage of training. For each direction k, characterized by its curvature λ_k, we monitor (a) the directional gradient magnitude; (b) gradient-eigenvector alignment; (c,d) signal-to-noise ratios of curvatures and gradients.

et al. (2020) (see Table S.2), but turn off regularization as it would alter gradients and curvature, and their respective noise.

Metrics are computed on a single held-out mini-batch during training, using the same batch size if possible (3C3D on CIFAR-10: $N = 128$, 2C2D FASHION-MNIST: $N = 128$), or a smaller value to fit the computation into memory (ALL-CNN-C on CIFAR-100: $N = 64$). We focus on the exact GGN without further approximations and use the full mini-batch for the directional derivatives.

Signal-to-noise ratios: From the empirical mini-batch distributions $\{\gamma_{nk}\}, \{\lambda_{nk}\}$ we compute

$$\mathrm{SNR}(\chi_k) = \frac{\mathbb{E}\left[\chi_k\right]^2}{\mathrm{Var}\left[\chi_k\right]} = \frac{\mathbb{E}\left[\chi_k\right]^2}{\mathbb{E}\left[\chi_k^2\right] - \mathbb{E}\left[\chi_k\right]^2} \quad \text{with} \quad \mathbb{E}\left[\chi_k\right] = \frac{1}{N}\sum_{n=1}^{N} \chi_{nk} \quad \text{for} \quad \chi \in \{\lambda, \gamma\}\,.$$

Summary: All analyzed runs exhibit similar behaviors as described in Section 4.1.

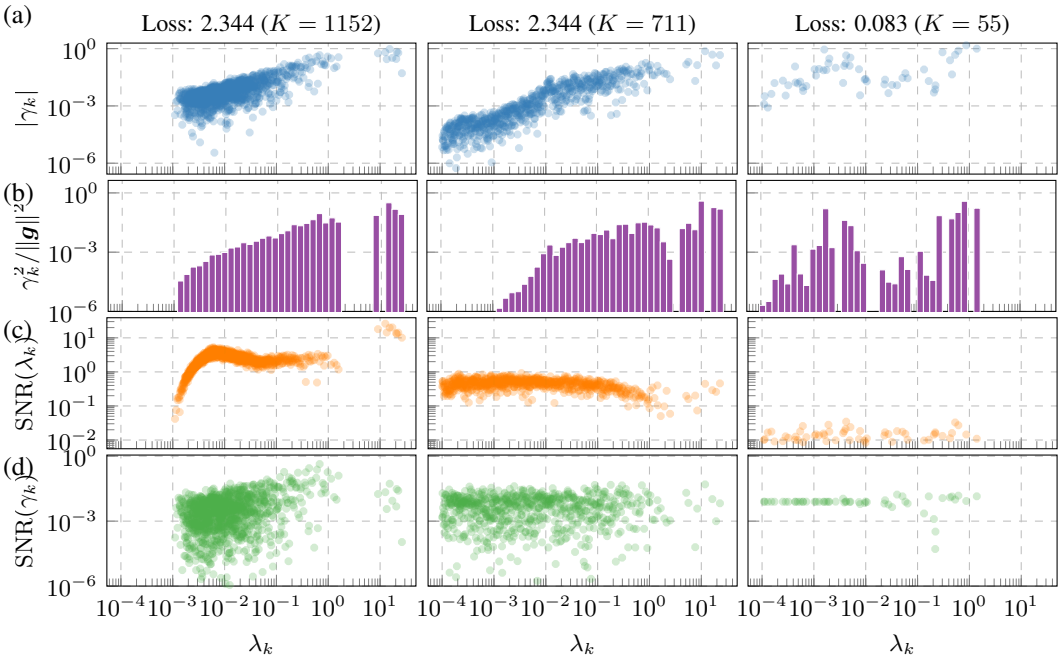


Figure S.7: **Gradient, curvature and noise during training of FASHION-MNIST 2C2D with SGD.** Individual columns show the architecture's state at initialization (*left*), an early (after 100 steps, *center*), and advanced (epoch 57, *right*) stage of training. For each direction k, characterized by its curvature λ_k, we monitor (a) the directional gradient magnitude; (b) gradient-eigenvector alignment; (c,d) signal-to-noise ratios of curvatures and gradients.

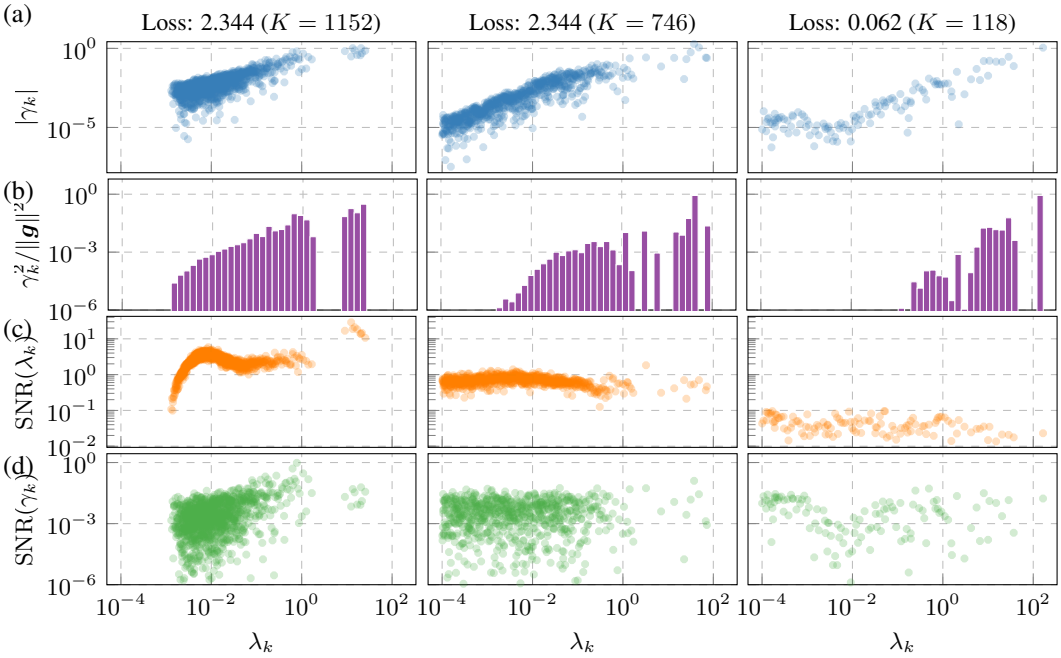


Figure S.8: **Gradient, curvature and noise during training of FASHION-MNIST 2C2D with ADAM.** Individual columns show the architecture's state at initialization (*left*), an early (after 100 steps, *center*), and advanced (epoch 57, *right*) stage of training. For each direction k, characterized by its curvature λ_k, we monitor (a) the directional gradient magnitude; (b) gradient-eigenvector alignment; (c,d) signal-to-noise ratios of curvatures and gradients.

