VIVIT: Curvature access through the generalized Gauss-Newton’s low-rank structure

Abstract

Curvature in form of the Hessian or its generalized Gauss-Newton (GGN) approximation is valuable for algorithms that rely on a local model for the loss to train, compress, or explain deep networks. Existing methods based on implicit multiplication via automatic differentiation or Kronecker-factored block diagonal approximations do not consider noise in the mini-batch. We present VIVIT, a curvature model that leverages the GGN’s low-rank structure without further approximations. It allows for efficient computation of eigenvalues, eigenvectors, as well as per-sample first- and second-order directional derivatives. The representation is computed in parallel with gradients in one backward pass and offers a fine-grained cost-accuracy trade-off, which allows it to scale. As examples for VIVIT’s usefulness, we investigate the directional gradients and curvatures during training, and how noise information can be used to improve the stability of second-order methods.

1 Introduction & Motivation

The large number of trainable parameters in deep neural networks imposes computational constraints on the information that can be made available to optimization algorithms. Standard machine learning libraries [1, 26] mainly provide access to first-order information in the form of average mini-batch gradients. This is a limitation that complicates the development of novel methods that may outperform the state-of-the-art: They must use the same objects to remain easy to implement and use, and to rely on the highly optimized code of those libraries. There is evidence that this has led to stagnation in the performance of first-order optimizers [32]. Here, we thus study how to provide efficient access to richer information, namely higher-order derivatives and full statistics of the mini-batch loss.

Recent advances in automatic differentiation [5, 8] have made such information more readily accessible through vectorization of algebraic structure in the differentiated loss. We leverage and extend this functionality to efficiently access curvature in form of the Hessian’s generalized Gauss-Newton (GGN) approximation. It offers practical advantages over the Hessian and is established for training [19, 21], compressing [36], or adding uncertainty to [29, 28, 17] neural nets. It is also linked theoretically to the natural gradient method [3] via the Fisher information matrix [20, Section 9.2], and has been used to investigate the generalization of neural networks [16, 37].

Traditional ways to access curvature fall into two categories. Firstly, repeated automatic differentiation allows for matrix-free exact multiplication with the Hessian [27] and GGN [35]. Iterative linear and eigensolvers can leverage such functionality to compute Newton steps [19, 39, 11] and spectral properties [30, 31, 2, 12, 25, 38, 13] on arbitrary architectures thanks to the generality of automatic differentiation. However, repeated matrix-vector products represent a critical factor for performance.

*Equal Contribution

Preprint. Under review.
Secondly, Kronecker-factored approximate curvature (K-FAC) [21, 14, 4, 22] constructs an explicit light-weight representation of the GGN based on its algebraic Kronecker structure. The computations are streamlined via gradient backpropagation and the resulting matrices are cheap to store and invert. This allows K-FAC to scale: It has been used successfully with very large mini-batches [23]. One reason for this efficiency is that K-FAC only approximates the GGN’s block diagonal, neglecting interactions across layers. Such terms could be useful, however, for other applications, like uncertainty quantification with Laplace approximations [29, 28, 17] that currently rely on K-FAC. Moreover, due to its specific design for optimization, the Kronecker representation does not become more accurate with more data. It remains a simplification, exact only under assumptions unlikely to be met in practice [21]. This might be a downside for applications that depend on a precise curvature proxy.

Here, we propose ViViT (inspired by $VV^T$ in Equation (3)), a vivid curvature model that leverages the GGN’s low-rank structure. Like K-FAC, its representation is computed in parallel with gradients. But it allows a cost-accuracy trade-off, ranging from the exact GGN to an approximation that has the cost of a single gradient computation. Our contributions are as follows:

- We highlight the GGN’s low-rank structure, and with it a structural limit for the inherent curvature information contained in a mini-batch.
- This low-rank structure allows for efficient computation of various GGN properties: The exact eigenvalue spectrum, including eigenvectors, and per-sample directional derivatives. They enable ViViT to model curvature noise in a mini-batch, in contrast to existing methods.
- Approximations allow ViViT to flexibly trade off computational cost and accuracy. We empirically demonstrate scalability on deep neural networks and provide a fully-featured efficient implementation in PyTORCH [26] on top of the BACKPACK [8] package.\(^\dagger\)

Using ViViT, we illustrate that noise in deep learning poses a challenge for the stability of second-order methods and give a simple example how its quantities can be used to address this problem.

2 Notation & Method

Consider a model $f: \Theta \times \mathbb{X} \rightarrow \mathbb{Y}$ and a dataset $\{(x_n, y_n) \in \mathbb{X} \times \mathbb{Y}\}_{n=1}^N$. For simplicity we use $N$ for both the mini-batch and training set size. The network, parameterized by $\theta \in \Theta$, maps a sample $x_n$ to a prediction $y_n$. Predictions are scored by a convex loss function $\ell: \mathbb{Y} \times \mathbb{Y} \rightarrow \mathbb{R}$ (e.g. cross-entropy or square loss), which compares to the ground truth $y_n$. The training objective $\mathcal{L}: \Theta \rightarrow \mathbb{R}$ is the empirical risk

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{n=1}^N \ell(f(\theta, x_n), y_n).$$

We use $\ell_n(\theta) = \ell(f(\theta, x_n), y_n)$ and $f_n(\theta) = f(\theta, x_n)$ for per-sample losses and predictions. For gradients, we write $g_n(\theta) = \nabla_{\theta} \ell_n(\theta), \mathbf{g}(\theta) = \nabla_{\theta} \mathcal{L}(\theta)$, suppressing $\theta$ if unambiguous. We also set $\Theta = \mathbb{R}^D$ and $\mathbb{Y} = \mathbb{R}^C$ with $D, C$ the model parameter and prediction space dimensions, respectively. For classification, $C$ is the number of classes.

**Hessian & GGN:** Two-fold chain rule application to the split $\ell \circ f$ decomposes the Hessian of Equation (1) into two parts $\nabla^2_{\theta} \ell_n(\theta) = \mathbf{G}(\theta) + \mathbf{R}(\theta) \in \mathbb{R}^{D \times D}$; the positive semi-definite GGN

$$\mathbf{G} = \frac{1}{N} \sum_{n=1}^N (J_{\theta} f_n)^\top \left( \nabla^2_{f_n} \ell_n \right) (J_{\theta} f_n) = \frac{1}{N} \sum_{n=1}^N \mathbf{G}_n$$

and a residual $\mathbf{R} = 1/N \sum_{n=1}^N \sum_{c=1}^C (\nabla^2_{f_n} \ell_n)_{c} [\nabla_{f_n} \ell_n]_{c}$. Here, we use the Jacobian $J_{\theta} b$ that contains partial derivatives of $b$ with respect to $\theta$, $[J_{\theta} b]_{ij} = \partial [b]_{ij} / \partial \theta[a]_j$. As the residual may alter the Hessian’s definiteness – an undesirable property for many applications – we focus on the GGN.

**Low-rank structure:** By basic inequalities, Equation (2) has $\operatorname{rank}(\mathbf{G}) \leq NC$.\(^\dagger\) To make this explicit, we factorize the positive semi-definite Hessian $\nabla^2_{f_n} \ell_n = \sum_{c=1}^C s_{nc} s_{nc}^\top$, where $s_{nc} \in \mathbb{R}^C$ and denote its backpropagated version by $v_{nc} = [J_{\theta} f_n]_c s_{nc} \in \mathbb{R}^D$. Absorbing sums into matrix multiplications, we arrive at the GGN’s outer product representation that lies at the heart of ViViT,

$$\mathbf{G} = \frac{1}{N} \sum_{n=1}^N \sum_{c=1}^C v_{nc} v_{nc}^\top = \mathbf{VV}^\top$$

with $\mathbf{V} = \frac{1}{\sqrt{N}} \left( v_{11} \quad v_{12} \quad \cdots \quad v_{NC} \right) \in \mathbb{R}^{D \times NC}$.\(^\dagger\)

\(^\dagger\)Code available at https://github.com/zv5dmjq5/vivit.

\(^\dagger\)We assume the overparameterized deep learning setting ($NC < D$) and suppress the trivial rank bound $D$.  

