DISSECTING HESSIAN: UNDERSTANDING COMMON STRUCTURE OF HESSIAN IN NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Hessian captures important properties of the deep neural network loss landscape. We observe that eigenvectors and eigenspaces of the layer-wise Hessian for neural network objective have several interesting structures – top eigenspaces for different models have high overlap, and top eigenvectors form low rank matrices when they are reshaped into the same shape as the corresponding weight matrix. These structures, as well as the low rank structure of the Hessian observed in previous studies, can be explained by approximating the Hessian using Kronecker factorization. Our new understanding can also explain why some of these structures become weaker when the network is trained with batch normalization. Finally, we show that the Kronecker factorization can be combined with PAC-Bayes techniques to get better explicit generalization bounds.

1 INTRODUCTION

Neural network objectives are complicated and non-convex. However, in practice neural networks can be trained by simple algorithms and they perform well on test data. A common explanation is that neural network objectives have good loss landscapes for optimization and generalization. In this paper we study the structure of Hessians for neural network objectives.

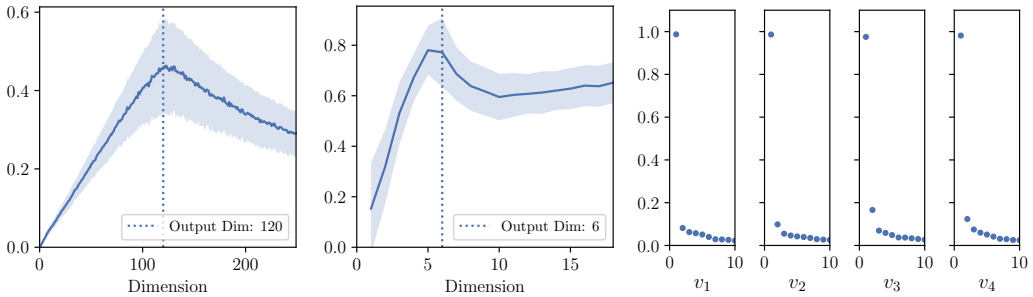
Hessians capture important properties of the loss landscape. For optimization, Hessian information is used explicitly in second order optimization algorithms, and even for gradient-based algorithms properties of the Hessian are often leveraged in analysis (Sra et al., 2012). For generalization, the Hessian captures the local structure of the loss function near a local minimum, which is believed to be related to generalization gaps (Keskar et al., 2017).

Are the Hessians of neural networks similar to a random matrix or highly structured, and does the structure of Hessians change with different architecture or training algorithms? Several previous results including (Sagun et al., 2018; Papyan, 2018) focused on the top eigenvalues of the Hessian for neural networks, observing that in many settings the Hessian often has around c large eigenvalues where c is the number of classes. In this paper, we focus on the structures in the top eigenspace. We show that the top eigenspace of Hessians are far from random – models trained with different random initializations still have a large overlap in their top eigenspace, and the top eigenvectors are close to rank 1 when they are reshaped into the same shape as the corresponding weight matrix.

To understand how the structures of the Hessian may depend on the choice of architecture or training algorithms, we give an explanation of the structures using a Kronecker decomposition. Our new understanding allows us to predict that the structures become weaker when batch normalization is used (which was observed for top eigenvalue in Ghorbani et al. (2019)), or even just when the mean of each layer is set to 0. We also show that our new understanding can be leveraged to improve explicit generalization bounds similar to those in Dziugaite & Roy (2017).

1.1 OUR RESULTS

Structure of Top Eigenspace for Layerwise Hessians: Consider two neural networks trained with different random initializations and potentially different hyper-parameters; their weights are usually nearly orthogonal. One might expect that the top eigenspace of their layer-wise Hessians are also



(a) Overlap between dominate eigenspace of layer-wise Hessian at different minima for fc1:LeNet5 (**left**) with output dimension 120 and conv1:LeNet5 (**right**) with output dimension 6.

(b) Top singular values of the top 4 eigenvectors of the layer-wise Hessian of fc1:LeNet5 after reshaped as matrix.

Figure 1: Some surprising observations on the structure of layer-wise Hessians.

very different. However, this is surprisingly false: the top eigenspace of the layer-wise Hessians have a very high overlap, and the overlap peaks at the dimension of the layer's output (see Fig. 1a). Another interesting phenomenon is that if we express the top eigenvectors of a layer-wise Hessian as a matrix with the same dimensions as the weight matrix, then the matrix is approximately rank 1. In Fig. 1b we show the singular values of several such reshaped eigenvectors.