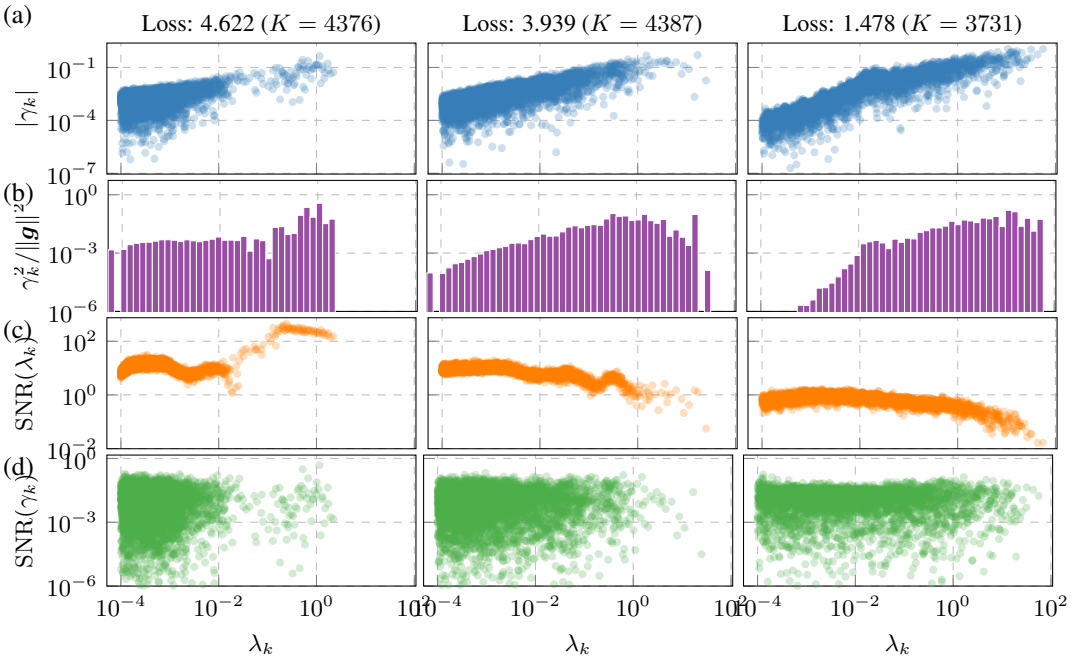


Figure S.9: **Gradient, curvature and noise during training of CIFAR-100 ALL-CNN-C with SGD.** Individual columns show the architecture's state at initialization (*left*), an early (epoch 5, *center*), and advanced (epoch 311, *right*) stage of training. For each direction k, characterized by its curvature λ_k, we monitor (a) the directional gradient magnitude; (b) gradient-eigenvector alignment; (c,d) signal-to-noise ratios of curvatures and gradients.

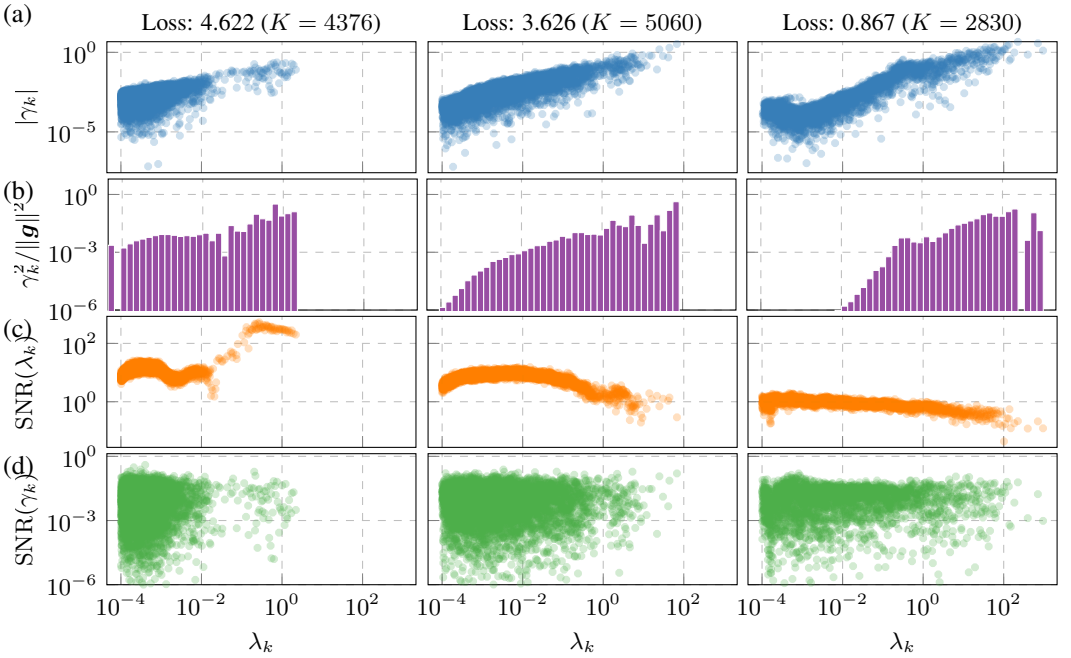


Figure S.10: **Gradient, curvature and noise during training of CIFAR-100 ALL-CNN-C with ADAM.** Individual columns show the architecture's state at initialization (*left*), an early (epoch 5, *center*), and advanced (epoch 311, *right*) stage of training. For each direction k, characterized by its curvature λ_k, we monitor (a) the directional gradient magnitude; (b) gradient-eigenvector alignment; (c,d) signal-to-noise ratios of curvatures and gradients.

B.3 BOOTSTRAP DAMPING FOR SECOND-ORDER METHODS

Bootstrap damping: The starting point for our directional bootstrap damping is Equation (10),

$$\mathcal{R}(\delta_k) = q(\boldsymbol{\theta}) - q(\boldsymbol{\theta} + s_k \boldsymbol{e}_k) = -s_k \left(\frac{1}{N}\sum_{n=1}^{N} \gamma_{nk}\right) - \frac{1}{2} s_k^2 \left(\frac{1}{N}\sum_{n=1}^{N} \lambda_{nk}\right).$$

It describes the reduction of the quadratic mini-batch model q when taking a step $s_k = -\frac{\gamma_k}{\lambda_k + \delta_k} \in \mathbb{R}$ in direction $\boldsymbol{e}_k$. Since our ultimate goal is to minimize the training loss, we would like to choose a damping such that its corresponding update not only reduces q, but *consistently* decreases the loss over all other mini-batch models as well.

One way to assess the step s_k in this regard is the non-parametric bootstrap (Efron, 1979). The general idea is that we can simulate additional samples for the derivatives in direction $\boldsymbol{e}_k$ on other batches by resampling from the data $\{\gamma_{nk}\}_{n=1}^N$, $\{\lambda_{nk}\}_{n=1}^N$. First, we draw N indices $i_1, ..., i_N \in \{1, ..., N\}$ with replacement. By taking the mean of $\gamma_{i_1 k}, ..., \gamma_{i_N k}$, we can simulate the first directional derivative of an alternative, equally valid quadratic model (and similarly for the second directional derivative by taking the mean of $\lambda_{i_1 k}, ..., \lambda_{i_N k}$). Replacing the directional derivatives $1/N \sum_{n=1}^N \gamma_{nk}$ and $1/N \sum_{n=1}^N \lambda_{nk}$ in the equation above by these new averages $1/N \sum_{j=1}^N \gamma_{i_j k}$ and $1/N \sum_{j=1}^N \lambda_{i_j k}$ yields a new sample for $\mathcal{R}_k(\delta_k)$. This allows to create an arbitrary number of samples for any δ_k.