2
allows for exact computations with the explicit GGN matrix, at linear rather than quadratic memory cost in $D$. We first formulate the extraction of relevant GGN properties from this factorization, before addressing how to further approximate $V$ to reduce memory and computation costs.

### 2.1 Computing the full GGN eigenspectrum

Each GGN eigenvalue $\lambda \in \mathbb{R}$ satisfies the characteristic polynomial $\det(G - \lambda I_D) = 0$ with identity matrix $I_D \in \mathbb{R}^{D \times D}$. Leveraging the VTV factorization of Equation (3) and the matrix determinant lemma, the $D$-dimensional eigenproblem reduces to that of the much smaller Gram matrix $\tilde{G} = V^\top V \in \mathbb{R}^{NC \times NC}$ which contains pairwise scalar products of $v_{nc}$ (see Appendix A.1),

$$
\det(G - \lambda I_D) = 0 \quad \iff \quad (-\lambda)^{D-NC} \det(\tilde{G} - \lambda I_{NC}) = 0. \tag{4}
$$

With at least $D - NC$ trivial solutions that represent vanishing eigenvalues, the GGN curvature is flat along most directions in parameter space. Nontrivial solutions that give rise to curved directions are fully-contained in the Gram matrix, and hence much cheaper to compute. For example, the left panel of Figure 1a visualizes the full, exact GGN’s empirical spectral density on a mini-batch for a deep convolutional neural net on CIFAR-10. It reproduces the characteristics that have been reported by numerous works, e.g. [31]: An extensive amount of vanishing or small eigenvalues and a small number of larger outliers.

Despite these various Hessian spectral studies which rely on iterative eigensolvers and implicit matrix multiplication [30, 31, 2, 12, 25, 38, 13], we are not aware of works that extract the exact GGN spectrum from its Gram matrix. In contrast to those techniques, this matrix can be computed in parallel with gradients in a single backward pass, which results in less sequential overhead. In fact, our approach allows for plots like Figure 1 to be efficiently live-monitored during training, which may be interesting for practitioners that seek to better understand their model [34].

Eigenvalues themselves can help identify reasonable hyperparameters, like learning rates. But we can also reconstruct the associated eigenvectors in parameter space. These are directions along which curvature information is contained in the mini-batch. Let $\tilde{S}_+ = \{(\lambda_k, \tilde{e}_k)| \lambda_k \neq 0, \tilde{G}\tilde{e}_k = \lambda_k \tilde{e}_k\}_{k=1}^K$ denote the nontrivial Gram spectrum with orthonormal eigenvectors $\tilde{e}_k$ (some form of curvature. Let $\tilde{q}(\theta)$ denote the first-and second-order directional derivatives determined by the local gradient and curvature. Specifically, the projection of Equation (6) onto $e$ gives rise to the first-and second-order directional derivatives

$$
\gamma_e = e^\top \nabla_\theta q(\theta_i) = e^\top g(\theta_i) \in \mathbb{R}, \tag{7a}
$$

$$
\lambda_e = e^\top \nabla_\theta^2 q(\theta_i) e = e^\top G(\theta_i) e \in \mathbb{R}. \tag{7b}
$$

As $G$’s characteristic directions are its eigenvectors, they form a natural basis for the quadratic model. Denoting $\gamma_k = \gamma_{e_k}$ and $\lambda_k = \lambda_{e_k}$ the directional gradients and curvatures along eigenvector $e_k$, we see from Equation (7b) that the directional curvature indeed coincides with the GGN’s eigenvector.
In addition to the mean gradient and curvature along an eigenvector $e_k$, we can expand the sum over samples from Equation (1) to obtain the per-sample contributions to each derivative. Let $\gamma_{nk}$ and $\lambda_{nk}$ denote these first- and second-order derivatives contributions of sample $x_n$ in direction $k$, i.e.

$$\gamma_{nk} = e_k^T g_n = \frac{\tilde{e}_k^T V^T g_n}{\sqrt{\lambda_k}}, \quad (8a)$$

$$\lambda_{nk} = e_k^T G_n e_k = \frac{\tilde{e}_k^T V^T V_n V_n^T V \tilde{e}_k}{\lambda_k} = \frac{\|V_n^T V \tilde{e}_k\|^2}{\lambda_k}, \quad (8b)$$

where $V_n \in \mathbb{R}^{D \times C}$ is the V1V1T factor of $G_n$ corresponding to a scaled sub-matrix of $V$ with fixed sample index. Note that directional derivatives can be evaluated efficiently with the Gram matrix eigenvectors without explicit access to the associated directions in parameter space. In Equation (7) gradient $g$ and curvature $G$ are sums over $g_n$ and $G_n$, respectively. This structure also carries over to the directional derivatives, i.e. $\gamma_k = 1/N \sum_{n=1}^{N} \gamma_{nk}$ and $\lambda_k = 1/N \sum_{n=1}^{N} \lambda_{nk}$.

Access to per-sample directional gradients $\gamma_{nk}$ and curvatures $\lambda_{nk}$ along $G$’s natural directions is a distinct feature of V1V1T. Not only do these quantities provide geometric information about the local loss landscape but also about its directional stochasticity over the mini-batch. Incorporating such knowledge about the noise into algorithms that rely on quadratic models provides a promising way to increase their performance and stability. In Section 4.2 we show how to use this information to make second-order optimization methods more robust against noise.

### 2.3 Computational complexity

So far, we have formulated the computation of GGN eigenvalues (Equation (4)) including eigenvectors (Equation (5)) and per-sample directional derivatives (Equation (8)). Now, we analyze their computational complexity in more detail to identify critical performance factors. Those limitations can effectively be addressed with approximations that allow the costs to be decreased in a fine-grained fashion. We evaluate their effectiveness on deep neural networks to demonstrate that V1V1T scales.

**Relation to gradient computation:** Machine learning libraries are optimized to backpropagate signals $1/N \nabla f \ell_n$ and accumulate the result into the mini-batch gradient $g = 1/N \sum_{n=1}^{N} [J f_n] \nabla f \ell_n$. Each column $v_{nc}$ of $V$ also involves the application of the Jacobian, but to a different vector $s_{nc}$ from the loss Hessian’s symmetric factorization. For popular loss functions, like square and cross-entropy loss, this factorization is analytically known and available at negligible overhead. Hence, computing $V$ basically costs $C$ gradient computations as it involves $NC$ backpropagations, while the gradient requires $N$. However, the practical overhead is expected to be smaller: Computations can re-use information from the Jacobians and enjoy significant additional speedup on parallel processors like GPUs. Because our implementation relies on vectorized Jacobians in BACKPACK [8], we expect similar run time performance as its second-order extensions, which have the same backpropagation.

**Stage-wise discarding $V$ (GGN eigenvalues & directional derivatives):** An interesting structure of $V$ which allows further memory savings without approximation is, that its columns correspond to backpropagated vectors. During backpropagation, sub-matrices of $V$ associated to parameters in the current layer become available once at a time. Many of the above GGN properties can be computed by contraction and accumulation of these stage-wise V1V1T factors, allowing them to be discarded immediately after their use.

One example is the Gram matrix $\tilde{G}$, which is formed by the pairwise scalar products of $\{v_{nc}\}_{n=1}^{N,C,c=1}$ in $O((NC)^2 D)$ operations. The spectral decomposition $\tilde{G}$ has additional cost of $O((NC)^3)$. Similarly, the terms required for the directional derivatives in Equation (8) can be built up stage-wise: First-order derivatives $\{\gamma_{nk}\}_{n=1,k=1}^{N,K}$ require the vectors $\{V^T g_n \in \mathbb{R}^{NC}\}_{n=1}^{N}$ that cost $O(N^2 CD)$ operations. Second-order derivatives are basically for free, as the matrices $\{V_n^T V \in \mathbb{R}^{NC \times NC}\}_{n=1}^{N}$ are already available from $\tilde{G}$.

**GGN eigenvectors:** Raising one Gram matrix eigenvector $\tilde{e}_k$ to the GGN eigenvector $e_k$ through application of $V^T$ (Equation (5)) costs $O(NCD)$ operations. However, repeated application of $V$ can be avoided for raising weighted sums of the form $\sum_k (c_k/\sqrt{\lambda_k}) e_k$ with arbitrary weights $c_k \in \mathbb{R}$.
The summation can be performed in the Gram space at negligible overhead, and only a single vector \( \sum_k c_k \tilde{e}_k \) needs to be transformed. For instance, this allows for efficient aggregation of Newton steps along directions in the Gram space before transforming them to parameter space (see Section 4.2).