Understanding Hessian Structure using Kronecker Factorization: We show that both of these new properties of Hessians can be explained by a Kronecker Factorization approximation. Under a decoupling conjecture, we can approximate the layer-wise Hessian using the Kronecker product of the output Hessian and input auto-correlation. This Kronecker approximation directly implies that the eigenvectors of the layer-wise Hessian should be approximately rank 1 when viewed as a matrix. We also analyze the behavior of the two components: the auto-correlation of the input is often very close to a rank 1 matrix and the output Hessian often has around c large eigenvectors. We show that when the input auto-correlation component is approximately rank 1, the layer-wise Hessians indeed have high overlap at the dimension of the layer's output, and the spectrum of the layer-wise Hessian is similar to the spectrum of the output Hessian. On the contrary, when the model is trained with batch normalization, the input auto-correlation matrix is much farther from rank 1 and the layer-wise Hessian often does not have the same low rank structure.

As a direct application of our results, we show that the Hessian structure can be used to improve the PAC-Bayes bound computed in Dziugaite & Roy (2017).

2 Related Works

Hessian-based analysis for neural networks (NNs): Hessian matrices for NNs reflect the second order information about the loss landscape, which is important in characterizing SGD dynamics (Jastrzebski et al., 2019) and related to generalization (Li et al., 2020), robustness to adversaries (Yao et al., 2018) and interpretation of NNs (Singla et al., 2019). People have empirically observed several interesting phenomena of the Hessian, e.g., the gradient during training converges to the top eigenspace of Hessian (Gur-Ari et al., 2018; Ghorbani et al., 2019), and the eigenspectrum of Hessian contains a "spike" which has about $c-1$ large eigenvalues and a continuous "bulk" (Sagun et al., 2016; 2018; Papyan, 2018). People developed different frameworks to explain these, including hierarchical clustering of logit gradients (Papyan, 2019; 2020), independent Gaussian model for logit gradients (Fort & Ganguli, 2019; Fort et al., 2019), and Neural Tangent Kernel (Jacot et al., 2020).

Kronecker factorization for training of NN: There have been papers using Kronecker factorizations to approximate the Fisher Information Matrices (FIM), which is similar to Hessian in neural network setting. This idea can be dated back to Heskes (2000). Martens & Grosse (2015) proposed Kronecker-factored approximate curvature (K-FAC) to approximate the inverse of FIM and perform approximated natural gradient descent (NGD) in training neural networks. Kronecker factored eigenbasis has also been utilized(George et al., 2018). This K-FAC method has been generalized to convolutional networks (Grosse & Martens, 2016), deep reinforcement learning (Wu et al.,

2017), large-scale distributed learning (Ba et al., 2017; Osawa et al., 2019), recurrent neural networks (Martens et al., 2018), Bayesian deep learning (Zhang et al., 2018), and structured pruning (Wang et al., 2019). Unlike these previous works which focus on accelerating computations using Kronecker factorization, in this paper we mainly use Kronecker factorization to explain the structures that arise in the top eigenspace of the layer-wise Hessians.

PAC-Bayes generalization bounds: People have established generalization bounds for neural networks under PAC-Bayes framework(McAllester, 1999), whose bound was further tightened by Langford & Seeger (2001), and Catoni (2007) proposed a faster-rate version. For neural networks, Dziugaite & Roy (2017) proposed the first non-vacuous generalization bound, which used PAC-Bayesian approach with optimization to bound the generalization error for a stochastic neural network. Their bound was then extended to ImageNet scale by Zhou et al. (2019) using compression.

3 Preliminaries and Notations

Basic Notations: In this paper, we generally follow the default notation suggested by Goodfellow et al. (2016). Additionally, for a vector $\boldsymbol{x}$, let $\|\boldsymbol{x}\|$ denote its ℓ_2 norm. For a matrix $\boldsymbol{M}$, let $\|\boldsymbol{M}\|_F$ denote its Frobenius norm. For two matrices $\boldsymbol{M} \in \mathbb{R}^{a_1 \times b_1}, \boldsymbol{N} \in \mathbb{R}^{a_2 \times b_2}$, we use $\boldsymbol{M} \otimes \boldsymbol{N}$ to denote the $(a_1 a_2) \times (b_1 b_2)$ matrix which is the Kronecker product of $\boldsymbol{M}$ and $\boldsymbol{N}$, in particular $[\boldsymbol{M} \otimes \boldsymbol{N}]_{(i_1-1)\times a_2 + i_2, (j_1-1)\times b_2 + j_2} = \boldsymbol{M}_{i_1,i_2} \boldsymbol{N}_{j_1,j_2}$.