For a given damping δ_k, such samples indicate what reduction in training loss to expect with the respective update s_k. Taking the 5% percentile of the bootstrap-generated samples, we obtain a *confident* lower bound to this reduction. We then choose the δ_k that maximizes this lower bound from candidates on a discrete grid. This approach is repeated for all non-trivial directions.

B.4 DERIVATION OF NOISY QUADRATIC MODEL

Here, we reverse-engineer a noisy quadratic model used by the optimizer to minimize an inaccessible objective function such that we have full control over the directional noisy first- and second-order derivatives observed by the optimizer through automatic differentiation (directions themselves are not subject to noise). Not only do we formulate the optimization problem mathematically, but also derive an equivalent BACKPACK-compatible neural network training procedure, such that we can use VIVIT to compute directional derivatives during a backward pass.

One-dimensional case: Consider a one-dimensional objective function $f : \mathbb{R} \to \mathbb{R}, \vartheta \mapsto f(\vartheta)$ which we want to optimize,

$$\min_{\vartheta} f(\vartheta).$$

At the current iterate $\vartheta_0 \in \mathbb{R}$, an optimizer constructs a local model of f to update its solution.

We choose the family of local models to be convex quadratics. Let $\varphi_{\text{true}} : \mathbb{R} \to \mathbb{R}, \vartheta \mapsto \varphi_{\text{true}}(\vartheta)$ denote one instance of a model, defined by its local curvature $\lambda_{\text{true}}(\vartheta_0) \in \mathbb{R}_+$, gradient $\gamma_{\text{true}}(\vartheta_0) \in \mathbb{R}$, and function value $\varphi_{\text{true}}(\vartheta_0) \in \mathbb{R}$,

$$\varphi_{\text{true}}(\vartheta) = \frac{1}{2}\lambda_{\text{true}}(\vartheta_0)(\vartheta - \vartheta_0)^2 + \gamma_{\text{true}}(\vartheta_0)(\vartheta - \vartheta_0) + \varphi_{\text{true}}(\vartheta_0). \tag{S.11}$$

By construction, $\nabla_\vartheta \varphi_{\text{true}}(\vartheta_0) = \gamma_{\text{true}}(\vartheta_0)$ and $\nabla_\vartheta^2 \varphi_{\text{true}}(\vartheta_0) = \lambda_{\text{true}}(\vartheta_0)$. For a 'good' local description of f, those values should be representative of $\nabla_\vartheta f(\vartheta_0), \nabla_\vartheta^2 f(\vartheta_0)$, but may sometimes deviate to guarantee model properties such as convexity.

Both the objective f and its local model φ_{true}, assumed by the optimizer, are inaccessible in practice. A noisy version $\varphi : \mathbb{R} \to \mathbb{R}, \vartheta \mapsto \varphi(\vartheta)$ of Equation (S.11) substitutes true curvature ($\lambda_{\text{true}}(\vartheta_0) \to \lambda(\vartheta_0)$) and gradient ($\gamma_{\text{true}}(\vartheta_0) \to \gamma(\vartheta_0)$) with random variables (the offset $\varphi_{\text{true}}(\vartheta_0)$ is not crucial for optimization, and will thus not be perturbed with noise in this presentation),

$$\varphi(\vartheta) = \frac{1}{2}\lambda(\vartheta_0)(\vartheta - \vartheta_0)^2 + \gamma(\vartheta_0)(\vartheta - \vartheta_0) + \varphi_{\text{true}}(\vartheta_0). \tag{S.12}$$

To observe $\varphi(\vartheta)$ multiple times, Equation (S.12) is evaluated on samples $\{(\lambda_n(\vartheta_0), \gamma_n(\vartheta_0)) \in \mathbb{R}_+ \times \mathbb{R}\}_{n=1}^N$ drawn from the joint distribution of $(\lambda(\vartheta_0), \gamma(\vartheta_0))$. Neglecting the offset, sample n gives rise to

$$\varphi_n(\vartheta) = \frac{1}{2}\lambda_n(\vartheta_0)(\vartheta - \vartheta_0)^2 + \gamma_n(\vartheta_0)(\vartheta - \vartheta_0). \tag{S.13a}$$

It is common to batch-process multiple samples, compute the average

$$\overline{\varphi}(\vartheta) = \frac{1}{N}\sum_{n=1}^{N} \varphi_n(\vartheta)\,, \tag{S.13b}$$

and use automatic differentiation to compute the quantities employed by an optimizer. Recap that Equation (S.13b) produces the correct noisy first- and second-order partial derivatives, i.e. $\nabla_\vartheta \varphi_n(\vartheta_0) = \gamma_n(\vartheta_0)$ and $\nabla^2_\vartheta \varphi_n(\vartheta_0) = \lambda_n(\vartheta_0)$. Because our work not only relies on automatic differentiation, but computes directional derivatives through VIVIT, we translate Equation (S.13b) into the training procedure of a sequential neural network with specifically engineered labeled data.

BACKPACK-compatible one-dimensional formulation: To reformulate Equation (S.13) as train loss of a sequential neural net, we complete the square in Equation (S.12),

$$\begin{aligned}
\varphi(\vartheta) &= \frac{1}{2}\lambda(\vartheta_0)\left(\vartheta - \vartheta_0 + \frac{\gamma(\vartheta_0)}{\lambda(\vartheta_0)}\right)^2 - \frac{1}{2}\frac{\gamma(\vartheta_0)^2}{\lambda(\vartheta_0)} + \varphi_{\text{true}}(\vartheta_0) \\
&= \left(\sqrt{\frac{\lambda(\vartheta_0)}{2}}(\vartheta - \vartheta_0) + \frac{\gamma(\vartheta_0)}{\sqrt{2\lambda(\vartheta_0)}}\right)^2 - \frac{1}{2}\frac{\gamma(\vartheta_0)^2}{\lambda(\vartheta_0)} + \varphi_{\text{true}}(\vartheta_0) \\
&= \left(\sqrt{\frac{\lambda(\vartheta_0)}{2}}\vartheta - \left(\sqrt{\frac{\lambda(\vartheta_0)}{2}}\vartheta_0 - \frac{\gamma(\vartheta_0)}{\sqrt{2\lambda(\vartheta_0)}}\right)\right)^2 - \frac{1}{2}\frac{\gamma(\vartheta_0)^2}{\lambda(\vartheta_0)} + \varphi_{\text{true}}(\vartheta_0) \\
&= (x\vartheta - y)^2 + \text{const.}
\end{aligned} \tag{S.14a}$$

with

$$x = \sqrt{\frac{\lambda(\vartheta_0)}{2}} \in \mathbb{R}\,, \tag{S.14b}$$

$$y = \sqrt{\frac{\lambda(\vartheta_0)}{2}}\vartheta_0 - \frac{\gamma(\vartheta_0)}{\sqrt{2\lambda(\vartheta_0)}} \in \mathbb{R}\,, \tag{S.14c}$$

$$\text{const.} = -\frac{1}{2}\frac{\gamma(\vartheta_0)^2}{\lambda(\vartheta_0)} + \varphi_{\text{true}}(\vartheta_0) \in \mathbb{R}\,. \tag{S.14d}$$

Note that the data, in which the noisy observations are embedded, must depend on the current location ϑ_0. Such a dependence is not common in practical tasks. But it is a consequence of the design, because we want to achieve full control of gradient and curvature noise that the optimizer is exposed to at any time during optimization.