2.4 Approximations & Implementation

Although the GGN’s representation by \( V \) has linear memory cost in \( D \), it requires memory equivalent to \( NC \) model copies. Of course, this is infeasible for many networks and data sets, e.g. IMAGENET (\( C = 1000 \)). So far, our formulation was concerned with exact computations. We now present approximations that allow \( N \) and \( C \) in the above cost analysis to be replaced by smaller numbers, enabling \( V^T V \) to trade-off accuracy and performance.

**Curvature sub-sampling & MC approximation:** To reduce the scaling in \( C \), we can approximate the factorization \( \nabla^2 \ell_n(\theta) = \sum_{c=1}^C s_{nc} s_{nc}^\top \) by a smaller set of vectors. One principled approach is to draw MC samples \( \{\tilde{s}_{nm}\} \) such that \( E_m[\tilde{s}_{nm} \tilde{s}_{nm}^\top] = \nabla^2 \ell_n(\theta) \) as in [8]. This reduces the scaling of backpropagated vectors from \( C \) to the number of MC samples (1 in the following). Another commonly used independent approximation to reduce the scaling with \( N \), is to sub-sample curvature by using only a subset of mini-batch samples in the computation [6, 39].

**Parameter groups (block-diagonal approximation):** Some applications, e.g. computing Newton steps, require \( V \) to be kept in memory for performing the transformation into the parameter space. Still, we can reduce costs by using the GGN’s diagonal blocks \( \{G^{(i)}\}_{i=1}^L \) of each layer, rather than the full matrix \( G \). Such blocks are available during backpropagation and can thus be used and discarded step by step. In addition to the previously described approximations for reducing the costs in \( N \) and \( C \), this technique tackles scaling with \( D \).

**A concrete example & implementation details:** To visualize the quality of the above approximations, Figure 1a shows the exact GGN eigenvalue spectrum (left) in comparison to its approximation through curvature sub-sampling (center) and MC (right). Even though the amount of backpropagated vectors is reduced by a factor of 8 and 10, respectively, the approximated spectra capture essential features. We also tabularize the critical batch sizes \( N_{\text{crit}} \) at which their computations experience out-of-memory errors in Figure 1b. On a standard GPU, they exceed the traditional mini-batch size used.
for training the underlying architecture, even for the exact scheme. Using approximations further increases the applicable batch size.\footnote{The critical batch sizes in Figure 1b and c differ strongly for similar reductions of 8 and 10 in the number of backpropagated vectors. This is because most neural networks have final layers with many parameters. During initial stages of backpropagation, expanding $V$ for these weights critically affects peak memory, and thus $N_{crit}$. This can be improved by leveraging structure in the Jacobian (see Appendix C.1).}

As a concrete example for the block-diagonal approximation, we group weights and biases layerwise and compute the exact Newton step implied by the GGN’s block-diagonal approximation (similar to [39]). The critical batch size of these computations is shown in Figure 1c and again supports our approach’s scalability.

BACKPACK’s functionality allows us to efficiently compute individual gradients and $V$ in a single backward pass, using either an exact or MC-factorization of the loss Hessian. To reduce memory consumption, we extend its implementation with a protocol to support mini-batch sub-sampling and parameter groups. By hooks into the package’s extensions, we can discard buffers as soon as possible during backpropagation, effectively implementing all discussed approximations and optimizations.

3 Related work

GGN spectrum & low-rank structure: Other works point out the GGN’s low-rank structure. Botev et al. [4] present the rank bound and propose an alternative to K-FAC based on backpropagating a decomposition of the loss Hessian. Papan [24] presents the factorization in Equation (3) and specifically studies the eigenvalue spectrum’s hierarchy for cross-entropy loss. In this setting, the GGN further decomposes into summands, some of which are then analyzed through similar Gram matrices. These can also be obtained as contractions of $\tilde{G}$, but our approach goes beyond them as it does not neglect terms. We are not aware of other works that obtain the exact spectrum and leverage a highly-efficient fully-parallel implementation. This may be because, until recently [5, 8], vectorized Jacobians that are required to perform those operations efficiently were not available.

Efficient operations with low-rank matrices in deep learning: Chen et al. [7] use Equation (3) for element-wise evaluation of the GGN in fully-connected feed-forward neural networks. They also present a variant based on MC sampling. This element-wise evaluation is then used to construct hierarchical matrix approximations of the GGN. $V^T V$ instead leverages the global low-rank structure that also enjoys efficient eigen-decomposition and linear solves.

Another prominent low-rank matrix in deep learning is the un-centered gradient covariance (sometimes called empirical Fisher). Singh & Alistarh [36] describe implicit multiplication with its inverse\footnote{For completeness, we describe implicit multiplication with the inverse GGN in Appendix C.2.} and apply it for neural network compression, assuming the empirical Fisher as proxy to the Hessian. However, using the gradient covariance as a curvature proxy has limitations, specifically for optimization [18]. In principle though, its low-rank structure also permits the application of our methods from Section 2.

4 Experiments

Different phases are encountered in the course of neural network training [10]. We view those regimes in light of our newly accessible quantities and identify challenges for second-order optimizers. Then, on a simple example, we use those quantities to stabilize such methods.

4.1 Noise during training

The interaction between gradient, curvature, and their stochasticity is crucial for the behavior of optimization methods [37]. Here, we aim to identify characteristic features of gradient noise, curvature noise, and their interaction to gain insights into the conditions under which deep learning optimizers are operating. Training different architectures from the DEEPOBS problem set using the baselines from [8], we evaluate the GGN’s nontrivial eigenvectors and per-sample directional derivatives on a fixed held-out mini-batch with $V^T V$. Figure 2 shows results for the 3C3D architecture trained on
Figure 2: Gradient, curvature and noise during training. Individual columns show the 3C3D architecture’s state at initialization (left), an early (epoch 5, center), and advanced (epoch 68, right) stage of training on CIFAR-10 with SGD (hyperparameters from [33, 8], details in Appendix B.2). For each direction $k$, characterized by its curvature $\lambda_k$, we monitor (a) the directional gradient magnitude; (b) gradient-eigenvector alignment; (c,d) signal-to-noise ratios of curvatures and gradients.

CIFAR-10 using SGD and cross-entropy loss. A broader evaluation that includes more problems, training with ADAM, and a description of procedural details, is given in Appendix B.2. We make the following observations.

**Directional gradient-curvature correlations:** Figure 2a shows pairs of GGN eigenvalues $\lambda_k$ and the associated directional gradient magnitude $|\gamma_k|$ at different training stages. Like for a quadratic function, gradients and curvatures are positively correlated.

**High gradient overlap with top eigenspace & shrinking of non-trivial eigenspace:** Similar to [15], we observe high alignment between the mini-batch gradient and the GGN’s top eigenspace. Specifically, we compute the normalized gradient overlap $\gamma_k^2/\|g\|^2$ with direction $k$. Due to their additivity, gradient overlaps from multiple directions can be grouped by summation. The histograms shown in Figure 2b show such overlaps: Each bin summarizes directions of similar curvature. The accumulated normalized overlap of the gradient with these directions is shown as the height of the histogram bar. The gradient aligns mostly with high-curvature directions. In particular, its overlap with flat directions, $1 - \sum_{k=1}^{K} \gamma_k^2/\|g\|^2$, is smaller than the axis limits. We observe this alignment relation to hold throughout training even though the GGN’s active sub-space dimension $K$ decreases.

**Curvature signal vanishes during training & gradient signal is consistently small:** To quantify the noise in both the directional gradients and curvatures, we compute their signal-to-noise ratios (SNR). It is given by the squared empirical mean divided by the empirical variance of the mini-batch samples $\{\lambda_{nk}\}_{n=1}^N$ and $\{\gamma_{nk}\}_{n=1}^N$ for each non-trivial direction.