Neural Networks: We consider classification problems with cross-entropy loss. For a c-class classification problem, we are given a collection of training samples $S = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$ where $\forall i \in [N], (\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathbb{R}^d \times \mathbb{R}^c$. We assume S is i.i.d. sampled from the underlying data distribution $\mathcal{D}$. Consider an L-layer fully connected ReLU neural network without skip connection $f_{\boldsymbol{\theta}} : \mathbb{R}^d \to \mathbb{R}^c$. With $\sigma(x) = x\mathbf{1}_{x \geq 0}$ as the Rectified Linear Unit (ReLU) function, the output of this network is a series of logits $\boldsymbol{z} \in \mathbb{R}^c$ computed as

$$\boldsymbol{z} := f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{W}^{(L)} \sigma(\boldsymbol{W}^{(L-1)} \sigma(\cdots \boldsymbol{W}^{(2)} \sigma(\boldsymbol{W}^{(1)} \boldsymbol{x} + \boldsymbol{b}^{(1)}) + \boldsymbol{b}^{(2)} \cdots) + \boldsymbol{b}^{(L-1)}) + \boldsymbol{b}^{(L)} \quad (1)$$

We denote $\boldsymbol{\theta} := (\boldsymbol{w}^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{w}^{(2)}, \boldsymbol{b}^{(2)}, \cdots, \boldsymbol{w}^{(L)}, \boldsymbol{b}^{(L)}) \in \mathbb{R}^P$ the parameters of the network. In particular, $\boldsymbol{w}^{(i)}$ is the flattened i-th layer weight coefficient matrix $\boldsymbol{W}^{(i)}$ and $\boldsymbol{b}^{(i)}$ is its bias vector. For convolutional networks, a similar analogue is presented in Appendix A.2.

For a single input $\boldsymbol{x} \in \mathbb{R}^d$ with label $\boldsymbol{y}$ and logit output $\boldsymbol{z}$, let $\boldsymbol{x}^{(p)}$ and $\boldsymbol{z}^{(p)}$ denote the input and output of the p-th layer, and their lengths be $n^{(p)}$ and $m^{(p)}$. For convolutional layers, we consider the number of output channels as $m^{(p)}$ and width of unfolded input as $n^{(p)}$. Note that $\boldsymbol{x}^{(1)} = \boldsymbol{x}, \boldsymbol{z}^{(L)} = \boldsymbol{z} = f_{\boldsymbol{\theta}}(\boldsymbol{x})$. We also denote $\boldsymbol{p} := \mathrm{softmax}(\boldsymbol{z}) = \frac{e^{\boldsymbol{z}}}{\sum_{i=1}^c e^{\boldsymbol{z}_i}}$ the output confidence.

With the loss function $\ell(\boldsymbol{p}, \boldsymbol{y}) = -\sum_{i=1}^c \boldsymbol{y}_i \log(\boldsymbol{p}_i) \in \mathbb{R}^+$ being the cross-entropy loss between the softmax of logits $\boldsymbol{z} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) \in \mathbb{R}^c$ and the one-hot label $\boldsymbol{y} \in \mathbb{R}^c$, the training process of the neural network optimizes parameter $\boldsymbol{\theta}$ to minimize the empirical training loss:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i) = \mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{y}) \in S} \left[\ell\left(\boldsymbol{z}, \boldsymbol{y}\right)\right]. \quad (2)$$

Hessians: Fixing the parameter $\boldsymbol{\theta}$, we use $\boldsymbol{H}_\ell(\boldsymbol{v}, \boldsymbol{x})$ to denote the Hessian of some vector $\boldsymbol{v}$ with respect to scalar loss function ℓ at input $\boldsymbol{x}$.

$$\boldsymbol{H}_\ell(\boldsymbol{v}, \boldsymbol{x}) = \nabla^2_{\boldsymbol{v}} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}) = \nabla^2_{\boldsymbol{v}} \ell(\boldsymbol{z}, \boldsymbol{y}). \quad (3)$$

Note $\boldsymbol{v}$ here can be any vector. For example, the parameter Hessian is $\boldsymbol{H}_\ell(\boldsymbol{\theta}, \boldsymbol{x})$, where $\boldsymbol{v} = \boldsymbol{\theta}$. The layer-wise weight Hessian of the p-th layer is $\boldsymbol{H}_\ell(\boldsymbol{w}^{(p)}, \boldsymbol{x})$.