With that, we conclude that minimizing Equation (S.12) through noisy observations of the form Equation (S.13) is equivalent to the following neural network training: Assume the optimizer's current iterate to be ϑ_0. Then, the following steps define a training iteration:

1. Generate data
   (a) Draw curvature and gradient samples $\{(\lambda_n(\vartheta_0), \gamma_n(\vartheta_0)\}_{n=1}^N$
   (b) Compute inputs $\{x_n = \sqrt{\lambda_n(\vartheta_0)/2}\}_{n=1}^N$ and labels $\{y_n = \sqrt{\lambda_n(\vartheta_0)/2}\vartheta_0 - \gamma_n(\vartheta_0)/\sqrt{2\lambda_n(\vartheta_0)}\}_{n=1}^N\}$
2. Forward pass
   (a) Feed $\{x_n\}_{n=1}^N$ through a linear layer ($\mathbb{R} \to \mathbb{R}$, no bias) with trainable weight $\vartheta \in \mathbb{R}$, set to ϑ_0
   (b) Feed the output $\{x_n\vartheta\}_{n=1}^N$ through the mean-squared error (MSE) with labels $\{y_n\}_{n=1}^N$,
   $$\text{MSE}(\{x_n\vartheta\}, \{y_n\}) = \frac{1}{N}\sum_{n=1}^{N}(x_n\vartheta - y_n)^2 = \overline{\varphi}(\vartheta)$$
3. Backward pass: Compute the first- and second-order directional derivatives $\{\gamma_n(\vartheta_0)\}_{n=1}^N$ and $\{\lambda_n(\vartheta_0)\}_{n=1}^N$ with VIVIT during a backward pass. By construction, this reproduces the sampled gradients and curvatures from Item 1a
4. Optimizer step: Update the value of ϑ using $\{\gamma_n(\vartheta_0)\}_{n=1}^N$ and $\{\lambda_n(\vartheta_0)\}_{n=1}^N$, set ϑ_0 to ϑ's new value

Multi-dimensional case: Next, we extend the one-dimensional case to multiple dimensions. Consider a multi-dimensional objective function $F : \mathbb{R}^D \to \mathbb{R}, \boldsymbol{\theta} \mapsto F(\boldsymbol{\theta})$ which we want to optimize,

$$\min_{\boldsymbol{\theta}} F(\boldsymbol{\theta}) \,.$$

The local model of F at $\boldsymbol{\theta}_0 \in \mathbb{R}^D$, employed by the optimizer, is a convex D-dimensional quadratic. Let $\phi_{\text{true}} : \mathbb{R}^D \to \mathbb{R}, \boldsymbol{\theta} \mapsto \phi_{\text{true}}(\boldsymbol{\theta})$ denote one instance of a model, defined by its local Hessian spectrum $\{(\lambda_{d,\text{true}}(\boldsymbol{\theta}_0), \boldsymbol{e}_d(\boldsymbol{\theta}_0))\}_{d=1}^D$ with local curvatures $\{\lambda_{d,\text{true}}(\boldsymbol{\theta}_0) \in \mathbb{R}_+\}_{d=1}^D$ such that the Hessian is $\nabla^2_{\boldsymbol{\theta}}\phi(\boldsymbol{\theta}_0) = \sum_{d=1}^D \lambda_{d,\text{true}}(\boldsymbol{\theta}_0)\boldsymbol{e}_d(\boldsymbol{\theta}_0)\boldsymbol{e}_d(\boldsymbol{\theta}_0)^\top$. It also requires local gradients $\{\gamma_{d,\text{true}}(\boldsymbol{\theta}_0) \in \mathbb{R}\}_{d=1}^D$ along the directions $\{\boldsymbol{e}_d(\boldsymbol{\theta}_0) \in \mathbb{R}^D\}_{d=1}^D$, such that $\nabla_{\boldsymbol{\theta}}\phi(\boldsymbol{\theta}_0) = \sum_{d=1}^D \gamma_{d,\text{true}}(\boldsymbol{\theta}_0)\boldsymbol{e}_d(\boldsymbol{\theta}_0)$, and the local function value at $\boldsymbol{\theta}_0 \in \mathbb{R}^D$,

$$\begin{aligned}\phi_{\text{true}}(\boldsymbol{\theta}) &= \frac{1}{2}(\boldsymbol{\theta}-\boldsymbol{\theta}_0)^\top \nabla^2_{\boldsymbol{\theta}}\phi(\boldsymbol{\theta}_0)(\boldsymbol{\theta}-\boldsymbol{\theta}_0) + (\boldsymbol{\theta}-\boldsymbol{\theta}_0)^\top \nabla_{\boldsymbol{\theta}}\phi(\boldsymbol{\theta}_0) + \phi_{\text{true}}(\boldsymbol{\theta}_0) \\ &= \left[\sum_{d=1}^D \frac{1}{2}\lambda_{d,\text{true}}(\boldsymbol{\theta}_0)\left(\boldsymbol{e}_d(\boldsymbol{\theta}_0)^\top(\boldsymbol{\theta}-\boldsymbol{\theta}_0)\right)^2 + \gamma_{d,\text{true}}(\boldsymbol{\theta}_0)\boldsymbol{e}_d(\boldsymbol{\theta}_0)^\top(\boldsymbol{\theta}-\boldsymbol{\theta}_0)\right] + \phi_{\text{true}}(\boldsymbol{\theta}_0)\,.\end{aligned} \tag{S.15}$$

Note that expanding $\boldsymbol{\theta} = \sum_{d=1}^D \vartheta_d \boldsymbol{e}_d(\boldsymbol{\theta}_0)$ and $\boldsymbol{\theta}_0 = \sum_{d=1}^D \vartheta_{d,0}\boldsymbol{e}_d(\boldsymbol{\theta}_0)$ in terms of scalar coordinates $\vartheta_d = \boldsymbol{e}_d(\boldsymbol{\theta}_0)^\top\boldsymbol{\theta}$ and $\vartheta_{d,0} = \boldsymbol{e}_d(\boldsymbol{\theta}_0)^\top\boldsymbol{\theta}_0$ decouples Equation (S.15) into one-dimensional quadratics,

$$\begin{aligned}\phi_{\text{true}}(\boldsymbol{\theta}) &= \left[\sum_{d=1}^D \frac{1}{2}\lambda_{d,\text{true}}(\boldsymbol{\theta}_0)\left(\vartheta_d - \vartheta_{d,0}\right)^2 + \gamma_{d,\text{true}}(\boldsymbol{\theta}_0)\left(\vartheta_d - \vartheta_{d,0}\right)\right] + \phi_{\text{true}}(\boldsymbol{\theta}_0) \\ &= \sum_{d=1}^D \varphi_{d,\text{true}}(\vartheta_d) + \phi_{\text{true}}(\boldsymbol{\theta}_0)\,,\end{aligned} \tag{S.16}$$

which is (up to constant offsets that are negligible for optimization) the sum of D one-dimensional quadratic functions of the form Equation (S.11), with $\varphi_{d,\text{true}}$ defined by $(\lambda_{d,\text{true}}(\boldsymbol{\theta}_0), \gamma_{d,\text{true}}(\boldsymbol{\theta}_0), \vartheta_{d,0})$ along each direction.