Figures 2c,d show the evolution of both ratios. At early stages, the curvature signal along some of the high-curvature directions and parts of the bulk dominates over the noise ($\text{SNR}(\lambda_k) > 1$), i.e. curvatures of per-sample GGNs are similar. As training proceeds, the signal decreases until all directions are dominated by noise ($\text{SNR}(\lambda_k) < 1$). In comparison, the directional gradients do not

---

6We show $|\gamma_k|$ because the directional gradient varies in sign, depending on the eigenvector’s orientation.
exhibit such a pattern. They are always strongly corrupted by noise (SNR(γ_k) < 1), irrespective of the curvature, even though their mean γ_k correlates with λ_k.

On the one hand, the high overlap between gradient and the top GGN eigenspace encourages gradient pre-conditioning with the GGN pseudo-inverse for second-order optimization, as negligible gradient information will be projected out. On the other hand, however, noise in both the gradients and curvatures, which can vary among directions and during training over several orders of magnitude, represents a challenge for such methods. A naive Newton step based on the mean gradient and curvature along each direction may be unstable due to large noise, eliminating all previously made progress. We now turn to those methods and investigate how to improve their stability by equipping them with a noise-aware adaptation strategy to act individually along directions.

4.2 Use-case: Improving damping in second-order methods

Spectral decomposition of Newton’s method into one-dimensional problems:  Second-order optimizers are based on a local quadratic approximation q (cf. Equation (6)) of the objective function. A common approach to make such methods work in practice is to regularize the curvature matrix with a damping term δD, 0 < δ ∈ R. The resulting quadratic is minimized by the update Δθ = −(G + δD)−1q. In the GGN eigenvector basis from S_+ and the trivial spectrum 5_0 = \{(λ_k, e_k) | λ_k = 0, Ge_k = 0\}_k=K+1 we observe different updates in the respective sub-spaces,

\[ Δθ = −(G + δD)^{-1}q = −\sum_{k=1}^{K} \frac{γ_k}{λ_k + δ} e_k - \sum_{k=K+1}^{D} \frac{γ_k}{λ_k + δ} e_k. \]  

(9)

In the non-trivial eigenspace, Δθ takes damped Newton-type steps, while flat directions are updated with SGD at a learning rate δ − 1. Based on the findings in Section 4.1 and [15], we omit the SGD update along flat directions in the following due to their negligible overlap with the gradient.

Global damping versus directional damping:  Due to mini-batching, the gradient and curvature of the quadratic model are corrupted by sub-sampling noise. Hence, the mini-batch averages γ_k = 1/N ∑_{n=1}^{N} γ_{nk} and λ_k = 1/N ∑_{n=1}^{N} λ_{nk} in Equation (9) may deviate considerably from the true values on the noise-free training loss. This noise can lead to overly large steps and thus cause the optimization procedure to become unstable. Damping helps to increase stability as it decreases the step size. However, its relative effect on the step length in a direction e_k depends sensitively on the scale of the curvature λ_k. Also, even at comparable curvature scale, stronger damping may be required in the presence of high uncertainty over γ_k and λ_k. It is thus expected that treating all directions identically yields diminishing returns along certain directions while it is required to keep uncertain ones stable. This suggests a directional damping, i.e. an update of the form Δθ = −∑_{k=1}^{K} γ_k e_k.

Noise-aware directional damping via bootstrap:  We use the quadratic mini-batch model’s loss reduction R(δ_k) to assess the step s_k = −γ_k e_k ∈ R in direction e_k,

\[ R(δ_k) = q(θ) - q(θ + s_k e_k) = −s_k \left( \frac{1}{N} \sum_{n=1}^{N} γ_{nk} \right) - \frac{s_k^2}{2} \left( \frac{1}{N} \sum_{n=1}^{N} λ_{nk} \right). \]  

(10)

From observations on a mini-batch, our goal is to minimize the training loss. Hence, we want 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. Ideally, one would compute directional derivatives on additional samples for this purpose. Instead, we use a resampling technique based on the non-parametric bootstrap [9] to simulate samples for R(δ_k) on other batches, only using information contained in γ_{nk}, λ_{nk} (details in Appendix B). For a given damping δ_k, this provides an indicator of what reduction in training loss to expect with the respective update. Taking the 5% percentile of the bootstrap-generated samples, we obtain a confident lower bound to R(δ_k). We then choose the δ_k that maximizes this lower bound from candidates on a discrete grid.

Evaluation on a noisy quadratic:  We consider a quadratic loss function L(θ) = θ^⊤Gθ with G_{ii} = i^2 for i ∈ {1, ..., D = 20}, and initialize θ at 100 · 1_D. At each step, the optimizer observes unit vectors as directions \{e_k\}_k=1^{K}, and noisy directional derivatives \{γ_{nk}\}_{n=1,k=1}^{N,K}, \{λ_{nk}\}_{n=1,k=1}^{N,K}. Specifically, we design the noise to provide unbiased samples with constant variance (see Appendix B for details). We compare our directional damping based on Equation (10) with constant damping and SGD. The results reported in Figure 3 were obtained from multiple runs with different random seeds.
With constant damping, there is a trade-off between large steps that may cause instabilities (small damping) and slower progress due to smaller steps (large damping). Both extremes are observed in our experiment. For $\delta = 10^{-4}$, there are unstable runs, as indicated by the loss mean and the maximum final distance to the minimum. For large constant damping, the behavior becomes more stable and increasingly resembles SGD. Compared to constant damping, the directional noise-aware damping $\delta_k$ provides the smallest median final distance to the minimum with relatively low variance.

The right panel of Figure 3 shows the directional damping $\delta_k$ in relation to the underlying true curvature $k^2$ for a single run. We make two observations: Firstly, the relative damping increases to compensate for the vanishing gradient signal, as the optimization approaches the minimum. Secondly, since constant noise is applied, the SNR in low curvature directions (yellow) is smaller than in high-curvature directions (red). This is also reflected in the directional damping, which tends to assign larger dampings to the noisier directions. The directional bootstrap damping works as expected and yields stable runs with consistently good final performance.

We provided evidence that VIViT’s quantities may be required for the improvement of second-order methods. As the main contribution of this paper is in delivering these quantities, we deliberately designed simplistic experiments — a full derivation and empirical evaluation would amount to a separate paper of its own right (and would be incommensurate with the space limitations).

5 Conclusion

We have presented VIViT, a curvature model based on the low-rank structure of the Hessian’s generalized Gauss-Newton approximation. This structure allows for efficient extraction of curvature properties, such as the full eigenvalue spectrum and directional gradients and curvatures along the associated eigenvectors. In contrast to alternatives, VIViT offers statistics of these directional derivatives across the mini-batch, and thus a rich noise model.

We demonstrated the utility of these new quantities by studying noise characteristics of representative deep learning tasks. We find that they pose challenges to the stability of second-order methods, and showed, in a simplistic toy model, how VIViT can provide quantities to improve their stability.

VIViT’s representation is efficiently computed in parallel with gradients during a single backward pass. As it mainly relies on vectorized Jacobians, it is even general enough to be integrated into existing machine learning libraries in the future. For the moment, we provide an efficient open-source implementation in PYTORCH by extending the existing BACKPACK library.
Acknowledgments and Disclosure of Funding

The authors gratefully acknowledge financial support by the European Research Council through ERC StG Action 757275 / PANAMA; the DFG Cluster of Excellence “Machine Learning - New Perspectives for Science”, EXC 2064/1, project number 390727645; the German Federal Ministry of Education and Research (BMBF) through the Tübingen AI Center (FKZ: 01IS18039A); and funds from the Ministry of Science, Research and Arts of the State of Baden-Württemberg. Moreover, the authors thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Felix Dangel and Lukas Tatzel. Further, we are grateful to Filip de Roos, Frank Schneider, Jonathan Wenger and Marius Hobbhahn for providing feedback to the manuscript.