For simplicity, define $\mathbb{E}$ as the empirical expectation over the training sample S unless explicitly stated otherwise. We focus on the layer-wise weight Hessians $\boldsymbol{H}_{\mathcal{L}}(\boldsymbol{w}^{(p)}) = \mathbb{E}[\boldsymbol{H}_\ell(\boldsymbol{w}^{(p)}, \boldsymbol{x})]$ with respect to loss, which are diagonal blocks in the full Hessian $\boldsymbol{H}_{\mathcal{L}}(\boldsymbol{\theta}) = \mathbb{E}[\boldsymbol{H}_\ell(\boldsymbol{\theta}, \boldsymbol{x})]$ corresponding to the cross terms between the weight coefficients of the same layer. We define $\boldsymbol{M}_{\boldsymbol{x}}^{(p)} := \boldsymbol{H}_\ell(\boldsymbol{z}^{(p)}, \boldsymbol{x})$ as the Hessian of output $\boldsymbol{z}^{(p)}$ with respect to empirical loss. With the notations defined above, we have the p-th layer-wise hessian for a single input as

$$\boldsymbol{H}_\ell(\boldsymbol{w}^{(p)}, \boldsymbol{x}) = \nabla^2_{\boldsymbol{w}^{(p)}} \ell(\boldsymbol{z}, \boldsymbol{y}) = \boldsymbol{M}_{\boldsymbol{x}}^{(p)} \otimes (\boldsymbol{x}^{(p)} \boldsymbol{x}^{(p)T}). \quad (4)$$

It follows that

$$\boldsymbol{H}_{\mathcal{L}}(\boldsymbol{w}^{(p)}) = \mathbb{E}\left[\boldsymbol{M}_{\boldsymbol{x}}^{(p)} \otimes \boldsymbol{x}^{(p)}\boldsymbol{x}^{(p)T}\right] = \mathbb{E}\left[\boldsymbol{M} \otimes \boldsymbol{x}\boldsymbol{x}^T\right]. \tag{5}$$

The subscription $\boldsymbol{x}$ and the superscription (p) will be omitted when there is no confusion, as our analysis primarily focuses on the same layer unless otherwise stated.

4 Kronecker Factorization of Layer-wise Hessian

The fact that layer-wise Hessian for a single sample can be decomposed into Kronecker product of two components natually leads to the following conjecture:

Conjecture (Decoupling Conjecture)**.** The layer-wise Hessian can be approximated by a Kronecker product of the expectation of its two components, that is

$$\boldsymbol{H}_{\mathcal{L}}(\boldsymbol{w}^{(p)}) = \mathbb{E}\left[\boldsymbol{M} \otimes \boldsymbol{x}\boldsymbol{x}^T\right] \approx \mathbb{E}[\boldsymbol{M}] \otimes \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]. \tag{6}$$

In particular, the top eigenvalues and eigenspace of $\boldsymbol{H}_{\mathcal{L}}(\boldsymbol{w}^{(p)})$ is close to those of $\mathbb{E}[\boldsymbol{M}] \otimes \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$.

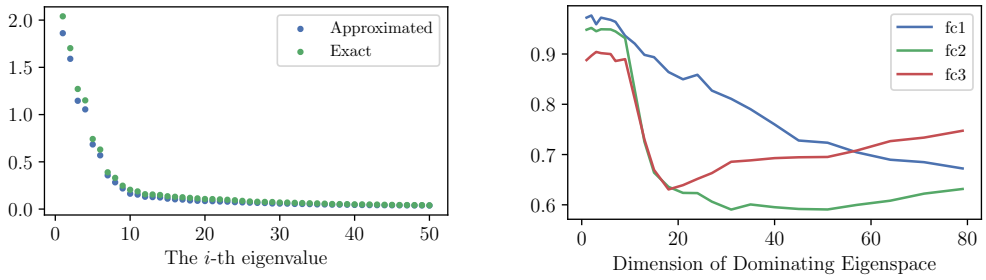
Note that this conjecture is certainly true when $\boldsymbol{M}$ and $\boldsymbol{x}\boldsymbol{x}^T$ are approximately statistically independent. In Section 4.1 and Section 4.2 we will show that this conjecture is true in practice.

Assuming the decoupling conjecture, we can analyze the layer-wise Hessian by analyzing the two components separately. Note that $\mathbb{E}[\boldsymbol{M}]$ is the Hessian of the layer-wise output with respect to empirical loss, and $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ is the auto-correlation matrix of the layer-wise inputs. For simplicity we call $\mathbb{E}[\boldsymbol{M}]$ the output Hessian and $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ the input auto-correlation. For convolutional layers, we define a similar factorization $\mathbb{E}[\boldsymbol{M}] \otimes \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ for the layer-wise Hessian, but with a different $\boldsymbol{M}$ motivated by Grosse & Martens (2016). (See Appendix A.2)