A noisy version $\phi : \mathbb{R}^D \to \mathbb{R}, \boldsymbol{\theta} \mapsto \phi(\boldsymbol{\theta})$ of Equation (S.15) replaces the true curvatures ($\{\lambda_{d,\text{true}}(\boldsymbol{\theta}_0)\}_{d=1}^D \to \{\lambda_d(\boldsymbol{\theta}_0)\}_{d=1}^D$) and local gradients ($\{\gamma_{d,\text{true}}(\boldsymbol{\theta}_0)\}_{d=1}^D \to \{\gamma_d(\boldsymbol{\theta}_0)\}_{d=1}^D$) at $\boldsymbol{\theta}_0$ with random variables (the offset $\phi_{\text{true}}(\boldsymbol{\theta}_0)$ is not crucial for optimization, and will thus not be perturbed),

$$\phi(\boldsymbol{\theta}) = \left[\sum_{d=1}^D \frac{1}{2}\lambda_d(\boldsymbol{\theta}_0)\left(\boldsymbol{e}_d(\boldsymbol{\theta}_0)^\top(\boldsymbol{\theta}-\boldsymbol{\theta}_0)\right)^2 + \gamma_d(\boldsymbol{\theta}_0)\boldsymbol{e}_d(\boldsymbol{\theta}_0)^\top(\boldsymbol{\theta}-\boldsymbol{\theta}_0)\right] + \phi_{\text{true}}(\boldsymbol{\theta}_0)\,. \tag{S.17}$$

The optimizer observes $\phi(\boldsymbol{\theta})$ through N samples $\{(\lambda_{nd}(\boldsymbol{\theta}_0), \gamma_{nd}(\boldsymbol{\theta}_0)) \in \mathbb{R}_+ \times \mathbb{R}\}_{n=1,d=1}^{N,D}$ drawn from the joint distribution of $(\lambda_d(\boldsymbol{\theta}_0), \gamma_d(\boldsymbol{\theta}_0))$. Neglecting the offset, sample $\{(\lambda_{nd}(\boldsymbol{\theta}_0), \gamma_{nd}(\boldsymbol{\theta}_0))\}_{d=1}^D$ gives rise to

$$\phi_n(\boldsymbol{\theta}) = \sum_{d=1}^D \frac{1}{2}\lambda_{nd}(\boldsymbol{\theta}_0)\left(\boldsymbol{e}_d(\boldsymbol{\theta}_0)^\top(\boldsymbol{\theta}-\boldsymbol{\theta}_0)\right)^2 + \gamma_{nd}(\boldsymbol{\theta}_0)\boldsymbol{e}_d(\boldsymbol{\theta}_0)^\top(\boldsymbol{\theta}-\boldsymbol{\theta}_0)\,. \tag{S.18a}$$

It is common to batch-process multiple samples and compute the average

$$\bar{\phi}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^N \phi_n(\boldsymbol{\theta}), \tag{S.18b}$$

which will give the correct first- and second-order directional derivatives in an automatic differentiation engine, i.e. $\boldsymbol{e}_d(\boldsymbol{\theta}_0)^\top\nabla_{\boldsymbol{\theta}}\phi_n(\boldsymbol{\theta}_0) = \gamma_{nd}(\boldsymbol{\theta}_0)$ and $\boldsymbol{e}_d(\boldsymbol{\theta}_0)^\top\nabla^2_{\boldsymbol{\theta}}\phi_n(\boldsymbol{\theta}_0)\boldsymbol{e}_d(\boldsymbol{\theta}_0) = \lambda_{nd}(\boldsymbol{\theta}_0)$. We now phrase minimizing $\phi(\boldsymbol{\theta})$ by observing $\bar{\phi}(\boldsymbol{\theta})$ as neural network training.

BACKPACK-compatible multi-dimensional formulation: Completing the square in Equation (S.15), and in analogy to Equation (S.14), gives

$$
\begin{aligned}
\phi(\boldsymbol{\theta}) &= \left[\sum_{d=1}^{D} \frac{1}{2}\lambda_d(\boldsymbol{\theta}_0)\left(\boldsymbol{e}_d(\boldsymbol{\theta}_0)^\top(\boldsymbol{\theta}-\boldsymbol{\theta}_0) + \frac{\gamma_d(\boldsymbol{\theta}_0)}{\lambda_d(\boldsymbol{\theta}_0)}\right)^2 - \frac{1}{2}\frac{\gamma_d(\boldsymbol{\theta}_0)^2}{\lambda_d(\boldsymbol{\theta}_0)}\right] + \phi_{\text{true}}(\boldsymbol{\theta}_0) \\
&= \left[\sum_{d=1}^{D} \left(\sqrt{\frac{\lambda_d(\boldsymbol{\theta}_0)}{2}}\boldsymbol{e}_d(\boldsymbol{\theta}_0)^\top(\boldsymbol{\theta}-\boldsymbol{\theta}_0) + \frac{\gamma_d(\boldsymbol{\theta}_0)}{\sqrt{2\lambda_d(\boldsymbol{\theta}_0)}}\right)^2 - \frac{1}{2}\frac{\gamma_d(\boldsymbol{\theta}_0)^2}{\lambda(\boldsymbol{\theta}_0)}\right] + \phi_{\text{true}}(\boldsymbol{\theta}_0) \\
&= \left[\sum_{d=1}^{D} \left(\sqrt{\frac{\lambda_d(\boldsymbol{\theta}_0)}{2}}\boldsymbol{e}_d(\boldsymbol{\theta}_0)^\top\boldsymbol{\theta} - \left(\sqrt{\frac{\lambda_d(\boldsymbol{\theta}_0)}{2}}\boldsymbol{e}_d(\boldsymbol{\theta}_0)^\top\boldsymbol{\theta}_0 - \frac{\gamma_d(\boldsymbol{\theta}_0)}{\sqrt{2\lambda_d(\boldsymbol{\theta}_0)}}\right)\right)^2 \right. \\
&\qquad \left. - \frac{1}{2}\frac{\gamma_d(\boldsymbol{\theta}_0)^2}{\lambda_d(\boldsymbol{\theta}_0)}\right] + \phi_{\text{true}}(\boldsymbol{\theta}_0) \\
&= \frac{1}{D}\left[\sum_{d=1}^{D} \left(\sqrt{\frac{\lambda_d(\boldsymbol{\theta}_0)D}{2}}\boldsymbol{e}_d(\boldsymbol{\theta}_0)^\top\boldsymbol{\theta} - \left(\sqrt{\frac{\lambda_d(\boldsymbol{\theta}_0)D}{2}}\boldsymbol{e}_d(\boldsymbol{\theta}_0)^\top\boldsymbol{\theta}_0 - \sqrt{\frac{D}{2\lambda_d(\boldsymbol{\theta}_0)}}\gamma_d(\boldsymbol{\theta}_0)\right)\right)^2 \right. \\
&\qquad \left. - \frac{1}{2}\frac{D\gamma_d(\boldsymbol{\theta}_0)^2}{\lambda_d(\boldsymbol{\theta}_0)}\right] + \phi_{\text{true}}(\boldsymbol{\theta}_0) \\
&= \frac{1}{D}\left\|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\right\|^2 + \text{const.}
\end{aligned}
\tag{S.19a}
$$