References


**A Mathematical details**

A.1 Reducing the GGN eigenvalue problem to the Gram matrix . . . . . . . . . . . . . 12
A.2 Relation between GGN and Gram matrix eigenvectors . . . . . . . . . . . . . . 13

**B Experimental details**

B.1 Performance evaluation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
B.2 Noise analysis during training . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
B.3 Bootstrap damping for second-order methods . . . . . . . . . . . . . . . . . . . . 19
B.4 Derivation of noisy quadratic model . . . . . . . . . . . . . . . . . . . . . . . . . 19
B.5 Experiment on noisy quadratic . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

**C Implementation details**

C.1 Optimized Gram matrix computation for linear layers . . . . . . . . . . . . . . . 24
C.2 Implicit multiplication with the inverse (block-diagonal) GGN . . . . . . . . . . . 24

**A Mathematical details**

A.1 Reducing the GGN eigenvalue problem to the Gram matrix

For Equation (4), consider the left hand size of the GGN’s characteristic polynomial \( \det(G - \lambda I_D) = 0 \). Inserting the ViViT factorization (Equation (3)) and using the matrix determinant lemma yields

\[
\det(-\lambda I_D + G) = \det(-\lambda I_D + VV^T) = \det(I_{NC} + V^T(-\lambda I_D)^{-1}V) \det(-\lambda I_D) = \det(I_{NC} - \frac{1}{\lambda} V^TV) (-\lambda)^D = (-1)^{NC} \det(V^TV - \lambda I_{NC}) (-\lambda)^D = (-\lambda)^{D-NC} \det(\tilde{G} - \lambda I_{NC}).
\]

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 \( \det(\tilde{G} - \lambda I_{NC}) = 0 \).

12
A.2 Relation between GGN and Gram matrix eigenvectors

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

To see the first, consider right-multiplication of the GGN with \( e_k \), then expand the low-rank structure,

\[
G e_k = \frac{1}{\sqrt{\lambda_k}} V V^\top V \tilde{e}_k \quad \text{(Equation (3) and definition of } e_k) \\
= \frac{1}{\sqrt{\lambda_k}} V \tilde{G} \tilde{e}_k \quad \text{(Gram matrix)} \\
= \lambda_k \frac{1}{\sqrt{\lambda_k}} V \tilde{e}_k \quad \text{(Eigenvector property of } \tilde{e}_k) \\
= \lambda_k e_k .
\]

Orthonormality of the \( e_k \) results from the Gram matrix eigenvector orthonormality,

\[
e_j^\top e_k = \left( \frac{1}{\sqrt{\lambda_j}} \tilde{e}_j^\top V^\top \right) \left( \frac{1}{\sqrt{\lambda_k}} V \tilde{e}_k \right) \quad \text{(Definition of } e_j, e_k) \\
= \frac{1}{\sqrt{\lambda_j \lambda_k}} \tilde{e}_j^\top \tilde{G} \tilde{e}_k \quad \text{(Gram matrix)} \\
= \frac{\lambda_k}{\sqrt{\lambda_j \lambda_k}} \tilde{e}_j^\top \tilde{e}_k \quad \text{(Eigenvector property of } \tilde{e}_k) \\
= \delta_{jk} . \quad \text{(Orthonormality)}
\]

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 (\( 1/s \), as in [39]) of mini-batch samples (\( N/sC \) vectors).
- exact, MC: Backpropagate an MC approximation of the loss Hessian (using a single MC sample) for a fraction (\( 1/s \), as in [39]) of mini-batch samples (\( N/s \) 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 DEEPDBS [33]: 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.

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
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.

<table>
<thead>
<tr>
<th>CIFAR-10 3c3d</th>
<th>FASHION-MNIST 2c2d</th>
<th>CIFAR-100 ALL-CNN-C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>N\text{crit}</strong> (eigenvalues, CPU)</td>
<td><strong>N\text{crit}</strong> (eigenvalues, CPU)</td>
<td><strong>N\text{crit}</strong> (eigenvalues, CPU)</td>
</tr>
<tr>
<td>GGN Batch</td>
<td>full</td>
<td>frac</td>
</tr>
<tr>
<td>exact</td>
<td>1166</td>
<td>3004</td>
</tr>
<tr>
<td>MC</td>
<td>8430</td>
<td>14908</td>
</tr>
<tr>
<td><strong>N\text{crit}</strong> (block Newton, CPU)</td>
<td><strong>N\text{crit}</strong> (block Newton, CPU)</td>
<td><strong>N\text{crit}</strong> (block Newton, CPU)</td>
</tr>
<tr>
<td>GGN Batch</td>
<td>full</td>
<td>frac</td>
</tr>
<tr>
<td>exact</td>
<td>1046</td>
<td>2423</td>
</tr>
<tr>
<td>MC</td>
<td>4997</td>
<td>6838</td>
</tr>
<tr>
<td><strong>N\text{crit}</strong> (full Newton, CPU)</td>
<td><strong>N\text{crit}</strong> (full Newton, CPU)</td>
<td><strong>N\text{crit}</strong> (full Newton, CPU)</td>
</tr>
<tr>
<td>GGN Batch</td>
<td>full</td>
<td>frac</td>
</tr>
<tr>
<td>exact</td>
<td>667</td>
<td>2215</td>
</tr>
<tr>
<td>MC</td>
<td>3473</td>
<td>5632</td>
</tr>
<tr>
<td><strong>N\text{crit}</strong> (full Newton, GPU)</td>
<td><strong>N\text{crit}</strong> (full Newton, GPU)</td>
<td><strong>N\text{crit}</strong> (full Newton, GPU)</td>
</tr>
<tr>
<td>GGN Batch</td>
<td>full</td>
<td>frac</td>
</tr>
<tr>
<td>exact</td>
<td>208</td>
<td>727</td>
</tr>
<tr>
<td>MC</td>
<td>1055</td>
<td>1816</td>
</tr>
</tbody>
</table>

involve individual gradients. Second-order directional derivatives $\lambda_{nk}$ (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 $\gamma_{nk}$ (Equation (8a)). As gradients are cheaper to compute, this suggests evaluating them on more samples compared to curvature, see e.g. [39]. 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 $V^{(i)}$ during backpropagation, $V$ 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, ViViTv’s representation is capable to adapt over a wide range.

B.2 Noise analysis during training

**Procedure:** We train the following DEEPOBS [33] architectures with SGD and ADAM: 3c3d on CIFAR-10 ($N = 128$), 2c2d on FASHION-MNIST ($N = 128$), and ALL-CNN-C on CIFAR-100.
Table S.2: Hyperparameter settings for training runs to analyze noise. For both SGD and ADAM, we report their learning rates $\alpha$ (taken from the baselines in [8]) and link to their visualization. Momentum for SGD was fixed to $0.9$, and ADAM uses the default parameters $(\beta_1, \beta_2) = (0.99, 0.999)$.

<table>
<thead>
<tr>
<th>Problem</th>
<th>SGD</th>
<th>ADAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10 3c3d</td>
<td>$\alpha \approx 3.79 \cdot 10^{-3}$ (Figure 2)</td>
<td>$\alpha \approx 2.98 \cdot 10^{-4}$ (Figure S.6)</td>
</tr>
<tr>
<td>FASHION-MNIST 2c2d</td>
<td>$\alpha \approx 2.07 \cdot 10^{-2}$ (Figure S.7)</td>
<td>$\alpha \approx 1.27 \cdot 10^{-4}$ (Figure S.8)</td>
</tr>
<tr>
<td>CIFAR-100 ALL-CNN-C</td>
<td>$\alpha \approx 4.83 \cdot 10^{-1}$ (Figure S.9)</td>
<td>$\alpha \approx 6.95 \cdot 10^{-4}$ (Figure S.10)</td>
</tr>
</tbody>
</table>

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 $\lambda_k$, we monitor (a) the directional gradient magnitude; (b) gradient-eigenvector alignment; (c,d) signal-to-noise ratios of curvatures and gradients.