We conduct experiments on the CIFAR-10 (Krizhevsky, 2009) and MNIST (LeCun et al., 1998) datasets using several different fully connected (fc) networks (a fc network with m hidden layers and n neurons each hidden layer is denoted as F-n^m), several variations of LeNet (LeCun et al., 1998), and VGG11 (Simonyan & Zisserman, 2015). The results shown in the main text are variants of LeNet5 trained on CIFAR-10 and F-200^2 trained on MNIST. Other representative results are shown in Appendix D. The eigenvalues and eigenvectors of the exact layer-wise Hessians are approximated using a modified Lanczos algorithm (Golmant et al., 2018), which is described in detail in Appendix C. For simplicity, we use "layer:network" to denote a layer of a particular network. For example, conv2:LeNet5 refers to the second convolutional layer in LeNet5.

4.1 Hessian Approximation

To verify the validity of the approximation, we compare the eigenvalues and top eigenspaces of the approximated Hessian and the true Hessian. To measure the similarity between top eigenspaces, we use the standard definition of subspace overlap below. As shown in Fig. 2, this approximation works reasonably well, especially for the top eigenvalues and eigenspace.



(a) Top eigenvalues of approximated and exact layer-wise Hessian for fc1:F-200^2.

(b) Top eigenspace overlap between approximated and true layer-wise Hessian of F-200^2.

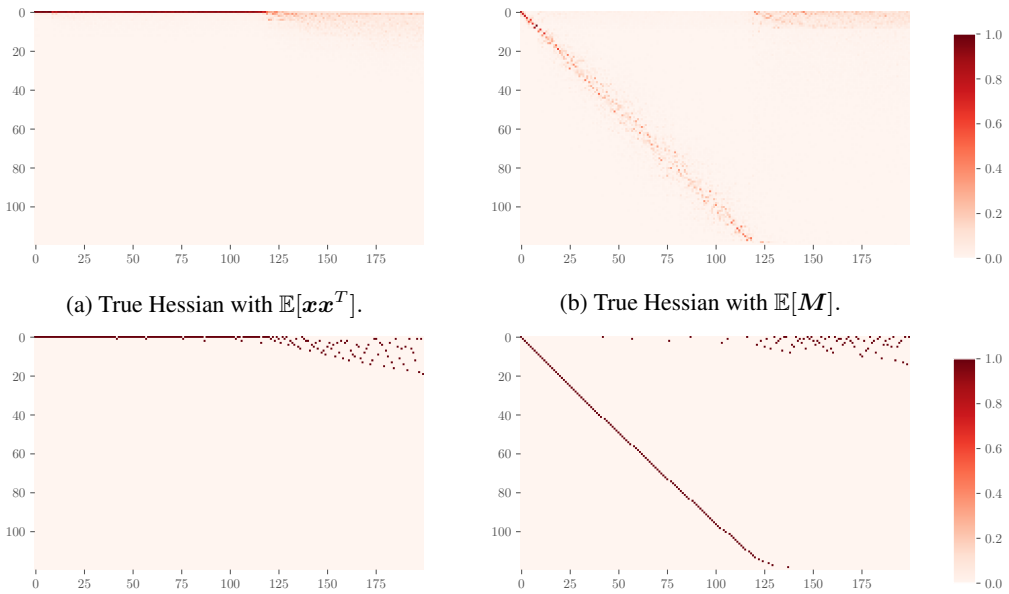
Figure 2: Comparison between the true and approximated layer-wise Hessians of F-200^2. The eigenspace overlap in (b) is defined in Definition 4.1.

Definition 4.1 (Subspace Overlap)**.** For k-dimensional subspaces $\boldsymbol{U}, \boldsymbol{V}$ in $\mathbb{R}^d$ ($d \geq k$) where the basis vectors $\boldsymbol{u}_i$s and $\boldsymbol{v}_i$s are column vectors, with $\boldsymbol{\phi}$ as the size k vector of canonical angles between $\boldsymbol{U}$ and $\boldsymbol{V}$, we define the subspace overlap of $\boldsymbol{U}$ and $\boldsymbol{V}$ as

$$\text{Overlap}(\boldsymbol{U}, \boldsymbol{V}) := \frac{1}{k}\|\boldsymbol{U}^T\boldsymbol{V}\|_F^2 = \frac{1}{k}\|\cos\boldsymbol{\phi}\|^2. \tag{7}$$

Note that when $k = 1$, the overlap is equivalent to the squared dot product between the two vectors.

4.2 Eigenvector Correspondence



(a) True Hessian with $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$.

(b) True Hessian with $\mathbb{E}[\boldsymbol{M}]$.

(c) Approximated Hessian with $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$.

(d) Approximated Hessian with $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$.