with

$$
\boldsymbol{X} = \sqrt{\frac{D}{2}}\begin{pmatrix} \sqrt{\lambda_1}\boldsymbol{e}_1(\boldsymbol{\theta}_0)^\top \\ \vdots \\ \sqrt{\lambda_D}\boldsymbol{e}_D(\boldsymbol{\theta}_0)^\top \end{pmatrix} \in \mathbb{R}^{D\times D}\,, \tag{S.19b}
$$

$$
\boldsymbol{y} = \sqrt{\frac{D}{2}}\begin{pmatrix} \sqrt{\lambda_1(\boldsymbol{\theta}_0)}\boldsymbol{e}_1(\boldsymbol{\theta}_0)^\top\boldsymbol{\theta}_0 - \frac{\gamma_1(\boldsymbol{\theta}_0)}{\sqrt{\lambda_1(\boldsymbol{\theta}_0)}} \\ \vdots \\ \sqrt{\lambda_D(\boldsymbol{\theta}_0)}\boldsymbol{e}_D(\boldsymbol{\theta}_0)^\top\boldsymbol{\theta}_0 - \frac{\gamma_D(\boldsymbol{\theta}_0)}{\sqrt{\lambda_D(\boldsymbol{\theta}_0)}} \end{pmatrix} \in \mathbb{R}^{D}\,, \tag{S.19c}
$$

$$
\text{const.} = \left(\sum_{d=1}^{D} -\frac{1}{2}\frac{\gamma_d(\boldsymbol{\theta}_0)^2}{\lambda_d(\boldsymbol{\theta}_0)}\right) + \phi_{\text{true}}(\boldsymbol{\theta}_0) \in \mathbb{R}\,. \tag{S.19d}
$$

Note that we extracted a fraction $1/D$. This is due to our rephrasing goal as neural network training.

With that, we conclude that minimizing Equation (S.17) through noisy observations of the form Equation (S.18) is equivalent to the following neural network training: Assume the optimizer's current iterate to be $\boldsymbol{\theta}_0$. Then, the following sequence defines a training iteration:

1. Generate data
   (a) Generate orthonormal eigenvectors $\{\boldsymbol{e}_d(\boldsymbol{\theta}_0)\}_{d=1}^{D}$ with $\boldsymbol{e}_d(\boldsymbol{\theta}_0)^\top\boldsymbol{e}_{d'}(\boldsymbol{\theta}_0) = \delta_{dd'}$
   (b) Draw curvature and gradient samples $\{(\lambda_{nd}(\boldsymbol{\theta}_0), \gamma_{nd}(\boldsymbol{\theta}_0)\}_{n=1,d=1}^{N,D}$
   (c) Compute inputs and labels

$$
\left\{\boldsymbol{X}_n = \sqrt{\frac{D}{2}}\begin{pmatrix} \sqrt{\lambda_{n1}(\boldsymbol{\theta}_0)}\boldsymbol{e}_1(\boldsymbol{\theta}_0)^\top \\ \vdots \\ \sqrt{\lambda_{nD}(\boldsymbol{\theta}_0)}\boldsymbol{e}_D(\boldsymbol{\theta}_0)^\top \end{pmatrix}\right\}_{n=1}^{N},
$$

$$
\left\{\boldsymbol{y}_n = \frac{D}{2}\begin{pmatrix} \sqrt{\lambda_{n1}}\boldsymbol{e}_1(\boldsymbol{\theta}_0)^\top\boldsymbol{\theta}_0 - \frac{\gamma_{n1}(\boldsymbol{\theta}_0)}{\sqrt{\lambda_{n1}(\boldsymbol{\theta}_0)}} \\ \vdots \\ \sqrt{\lambda_{nD}(\boldsymbol{\theta}_0)}\boldsymbol{e}_D(\boldsymbol{\theta}_0)^\top\boldsymbol{\theta}_0 - \frac{\gamma_{nD}(\boldsymbol{\theta}_0)}{\sqrt{\lambda_{nD}(\boldsymbol{\theta}_0)}} \end{pmatrix}\right\}_{n=1}^{N}
$$

2. Forward pass
   (a) Feed $\{\boldsymbol{X}_n\}$ through a linear layer ($\mathbb{R}^{\dots \times D} \to \mathbb{R}^{\dots \times 1}$ where $\dots$ denotes free axes preserved by the affine transformation[6]) with trainable weight $\boldsymbol{\theta}^\top$ and no bias
   (b) Feed the output $\{\boldsymbol{X}_n \boldsymbol{\theta}\}_{n=1}^N$ through the mean-squared error (MSE) with labels $\{\boldsymbol{y}_n\}_{n=1}^N$,

$$\text{MSE}(\{\boldsymbol{X}_n \boldsymbol{\theta}\}, \{\boldsymbol{y}_n\}) = \frac{1}{N} \frac{1}{D} \sum_{n=1}^{N} \left\| \boldsymbol{X}_n \boldsymbol{\theta} - \boldsymbol{y}_n \right\|^2 = \frac{1}{N} \sum_{n=1}^{N} \phi_n(\boldsymbol{\theta}) = \overline{\phi}(\boldsymbol{\theta}) \,. \quad \text{(S.20)}$$

   Note that the factor $1/D$ is required as the MSE implementation in common machine learning libraries averages the squared residuals over all components.[7]

3. Backward pass: Compute first- and second-order directional derivatives $\{\gamma_{nd}(\boldsymbol{\theta}_0)\}_{n=1,d=1}^{N,D}$ and $\{\lambda_{nd}(\boldsymbol{\theta}_0)\}_{n=1,d=1}^{N,D}$ with VIVIT during a backward pass. They are of same value as the curvature and gradient samples defined in Item 1b
4. Optimizer step: Update the value of $\boldsymbol{\theta}$ using $\{\gamma_{nd}(\boldsymbol{\theta}_0)\}_{n=1,d=1}^{N,D}$ and $\{\lambda_{nd}(\boldsymbol{\theta}_0)\}_{n=1,d=1}^{N,D}$, set $\boldsymbol{\theta}_0$ to $\boldsymbol{\theta}$'s new value

B.5 Experiment on noisy quadratic

We consider a quadratic loss function $\mathcal{L}(\boldsymbol{\theta}) = \boldsymbol{\theta}^\top \boldsymbol{G}\, \boldsymbol{\theta}$ with $\boldsymbol{G}_{ii} = i^2$ for $i \in \{1, \dots, D = 20\}$ and use the noisy quadratic network from Appendix B.4 to gain full control over the directions $\boldsymbol{e}_k$ as well as directional derivative samples γ_{nk}, λ_{nk} observed by the optimizers at each step.