$(N = 256)$. To assert successful training, we use the hyperparameters from [8] (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

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

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 $\lambda_k$, 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 $\lambda_k$, 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 $\lambda_k$, 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 $\lambda_k$, 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),
\[ \mathcal{R}(\delta_k) = q(\theta) - q(\theta + s_k e_k) = -s_k \left( \frac{1}{N} \sum_{n=1}^{N} \gamma_{nk} \right) - \frac{1}{2} s_k^2 \left( \frac{1}{N} \sum_{n=1}^{N} \lambda_{nk} \right). \]

It describes the reduction of the quadratic mini-batch model when taking a step \( s_k = -\frac{\gamma_{nk}}{\lambda_{nk}} \in \mathbb{R} \) in direction \( e_k \). 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 [9]. The general idea is that we can simulate additional samples for the derivatives in direction \( e_k \) on other batches by resampling from the data \( \{\gamma_{nk}\}_{n=1}^{N}, \{\lambda_{nk}\}_{n=1}^{N} \). First, we draw \( N \) indices \( i_1, \ldots, i_N \in \{1, \ldots, N\} \) with replacement. By taking the mean of \( \gamma_{i_1}, \ldots, \gamma_{i_N} \), 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 \( \lambda_{i_1, \ldots, \lambda_{i_N}} \)). Replacing the directional derivatives \( \frac{1}{N} \sum_{n=1}^{N} \gamma_{nk} \) and \( \frac{1}{N} \sum_{n=1}^{N} \lambda_{nk} \) in the equation above by these new averages \( \frac{1}{N} \sum_{j=1}^{N} \gamma_{ij,k} \) and \( \frac{1}{N} \sum_{j=1}^{N} \lambda_{ij,k} \) gives rise to \( \mathcal{R}_k(\delta_k) \). This allows to create an arbitrary number of samples for any \( \delta_k \).

For a given damping \( \delta_k \), 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 \( \delta_k \) 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 \( \varphi_{\text{true}}(\vartheta_0) \in \mathbb{R} \),
\[ \varphi_{\text{true}}(\vartheta) = \frac{1}{2} \lambda_{\text{true}}(\vartheta_0)(\vartheta - \vartheta_0)^2 + \gamma_{\text{true}}(\vartheta_0)(\vartheta - \vartheta_0) + \varphi_{\text{true}}(\vartheta_0). \] (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 \( \nabla_\vartheta^2 f(\vartheta_0), \nabla_\vartheta^2 f(\vartheta_0) \), but may sometimes deviate to guarantee model properties such as convexity.

Both the objective \( f \) and its local model \( \varphi_{\text{true}} \), 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),
\[ \varphi(\vartheta) = \frac{1}{2} \lambda(\vartheta_0)(\vartheta - \vartheta_0)^2 + \gamma(\vartheta_0)(\vartheta - \vartheta_0) + \varphi_{\text{true}}(\vartheta_0). \] (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
\[ \varphi_n(\vartheta) = \frac{1}{2} \lambda_n(\vartheta_0)(\vartheta - \vartheta_0)^2 + \gamma_n(\vartheta_0)(\vartheta - \vartheta_0). \] (S.13a)
It is common to batch-process multiple samples, compute the average
\[
\bar{\varphi}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \varphi_n(\theta),
\]  
(S.13b)
and use automatic differentiation to compute the quantities employed by an optimizer. Recall that Equation (S.13b) produces the correct noisy first- and second-order partial derivatives, i.e. \(\nabla_{\theta} \varphi_n(\theta_0) = \gamma_n(\theta_0)\) and \(\nabla_{\theta}^2 \varphi_n(\theta_0) = \lambda_n(\theta_0)\). 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),
\[
\varphi(\theta) = \frac{1}{2} \lambda(\theta_0) \left( \theta - \theta_0 \right)^2 + \frac{1}{2} \Gamma(\theta_0) + \varphi_{\text{true}}(\theta_0)
\]
(S.14a)
\[
= \left( \sqrt{\frac{\lambda(\theta_0)}{2}} (\theta - \theta_0) + \frac{\gamma(\theta_0)}{\sqrt{2 \lambda(\theta_0)}} \right)^2 - \frac{1}{2} \Gamma(\theta_0) + \varphi_{\text{true}}(\theta_0)
\]
(S.14b)
\[
= (x \theta - y)^2 + \text{const.}
\]
(S.14c)
with
\[
x = \sqrt{\frac{\lambda(\theta_0)}{2}} \in \mathbb{R},
\]
(S.14b)
\[
y = -\frac{\gamma(\theta_0)}{\sqrt{2 \lambda(\theta_0)}} \in \mathbb{R},
\]
(S.14c)
\[
\text{const.} = -\frac{1}{2} \frac{\gamma(\theta_0)^2}{\lambda(\theta_0)} + \varphi_{\text{true}}(\theta_0) \in \mathbb{R}.
\]
(S.14d)

Note that the data, in which the noisy observations are embedded, must depend on the current location \(\theta_0\). 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 \(\theta_0\). Then, the following steps define a training iteration:

1. **Generate data**
   (a) Draw curvature and gradient samples \(\{\lambda_n(\theta_0), \gamma_n(\theta_0)\}_{n=1}^{N}\)
   (b) Compute inputs \(\{x_n = \sqrt{\lambda_n(\theta_0)/2}\}_{n=1}^{N}\) and labels \(\{y_n = \sqrt{\lambda_n(\theta_0)/2}\theta_0 - \gamma_n(\theta_0)/\sqrt{2\lambda_n(\theta_0)}\}_{n=1}^{N}\)

2. **Forward pass**
   (a) Feed \(\{x_n\}_{n=1}^{N}\) through a linear layer \((\mathbb{R} \rightarrow \mathbb{R},\text{no bias})\) with trainable weight \(\theta \in \mathbb{R}\), set to \(\theta_0\)
   (b) Feed the output \(\{x_n\theta\}_{n=1}^{N}\) through the mean-squared error (MSE) with labels \(\{y_n\}_{n=1}^{N}\)
   \[
   \text{MSE}(\{x_n \theta\}, \{y_n\}) = \frac{1}{N} \sum_{n=1}^{N} (x_n \theta - y_n)^2 = \bar{\varphi}(\theta)
   \]

3. **Backward pass**: Compute the first- and second-order directional derivatives \(\{\gamma_n(\theta_0)\}_{n=1}^{N}\) and \(\{\lambda_n(\theta_0)\}_{n=1}^{N}\) 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 \(\theta\) using \(\{\gamma_n(\theta_0)\}_{n=1}^{N}\) and \(\{\lambda_n(\theta_0)\}_{n=1}^{N}\), set \(\theta_0\) to \(\theta\)’s new value
Multi-dimensional case:  Next, we extend the one-dimensional case to multiple dimensions. Consider a multi-dimensional objective function \( F : \mathbb{R}^D \to \mathbb{R}, \theta \mapsto F(\theta) \) which we want to optimize,

\[
\min_{\theta} F(\theta). 
\]

The local model of \( F \) at \( \theta_0 \in \mathbb{R}^D \), employed by the optimizer, is a convex \( D \)-dimensional quadratic. Let \( \phi_{\text{true}} : \mathbb{R}^D \to \mathbb{R}, \theta \mapsto \phi_{\text{true}}(\theta) \) denote one instance of a model, defined by its local Hessian spectrum \( \{(\lambda_d, \text{true}(\theta_0), e_d(\theta_0))\}_{d=1}^D \) with local curvatures \( \{(\lambda_d, \text{true}(\theta_0))\}_{d=1}^D \) and local gradients \( \{(\gamma_d, \text{true}(\theta_0))\}_{d=1}^D \) along the directions \( \{e_d(\theta_0) \in \mathbb{R}^D\}_{d=1}^D \) and the local function value at \( \theta_0 \in \mathbb{R}^D \),

\[
\phi_{\text{true}}(\theta) = \frac{1}{2} (\theta - \theta_0)^T \nabla^2 \phi(\theta_0) (\theta - \theta_0) + (\theta - \theta_0)^T \nabla \phi(\theta_0) + \phi_{\text{true}}(\theta_0)
\]

\[
= \sum_{d=1}^D \frac{1}{2} \lambda_d, \text{true}(\theta_0) (e_d(\theta_0)^T (\theta - \theta_0))^2 + \gamma_d, \text{true}(\theta_0) e_d(\theta_0)^T (\theta - \theta_0) \right) + \phi_{\text{true}}(\theta_0).
\]

Note that expanding \( \theta = \sum_{d=1}^D \vartheta_d e_d(\theta_0) \) and \( \theta_0 = \sum_{d=1}^D \vartheta_d,0 e_d(\theta_0) \) in terms of scalar coordinates \( \vartheta_d = e_d(\theta_0)^T \) and \( \vartheta_d,0 = e_d(\theta_0) \) decouples Equation (S.15) into one-dimensional quadratics,

\[
\phi_{\text{true}}(\theta) = \sum_{d=1}^D \frac{1}{2} \lambda_d, \text{true}(\theta_0) (\vartheta_d - \vartheta_d,0)^2 + \gamma_d, \text{true}(\theta_0) (\vartheta_d - \vartheta_d,0) \right) + \phi_{\text{true}}(\theta_0)
\]

\[
= \sum_{d=1}^D \varphi_d, \text{true}(\vartheta_d) + \phi_{\text{true}}(\theta_0),
\]

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 \( \{(\lambda_d, \text{true}(\theta_0), e_d(\theta_0))\}_{d=1}^D \) and \( \vartheta_d,0 \) along each direction.