Figure 3: Heatmap of Eigenvector Correspondence Matrices for fc1:LeNet5, which has 120 output neurons. Here we take the top left corner of the eigenvector correspondence matrices.

Suppose the i-th eigenvector for $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ is $\boldsymbol{v}_i$ and the j-th eigenvector for $\mathbb{E}[\boldsymbol{M}]$ is $\boldsymbol{u}_j$. Then the Kronecker product $\mathbb{E}[\boldsymbol{M}] \otimes \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ has an eigenvector $\boldsymbol{u}_j \otimes \boldsymbol{v}_i$. Therefore if the decoupling conjecture is true, one would expect that the top eigenvector of the layer-wise Hessian have a clear correspondence with the top eigenvectors of its two components. Note that $\boldsymbol{u} \otimes \boldsymbol{v}$ is just the flattened matrix $\boldsymbol{u}\boldsymbol{v}^T$. For better analysis of eigenvectors, we define the following reshape operation.

Definition 4.2 (Layer-wise Eigenvector Matricization)**.** Consider a layer with input dimension n and output dimension m. For an eigenvector $\boldsymbol{h} \in \mathbb{R}^{mn}$ of its layer-wise Hessian, the matricized form of $\boldsymbol{h}$ is $\text{Mat}(\boldsymbol{h}) \in \mathbb{R}^{m\times n}$ where $\text{Mat}(\boldsymbol{h})_{i,j} = \boldsymbol{h}_{(i-1)m+j}$.

More concretely, to demonstrate the correspondence between the eigenvectors of the layerwise hessian and the eigenvectors of matrix $\mathbb{E}[\boldsymbol{M}]$ and $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$, we introduce "eigenvector correspondence matrices" as shown in Fig. 3.

Definition 4.3 (Eigenvector Correspondence Matrices)**.** For layer-wise Hessian matrix $\boldsymbol{H} \in \mathbb{R}^{mn\times mn}$ with eigenvectors $\boldsymbol{h}_1, \cdots, \boldsymbol{h}_{mn}$, and its corresponding auto-correlation matrix $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T] \in \mathbb{R}^{n\times n}$ with eigenvectors $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n$. The correspondence between $\boldsymbol{v}_i$ and $\boldsymbol{h}_j$ can be defined as

$$\text{Corr}(\boldsymbol{v}_i, \boldsymbol{h}_j) := \|\text{Mat}(\boldsymbol{h}_j)\boldsymbol{v}_i\|^2. \tag{8}$$

For the output Hessian matrix $\mathbb{E}[\boldsymbol{M}] \in \mathbb{R}^{m\times m}$ with eigenvectors $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_m$, we can likewise define correspondence between $\boldsymbol{v}_i$ and $\boldsymbol{h}_j$ as

$$\text{Corr}(\boldsymbol{u}_i, \boldsymbol{h}_j) := \|\text{Mat}(\boldsymbol{h}_j)^T\boldsymbol{u}_i\|^2 \tag{9}$$

We may then define the eigenvector correspondence matrix between $\boldsymbol{H}$ and $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ as a $n \times mn$ matrix whose i, j-th entry is $\mathrm{Corr}(\boldsymbol{v}_i, \boldsymbol{h}_j)$, and the eigenvector correspondence matrix between $\boldsymbol{H}$ and $\mathbb{E}[\boldsymbol{M}]$ as a $m \times mn$ matrix whose i, j-th entry is $\mathrm{Corr}(\boldsymbol{u}_i, \boldsymbol{h}_j)$.

Intuitively, if the i, j-th entry of the corresponding matrix is close to 1, then the eigenvector $\boldsymbol{h}_j$ is likely to be the Kronecker product of $\boldsymbol{v}_i$ (or $\boldsymbol{u}_i$) with some vector. Note that if the decoupling conjecture holds, every eigenvector of the layer-wise Hessian (column of the correspondence matrices) should have a perfect correlation of 1 with exactly one of $\boldsymbol{v}_i$ and one of $\boldsymbol{u}_i$. In Fig. 3 we can see that the correspondence matrices for the true layer-wise Hessian approximately satisfies this property for top eigenvectors. The similarity between the correspondence patterns for the true and approximated Hessian also verifies the validity of the Kronecker approximation for dominate eigenspace. From Fig. 3a and Fig. 3c, the top m eigenvectors of the true layer-wise Hessian and the approximated Hessian are all highly correlated with $\boldsymbol{v}_1$, the first eigenvector of $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$. From Fig. 3b and Fig. 3d, the correspondence with the $\mathbb{E}[\boldsymbol{M}]$ component has a near diagonal pattern for both the true Hessian and the Kronecker approximation. Thus for small i we have $\boldsymbol{h}_i \approx \boldsymbol{v}_1 \otimes \boldsymbol{u}_i$.