We set $\boldsymbol{e}_k$ to the k-th unit vector for $k \in \{1, \dots, D\}$, $N = 128$ and sample γ_{nk} from a Normal distribution with mean $\boldsymbol{e}_k^\top \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$ (the actual directional first derivative of $\mathcal{L}$) and constant variance $5{,}000$. Note that this variance implies a gradient SNR of at least 2 over all directions at $\boldsymbol{\theta}_{\text{init}}$. Since the gradient vanishes when moving towards the minimum, this SNR becomes arbitrarily low. When sampling λ_{nk}, we have to make sure that these samples are non-negative, since they correspond to projections of positive semi-definite matrices $\boldsymbol{G}_n$ (compare Equation (8b)). This constraint can be incorporated by sampling λ_{nk} from a Gamma distribution with mean $\boldsymbol{e}_k^\top \nabla_{\boldsymbol{\theta}}^2 \mathcal{L}(\boldsymbol{\theta}) \boldsymbol{e}_k$ and constant variance 50. This globally limits the curvature SNR to values above 0.02. We compare SGD (learning rate 10^{-3}) to the second-order optimizers with global damping $\delta \in \{10^{-4}, 10^{-3}, \dots, 10^{2}\}$ and directional damping δ_k using a log-equidistant grid from 10^{-4} to 10^2 with 200 grid points and 100 bootstrap samples. We run all optimizers for 20 steps. To obtain a reliable estimation of optimizer stability, this procedure is repeated 100 times. The results are shown in Figure 3.

C Implementation details

Layer view of backpropagation: Consider a single layer $T_{\boldsymbol{\theta}^{(i)}}^{(i)}$ that transforms inputs $\boldsymbol{z}_n^{(i-1)} \in \mathbb{R}^{h^{(i-1)}}$ into outputs $\boldsymbol{z}_n^{(i)} \in \mathbb{R}^{h^{(i)}}$ by means of a parameter $\boldsymbol{\theta}^{(i)} \in \mathbb{R}^{d^{(i)}}$. During backpropagation for $\boldsymbol{V}$, the layer receives vectors $\boldsymbol{s}_{nc}^{(i)} = (\mathrm{J}_{\boldsymbol{z}_n^{(i)}} f_n)^\top \boldsymbol{s}_{nc}$ from the previous stage (recall $\nabla_f^2 \ell_n = \sum_{c=1}^C \boldsymbol{s}_{nc} \boldsymbol{s}_{nc}^\top$). Parameter contributions $\boldsymbol{v}_{nc}^{(i)}$ to $\boldsymbol{V}$ are obtained by application of its Jacobian,

$$\begin{aligned}
\boldsymbol{v}_{nc}^{(i)} &= \left(\mathrm{J}_{\boldsymbol{\theta}^{(i)}} f_n\right)^\top \boldsymbol{s}_{nc} \\
&= \left(\mathrm{J}_{\boldsymbol{\theta}^{(i)}} \boldsymbol{z}_n^{(i)}\right)^\top \left(\mathrm{J}_{\boldsymbol{z}_n^{(i)}} f_n\right)^\top \boldsymbol{s}_{nc} && \text{(Chain rule)} \\
&= \left(\mathrm{J}_{\boldsymbol{\theta}^{(i)}} \boldsymbol{z}_n^{(i)}\right)^\top \boldsymbol{s}_{nc}^{(i)} \,. && \text{(Definition of } \boldsymbol{s}_{nc}^{(i)}\text{)} \qquad \text{(S.21)}
\end{aligned}$$

Consequently, the contribution of $\boldsymbol{\theta}^{(i)}$ to $\boldsymbol{V}$, denoted by $\boldsymbol{V}^{(i)} \in \mathbb{R}^{d^{(i)} \times NC}$, is

$$\boldsymbol{V}^{(i)} = \frac{1}{\sqrt{N}} \begin{pmatrix} \boldsymbol{v}_{11}^{(i)} & \boldsymbol{v}_{12}^{(i)} & \dots & \boldsymbol{v}_{NC}^{(i)} \end{pmatrix} \quad \text{with} \quad \boldsymbol{v}_{nc}^{(i)} = \left(\mathrm{J}_{\boldsymbol{\theta}^{(i)}} f_n\right)^\top \boldsymbol{s}_{nc} \,. \qquad \text{(S.22)}$$

[6] This complies to the implementation in PyTorch, see `torch.nn.Linear`.
[7] See e.g. the PyTorch implementation of `torch.nn.MSELoss`.

C.1 Optimized Gram matrix computation for linear layers

Our goal is to efficiently extract $\boldsymbol{\theta}^{(i)}$'s contribution to the Gram matrix $\tilde{\mathbf{G}}$, given by

$$\tilde{\mathbf{G}}^{(i)} = {\boldsymbol{V}^{(i)}}^\top \boldsymbol{V}^{(i)} \in \mathbb{R}^{NC \times NC} . \tag{S.23}$$

Gram matrix via expanding $\boldsymbol{V}^{(i)}$: One way to construct $\boldsymbol{G}^{(i)}$ is to first expand $\boldsymbol{V}^{(i)}$ (Equation (S.22)) via the Jacobian $\mathrm{J}_{\boldsymbol{\theta}^{(i)}} \boldsymbol{z}_n^{(i)}$, then contract it (Equation (S.23)). This can be a memory bottleneck for large linear layers which are common in many architectures close to the network output. However if only the Gram matrix rather than $\boldsymbol{V}$ is required, structure in the Jacobian can be used to construct $\tilde{\mathbf{G}}^{(i)}$ without expanding $\boldsymbol{V}^{(i)}$ and thus reduce this overhead.

Optimization for linear layers: Now, let $T^{(i)}_{\boldsymbol{\theta}^{(i)}}$ be a linear layer with weights $\boldsymbol{W}^{(i)} \in \mathbb{R}^{h^{(i)} \times h^{(i-1)}}$, i.e. $\boldsymbol{\theta}^{(i)} = \mathrm{vec}(\boldsymbol{W}^{(i)}) \in \mathbb{R}^{d^{(i)} = h^{(i)} h^{(i-1)}}$ with column stacking convention for vectorization,

$$T^{(i)}_{\boldsymbol{\theta}^{(i)}} : \quad \boldsymbol{z}_n^{(i)} = \boldsymbol{W}^{(i)} \boldsymbol{z}_n^{(i-1)} .$$

The Jacobian is

$$\mathrm{J}_{\boldsymbol{\theta}^{(i)}} \boldsymbol{z}_n^{(i)} = {\boldsymbol{z}_n^{(i-1)}}^\top \otimes \boldsymbol{I}_{h^{(i)}} . \tag{S.24}$$