A noisy version \( \phi : \mathbb{R}^D \to \mathbb{R}, \theta \mapsto \phi(\theta) \) of Equation (S.15) replaces the true curvatures \( \{(\lambda_d, \text{true}(\theta_0))\}_{d=1}^D \to \{(\lambda_d(\theta_0))\}_{d=1}^D \) and local gradients \( \{(\gamma_d, \text{true}(\theta_0))\}_{d=1}^D \to \{(\gamma_d(\theta_0))\}_{d=1}^D \) at \( \theta_0 \) with random variables (the offset \( \phi_{\text{true}}(\theta_0) \) is not crucial for optimization, and will thus not be perturbed),

\[
\phi(\theta) = \sum_{d=1}^D \frac{1}{2} \lambda_d(\theta_0) (e_d(\theta_0)^T (\theta - \theta_0))^2 + \gamma_d(\theta_0) e_d(\theta_0)^T (\theta - \theta_0) \right) + \phi_{\text{true}}(\theta_0).
\]

The optimizer observes \( \phi(\theta) \) through \( N \) samples \( \{(\lambda_{nd}(\theta_0), \gamma_{nd}(\theta_0)) \in \mathbb{R}_+ \times \mathbb{R}\}_{n=1}^{N,D} \) drawn from the joint distribution of \( (\lambda_d(\theta_0), \gamma_d(\theta_0)) \). Neglecting the offset, sample \( \{(\lambda_{nd}(\theta_0), \gamma_{nd}(\theta_0))\}_{d=1}^D \) gives rise to

\[
\phi_n(\theta) = \sum_{d=1}^D \frac{1}{2} \lambda_{nd}(\theta_0) (e_d(\theta_0)^T (\theta - \theta_0))^2 + \gamma_{nd}(\theta_0) e_d(\theta_0)^T (\theta - \theta_0). \]

(S.18a)

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

\[
\bar{\phi}(\theta) = \frac{1}{N} \sum_{n=1}^N \phi_n(\theta), \quad \text{(S.18b)}
\]

which will give the correct first- and second-order directional derivatives in an automatic differentiation engine, i.e. \( e_d(\theta_0)^T \nabla \phi_n(\theta_0) = \gamma_{nd}(\theta_0) \) and \( e_d(\theta_0)^T \nabla^2 \phi_n(\theta_0)e_d(\theta_0) = \lambda_{nd}(\theta_0) \). We now phrase minimizing \( \phi(\theta) \) by observing \( \bar{\phi}(\theta) \) as neural network training.
With that, we conclude that minimizing Equation (S.17) through noisy observations of the form

\[ \frac{1}{2} \lambda_d(\theta_0) \left( e_d(\theta_0) \right)^\top (\theta - \theta_0) + \frac{\gamma_d(\theta_0)}{\lambda_d(\theta_0)} \]

Note that we extracted a fraction \( \frac{1}{2} \). This is due to our rephrasing goal as neural network training.

\[ \phi(\theta) = \sum_{d=1}^{D} \frac{1}{2} \lambda_d(\theta_0) \left( e_d(\theta_0) \right)^\top (\theta - \theta_0) + \frac{\gamma_d(\theta_0)}{\lambda_d(\theta_0)} - \frac{1}{2} \frac{\gamma_d(\theta_0)^2}{\lambda_d(\theta_0)} + \phi_{\text{true}}(\theta_0) \]

\[ = \sum_{d=1}^{D} \left( \sqrt{\frac{\lambda_d(\theta_0)}{2}} e_d(\theta_0)^\top (\theta - \theta_0) + \frac{\gamma_d(\theta_0)}{\sqrt{2\lambda_d(\theta_0)}} \right)^2 - \frac{1}{2} \frac{\gamma_d(\theta_0)^2}{\lambda_d(\theta_0)} + \phi_{\text{true}}(\theta_0) \]

\[ = \sum_{d=1}^{D} \left( \sqrt{\frac{\lambda_d(\theta_0)}{2}} e_d(\theta_0)^\top - \left( \sqrt{\frac{\lambda_d(\theta_0)}{2}} e_d(\theta_0)^\top \theta_0 - \frac{\gamma_d(\theta_0)}{\sqrt{2\lambda_d(\theta_0)}} \right) \right)^2 - \frac{1}{2} \frac{\gamma_d(\theta_0)^2}{\lambda_d(\theta_0)} \]

\[ = \frac{1}{D} \| X \theta - y \|^2 + \text{const.} \]  

(S.19a)

with

\[ X = \frac{\sqrt{D}}{2} \left( \sqrt{\lambda_1} e_1(\theta_0)^\top \right) \begin{array}{c} \vdots \\ \sqrt{\lambda_D} e_D(\theta_0)^\top \end{array} \in \mathbb{R}^{D \times D}, \]  

(S.19b)

\[ y = \frac{\sqrt{D}}{2} \left( \sqrt{\lambda_1} e_1(\theta_0)^\top \theta_0 - \frac{\gamma_1(\theta_0)}{\sqrt{\lambda_1(\theta_0)}} \right) \begin{array}{c} \vdots \\ \sqrt{\lambda_D} e_D(\theta_0)^\top \theta_0 - \frac{\gamma_D(\theta_0)}{\sqrt{\lambda_D(\theta_0)}} \end{array} \in \mathbb{R}^D, \]  

(S.19c)

\[ \text{const.} = \sum_{d=1}^{D} \frac{1}{2} \frac{\gamma_d(\theta_0)^2}{\lambda_d(\theta_0)} + \phi_{\text{true}}(\theta_0) \in \mathbb{R}. \]  

(S.19d)

Note that we extracted a fraction \( \frac{1}{2}. \) 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 \( \theta_0. \) Then, the following sequence defines a training iteration:

1. Generate data
   (a) Generate orthonormal eigenvectors \( \{ e_d(\theta_0) \}_{d=1}^{D} \) with \( e_d(\theta_0)^\top e_d(\theta_0) = \delta_{dd} \)
   (b) Draw curvature and gradient samples \( \{ (\lambda_{nd}(\theta_0), \gamma_{nd}(\theta_0)) \}_{n=1,d=1}^{N,D} \)
   (c) Compute inputs and labels

\[ X_n = \frac{\sqrt{D}}{2} \left( \sqrt{\lambda_{n1}} e_1(\theta_0)^\top \right) \begin{array}{c} \vdots \\ \sqrt{\lambda_{nD}} e_D(\theta_0)^\top \end{array} \]  

\[ y_n = \frac{D}{2} \left( \sqrt{\lambda_{n1}} e_1(\theta_0)^\top \theta_0 - \frac{\gamma_{n1}(\theta_0)}{\sqrt{\lambda_{n1}(\theta_0)}} \right) \begin{array}{c} \vdots \\ \sqrt{\lambda_{nD}} e_D(\theta_0)^\top \theta_0 - \frac{\gamma_{nD}(\theta_0)}{\sqrt{\lambda_{nD}(\theta_0)}} \end{array} \]  

\[ X_n \in \mathbb{R}^{D \times D}, \quad y_n \in \mathbb{R}^D. \]
2. Forward pass
   (a) Feed \( \{X_n\} \) through a linear layer \((\mathbb{R}^{\cdots \times D} \to \mathbb{R}^{\cdots \times 1})\) where \(\cdots\) denotes free axes preserved by the affine transformation\(^3\) with trainable weight \(\theta^T\) and no bias
   (b) Feed the output \( \{X_n \theta\}^N_{n=1} \) through the mean-squared error (MSE) with labels \(\{y_n\}^N_{n=1} \).
   \[
   \text{MSE}(\{X_n \theta\}, \{y_n\}) = \frac{1}{N} \frac{1}{D} \sum_{n=1}^N \|X_n \theta - y_n\|^2 = \frac{1}{N} \sum_{n=1}^N \phi_n(\theta) = \overline{\phi}(\theta). \quad (S.20)
   \]
   Note that the factor \(1/d\) is required as the MSE implementation in common machine learning libraries averages the squared residuals over all components.\(^8\)