4.3 Structure of Auto-correlation Matrix $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$

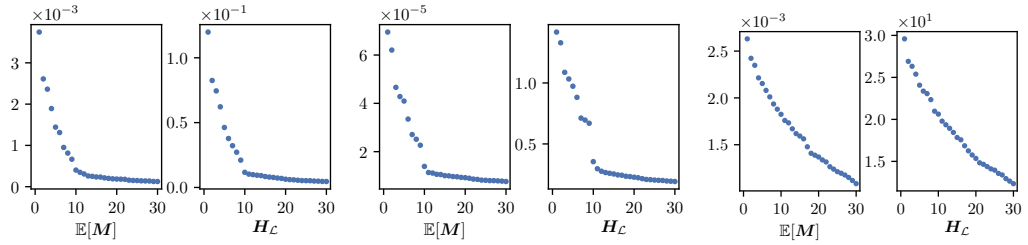
For the auto-correlation matrix, a key observation is that the input $\boldsymbol{x}$ for most layers are outputs of a ReLU, therefore it has nonnegative coordinates. We can decompose the auto-correlation matrix as

$$\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T] = \mathbb{E}[\boldsymbol{x}]\mathbb{E}[\boldsymbol{x}]^T + \mathbb{E}[(\boldsymbol{x} - \mathbb{E}[\boldsymbol{x}])(\boldsymbol{x} - \mathbb{E}[\boldsymbol{x}])^T] =: \mathbb{E}[\boldsymbol{x}]\mathbb{E}[\boldsymbol{x}]^T + \boldsymbol{\Sigma}_{\boldsymbol{x}}, \tag{10}$$

where $\boldsymbol{\Sigma}_{\boldsymbol{x}}$ is the auto-covariance matrix. As every sample $\boldsymbol{x}$ is nonnegative, the expectation $\mathbb{E}[\boldsymbol{x}]\mathbb{E}[\boldsymbol{x}]^T$ has a large norm and usually dominates the covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{x}}$. Empirical results have shown that the $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ matrices are all close to rank 1 throughout the training trajectory. Moreover, its principle component is almost equal to the normalized $\mathbb{E}[\boldsymbol{x}]$ ($\hat{\mathbb{E}}[\boldsymbol{x}] = \frac{\mathbb{E}[\boldsymbol{x}]}{\|\mathbb{E}[\boldsymbol{x}]\|}$), as their inner product are usually larger than 0.998. This suggests that $\mathbb{E}[\boldsymbol{x}]\mathbb{E}[\boldsymbol{x}]^T$ is approximately equal to $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ and dominates the covariance $\boldsymbol{\Sigma}_{\boldsymbol{x}}$.It is also verified as the spectral norm of $\mathbb{E}[\boldsymbol{x}]\mathbb{E}[\boldsymbol{x}]^T$ is usually more than 5 times the spectral norm of $\boldsymbol{\Sigma}_{\boldsymbol{x}}$ in our experiments (as in Appendix D.1).

4.4 Structure of $\mathbb{E}[\boldsymbol{M}]$ and Outliers in the Hessian Eigenspectrum

As mentioned before, several privious work observed that there is a gap in Hessian eigenvalue distribution around the number of classes c (where $c = 10$ in our experiments).



(a) fc1:LeNet5 at initialization (CIFAR-10).

(b) fc1:LeNet5 at minimum (CIFAR-10).

(c) fc1:LeNet5 at minimum (CIFAR-10 Random Labeled).

Figure 4: Eigenspectrum of $\mathbb{E}[\boldsymbol{M}]$ and $\boldsymbol{H}_{\mathcal{L}}(\boldsymbol{w}^{(p)})$. The vertical axis represents the eigenvalues.

Since $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ is close to rank 1 and the Kronecker factorization is a good approximation for top eigenspace, the top eigenvalues of layer-wise Hessian can be approximated as the top eigenvalues of $\mathbb{E}[\boldsymbol{M}]$ multiplied by the first eigenvalue of $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$. Thus, the top eigenvalues of Hessians should have the same relative ratios as the top eigenvalues of their corresponding $\mathbb{E}[\boldsymbol{M}]$'s. Therefore, the outlier should also appear in $\mathbb{E}[\boldsymbol{M}]$.