Its structure can be used to directly compute entries of the Gram matrix without expanding $\boldsymbol{V}^{(i)}$,

$$\begin{aligned}
\left[\tilde{\mathbf{G}}^{(i)}\right]_{(nc)(n'c')} &= {\boldsymbol{v}_{nc}^{(i)}}^\top \boldsymbol{v}_{n'c'}^{(i)} && \text{(Equation (S.23))} \\
&= {\boldsymbol{s}_{nc}^{(i)}}^\top \left(\mathrm{J}_{\boldsymbol{\theta}^{(i)}} \boldsymbol{z}_n^{(i)}\right) \left(\mathrm{J}_{\boldsymbol{\theta}^{(i)}} \boldsymbol{z}_{n'}^{(i)}\right)^\top \boldsymbol{s}_{n'c'}^{(i)} \\
&= {\boldsymbol{s}_{nc}^{(i)}}^\top \left({\boldsymbol{z}_n^{(i-1)}}^\top \otimes \boldsymbol{I}_{h^{(i)}}\right) \left({\boldsymbol{z}_{n'}^{(i-1)}}^\top \otimes \boldsymbol{I}_{h^{(i)}}\right)^\top \boldsymbol{s}_{n'c'}^{(i)} && \text{(Equation (S.24))} \\
&= {\boldsymbol{s}_{nc}^{(i)}}^\top \left({\boldsymbol{z}_n^{(i-1)}}^\top \boldsymbol{z}_{n'}^{(i-1)} \otimes \boldsymbol{I}_{h^{(i)}}\right) \boldsymbol{s}_{n'c'}^{(i)} && \text{(Equation (S.21))} \\
&= {\boldsymbol{z}_n^{(i-1)}}^\top \boldsymbol{z}_{n'}^{(i-1)} {\boldsymbol{s}_{nc}^{(i)}}^\top \boldsymbol{I}_{h^{(i)}} \boldsymbol{s}_{n'c'}^{(i)} && ({\boldsymbol{z}_n^{(i-1)}}^\top \boldsymbol{z}_{n'}^{(i-1)} \in \mathbb{R}) \\
&= \left({\boldsymbol{z}_n^{(i-1)}}^\top \boldsymbol{z}_{n'}^{(i-1)}\right) \left({\boldsymbol{s}_{nc}^{(i)}}^\top \boldsymbol{s}_{n'c'}^{(i)}\right) .
\end{aligned}$$

We see that the Gram matrix is built from two Gram matrices based on $\{\boldsymbol{z}_n^{(i-1)}\}_{n=1}^N$ and $\{\boldsymbol{s}_{nc}^{(i)}\}_{n=1,c=1}^{N,C}$, that require $\mathcal{O}(N^2)$ and $\mathcal{O}((NC)^2)$ memory, respectively. In comparison, the naïve approach via $\boldsymbol{V}^{(i)} \in \mathbb{R}^{d^{(i)} \times NC}$ scales with the number of weights, which is often comparable to D. For instance, the 3C3D architecture on CIFAR-10 has $D = 895{,}210$ and the largest weight matrix has $d^{(i)} = 589{,}824$, whereas $NC = 1{,}280$ during training (Schneider et al., 2019).

C.2 Implicit multiplication with the inverse (block-diagonal) GGN

Inverse GGN-vector products: Consider the damped Newton step of Equation (9) that requires multiplication by $(\boldsymbol{G} + \delta \boldsymbol{I}_D)^{-1}$.[8] By means of Equation (3) and the matrix inversion lemma,

$$\begin{aligned}
(\delta \boldsymbol{I}_D + \boldsymbol{G})^{-1} &= \left(\delta \boldsymbol{I}_D + \boldsymbol{V}\boldsymbol{V}^\top\right)^{-1} && \text{(Equation (3))} \\
&= \frac{1}{\delta} \left(\boldsymbol{I}_D + \frac{1}{\delta} \boldsymbol{V}\boldsymbol{V}^\top\right)^{-1} \\
&= \frac{1}{\delta} \left[\boldsymbol{I}_D - \frac{1}{\delta} \boldsymbol{V} \left(\boldsymbol{I}_{NC} + \boldsymbol{V}^\top \frac{1}{\delta} \boldsymbol{V}\right)^{-1} \boldsymbol{V}^\top\right] && \text{(Matrix inversion lemma)} \\
&= \frac{1}{\delta} \left[\boldsymbol{I}_D - \boldsymbol{V} \left(\delta \boldsymbol{I}_{NC} + \boldsymbol{V}^\top \boldsymbol{V}\right)^{-1} \boldsymbol{V}^\top\right] && \text{(Gram matrix)} \\
&= \frac{1}{\delta} \left[\boldsymbol{I}_D - \boldsymbol{V} \left(\delta \boldsymbol{I}_{NC} + \tilde{\mathbf{G}}\right)^{-1} \boldsymbol{V}^\top\right] .
\end{aligned} \tag{S.25}$$

[8] $\delta \boldsymbol{I}_D$ can be replaced by other easy-to-invert matrices.

Inverse GGN-vector products require inversion of the damped Gram matrix as well as applications of $\boldsymbol{V}, \boldsymbol{V}^\top$ for the transformations between Gram and parameter space.

Inverse block-diagonal GGN-vector products: Next, we replace the full GGN by its block diagonal approximation $\boldsymbol{G} \approx \boldsymbol{G}_{\text{BDA}} = \text{diag}(\boldsymbol{G}^{(1)}, \boldsymbol{G}^{(2)}, \dots)$ with

$$\boldsymbol{G}^{(i)} = \boldsymbol{V}^{(i)} {\boldsymbol{V}^{(i)}}^\top \in \mathbb{R}^{d^{(i)} \times d^{(i)}}$$

and $\boldsymbol{V}^{(i)}$ as in Equation (S.22). Then, inverse multiplication reduces to each block,

$$\boldsymbol{G}_{\text{BDA}}^{-1} = \text{diag}\left({\boldsymbol{G}^{(1)}}^{-1}, {\boldsymbol{G}^{(2)}}^{-1}, \dots \right) .$$

If again a damped Newton step is considered, we can reuse Equation (S.25) with the substitutions

$$\left(\boldsymbol{G}, D, \boldsymbol{V}, \boldsymbol{V}^\top, \tilde{\mathbf{G}} \right) \leftrightarrow \left(\boldsymbol{G}^{(i)}, d^{(i)}, \boldsymbol{V}^{(i)}, {\boldsymbol{V}^{(i)}}^\top, \tilde{\mathbf{G}}^{(i)} \right)$$

to apply the inverse and immediately discard the ViViT factors: At backpropagation of layer $T^{(i)}_{\boldsymbol{\theta}^{(i)}}$

1. Compute $\boldsymbol{V}^{(i)}$ using Equation (S.22).
2. Compute $\tilde{\mathbf{G}}^{(i)}$ using Equation (S.23).
3. Compute $\left(\delta \boldsymbol{I}_{NC} + \tilde{\mathbf{G}}^{(i)} \right)^{-1}$.
4. Apply the inverse in Equation (S.25) with the above substitutions to the target vector.
5. Discard $\boldsymbol{V}^{(i)}$, ${\boldsymbol{V}^{(i)}}^\top$, $\tilde{\mathbf{G}}^{(i)}$, and $\left(\delta \boldsymbol{I}_{NC} + \tilde{\mathbf{G}}^{(i)} \right)^{-1}$. Proceed to layer $i - 1$.

Note that the above scheme should only be used for parameters that satisfy $d^{(i)} > NC$, i.e. $\dim(\boldsymbol{G}^{(i)}) > \dim(\tilde{\mathbf{G}}^{(i)})$. Low-dimensional parameters can be grouped with others to increase their joint dimension, and to control the block structure of $\boldsymbol{G}_{\text{BDA}}$.