3. Backward pass: Compute first- and second-order directional derivatives \( \{\gamma_{nd}(\theta_0)\}^N_{n=1,d=1} \) and \( \{\lambda_{nd}(\theta_0)\}^N_{n=1,d=1} \) with \( \mathcal{V} \mathcal{V} \mathcal{T} \) 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 \( \theta \) using \( \{\gamma_{nd}(\theta_0)\}^N_{n=1,d=1} \) and \( \{\lambda_{nd}(\theta_0)\}^N_{n=1,d=1} \), set \( \theta_0 \) to \( \theta \)'s new value

B.5 Experiment on noisy quadratic

We consider a quadratic loss function \( \mathcal{L}(\theta) = \theta^T G \theta \) with \( G_{ii} = i^2 \) for \( i \in \{1, \ldots, D = 20\} \) and use the noisy quadratic network from Appendix B.4 to gain full control over the directions \( e_k \) as well as directional derivative samples \( \gamma_{nk}, \lambda_{nk} \) observed by the optimizers at each step.

We set \( e_k \) to the \( k \)-th unit vector for \( k \in \{1, \ldots, D\} \), \( N = 128 \) and sample \( \gamma_{nk} \) from a Normal distribution with mean \( e_k^T \nabla \mathcal{L}(\theta) \) (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 \( \theta_{\text{init}} \). Since the gradient vanishes when moving towards the minimum, this SNR becomes arbitrarily low. When sampling \( \lambda_{nk} \), we have to make sure that these samples are non-negative, since they correspond to projections of positive semi-definite matrices \( G_{\theta} \) (compare Equation (8b)). This constraint can be incorporated by sampling \( \lambda_{nk} \) from a Gamma distribution with mean \( e_k^T \nabla^2 \mathcal{L}(\theta) e_k \) 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 \( \delta \in \{10^{-4}, 10^{-3}, \ldots, 10^2\} \) and directional damping \( \delta_k \) using a log-equidistant grid from \( 10^{-4} \) to \( 10^4 \) 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 \( T_{\theta(i)} \) that transforms inputs \( z_n^{(i-1)} \in \mathbb{R}^{h^{(i-1)}} \) into outputs \( z_n^{(i)} \in \mathbb{R}^{h^{(i)}} \) by means of a parameter \( \theta^{(i)} \in \mathbb{R}^{d^{(i)}} \). During backpropagation for \( V \), the layer receives vectors \( s_{nc}^{(i)} = (J_{z_n^{(i)}} f_{n})^T s_{nc} \) from the previous stage (recall \( \nabla_{f n} = \sum_{c=1}^C s_{nc} s_{nc}^T \)). Parameter contributions \( v_{nc}^{(i)} \) to \( V \) are obtained by application of its Jacobian,

\[
\begin{align*}
   v_{nc}^{(i)} & = (J_{\theta^{(i)}} f_{n})^T s_{nc} \\
   & = (J_{\theta^{(i)}} z_n^{(i)})^T (J_{z_n^{(i)}} f_{n})^T s_{nc} \\
   & = (J_{\theta^{(i)}} z_n^{(i)})^T s_{nc}^{(i)}. \quad \text{(Chain rule)}
\end{align*}
\]

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

\[
V^{(i)} = \frac{1}{\sqrt{N}} \begin{bmatrix} v_{11}^{(i)} & v_{12}^{(i)} & \cdots & v_{1NC}^{(i)} \end{bmatrix} \quad \text{with} \quad v_{nc}^{(i)} = (J_{\theta^{(i)}} f_{n})^T s_{nc}. \quad (S.21)
\]

---

\(^3\)This complies to the implementation in PYTORCH, see torch.nn.Linear.

\(^8\)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 $\theta^{(i)}$’s contribution to the Gram matrix $\tilde{G}$, given by

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

**Gram matrix via expanding $V^{(i)}$:** One way to construct $G^{(i)}$ is to first expand $V^{(i)}$ (Equation (S.22)) via the Jacobian $J_{\theta^{(i)}}z^{(i)}_n$, 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 $V$ is required, structure in the Jacobian can be used to construct $\tilde{G}$ without expanding $V^{(i)}$ and thus reduce this overhead.

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

$$T^{(i)}_{\theta^{(i)}} : \ z_n^{(i)} = W^{(i)}z_n^{(i-1)}. \quad (S.24)$$

The Jacobian is

$$J_{\theta^{(i)}}z_n^{(i)} = z_n^{(i-1)\top} \otimes I_{h^{(i)}}. \quad (S.24)$$

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

$$\left[\tilde{G}^{(i)}\right]_{(nc)(nc')} = v^{(i)\top}_{nc'}V^{(i)} \quad \text{(Equation (S.23))}$$

$$= s^{(i)\top}_{nc} \left( J_{\theta^{(i)}}z_n^{(i)} \right) \left( J_{\theta^{(i)}}z_n^{(i')} \right) \top s^{(i)}_{nc'}$$

$$= s^{(i)\top}_{nc} \left( \left( z_n^{(i-1)} \otimes I_{h^{(i)}} \right) \left( z_n^{(i-1)} \otimes I_{h^{(i)}} \right) \top \right)$$

$$= s^{(i)\top}_{nc} \left( \left( z_n^{(i-1)} \otimes I_{h^{(i)}} \right) \left( z_n^{(i-1)} \otimes I_{h^{(i)}} \right) \top \right)$$

$$= \left( z_n^{(i-1)}\top \right) \left( z_n^{(i-1)}\top \right) \top.$$

We see that the Gram matrix is built from two Gram matrices based on $\{z_n^{(i-1)}\}_{n=1}^N$ and $\{s^{(i)}\}_{n=1,c=1}^{N,C}$, that require $O(N^2)$ and $O((NC)^2)$ memory, respectively. In comparison, the naive approach via $V^{(i)} \in \mathbb{R}^{d^{(i)} \times NC}$ 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 [33].

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 $(G + \delta I_D)^{-1}$. By means of Equation (3) and the matrix inversion lemma,

$$(\delta I_D + G)^{-1} = (\delta I_D + VV^\top)^{-1} \quad \text{(Equation (3))}$$

$$= \frac{1}{\delta} \left( I_D + \frac{1}{\delta}VV^\top \right)^{-1}$$

$$= \frac{1}{\delta} \left[ I_D - \frac{1}{\delta}V \left( I_{NC} + V^\top \frac{1}{\delta}V \right)^{-1} V^\top \right] \quad \text{(Matrix inversion lemma)}$$

$$= \frac{1}{\delta} \left[ I_D - V \left( \delta I_{NC} + V^\top V \right)^{-1} V^\top \right] \quad \text{(Gram matrix)}$$

$$= \frac{1}{\delta} \left[ I_D - V \left( \delta I_{NC} + \tilde{G} \right)^{-1} V^\top \right]. \quad (S.25)$$

$^9 \delta I_D$ 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 $V, V^\top$ 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 $G \approx G_{\text{BDA}} = \text{diag}(G^{(1)}, G^{(2)}, \ldots)$ with

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

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

$$G^{-1}_{\text{BDA}} = \text{diag}(G^{(1)}^{-1}, G^{(2)}^{-1}, \ldots).$$

If again a damped Newton step is considered, we can reuse Equation (S.25) with the substitutions $(G, D, V, V^\top, \tilde{G}) \leftrightarrow (G^{(i)}, d^{(i)}, V^{(i)}, V^{(i)\top}, \tilde{G}^{(i)})$

to apply the inverse and immediately discard the $V_i V_i^\top$ factors: At backpropagation of layer $T^{(i)}_{\theta_{(i)}}$

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

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

25