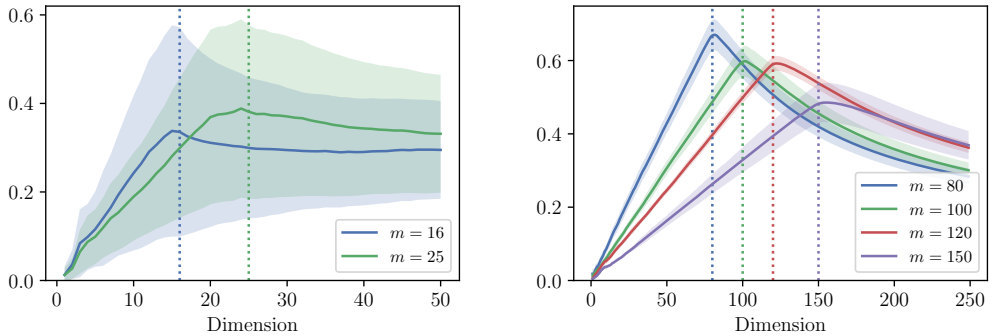
Fig. 4 shows the similarity of eigenvalue spectrum between $\mathbb{E}[\boldsymbol{M}]$ and layer-wise Hessians in different situations, agreeing with our prediction. However, the outliers only appear at initialization and at minimum for true labels (Fig. 4a and Fig. 4b), but not at miminum for random labels (see Fig. 4c).

We investigate why outliers occur in Appendix E.1 and explained the case at initialization. Papyan (2019) provides explanation for this phenomenon using class clustering. Although we reproduced their results on their networks, their explanations do not apply to networks we experiment on because there is no clear class clustering for our networks. The reason is unclear but we conjecture that class clustering is only significant for large networks. The experiment results are shown in Appendix E.1.

5 Understanding Structures of Hessian

5.1 Eigenspace Overlap of Different Models

We now use the decoupling conjecture and eigenvalue correspondence to explain the structures of Hessians. Consider models trained using different random initializations with the same network structure and dataset. We observe surpisingly high top eigenspace overlap between their layer-wise Hessians, despite no obvious similarity between their parameters.



(a) conv2:LeNet with 16/25 output channels.

(b) fc1:LeNet5 with 80/100/120/150 output neuron.

Figure 5: Eigenspace overlap between differently paramaterized models.

Fig. 5 includes 5 different structural variants of LeNet5 trained on CIFAR-10. For each structural variant, 5 models are trained independently from different random initializations. We plot the average pairwise overlap between the top eigenspaces of those 5 models' layer-wise Hessians. In each figure, we vary the number of output neuron/channels ($m = 16/25$ for Fig. 5a and $m = 80/100/120/150$ for Fig. 5b). It is clear that for the same structure, the top eigenspaces of different models exhibits a highly non-trivial overlap, and the overlap peaks near m – the dimension of the layer's output.

As we observed in Section 4.3, the auto-correlation matrix $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ is approximately $\mathbb{E}[\boldsymbol{x}]\mathbb{E}[\boldsymbol{x}]^T$. Thus if the i-th eigenvector of $\mathbb{E}[\boldsymbol{M}]$ is $\boldsymbol{u}_i$, the i-th eigenvector of the layer-wise Hessian would be close to $\boldsymbol{u}_i \otimes \hat{\mathbb{E}}[\boldsymbol{x}]$, where $\hat{\mathbb{E}}[\boldsymbol{x}]$ is the normalized $\mathbb{E}[\boldsymbol{x}]$ ($\frac{\mathbb{E}[\boldsymbol{x}]}{\|\mathbb{E}[\boldsymbol{x}]\|}$). Even though the directions of $\boldsymbol{u}_i$'s can be very different for different models, at rank m these vectors always span the entire space, as a result the top-m eigenspace for layer-wise Hessian is close to $\boldsymbol{I}_m \otimes \hat{\mathbb{E}}[\boldsymbol{x}]$.

Now suppose we have 2 different models with $\hat{\mathbb{E}}[\boldsymbol{x}]_1$ and $\hat{\mathbb{E}}[\boldsymbol{x}]_2$ respectively. Their top-m eigenspaces are close to $\boldsymbol{I}_m \otimes \hat{\mathbb{E}}[\boldsymbol{x}]_1$ and $\boldsymbol{I}_m \otimes \hat{\mathbb{E}}[\boldsymbol{x}]_2$ respectively. In this case, it is easy to check that the overlap at m is approximately $(\hat{\mathbb{E}}[\boldsymbol{x}]_1^T \hat{\mathbb{E}}[\boldsymbol{x}]_2)^2$. Since $\hat{\mathbb{E}}[\boldsymbol{x}]_1$ and $\hat{\mathbb{E}}[\boldsymbol{x}]_2$ are the same for the input layer and all non-negative for other layers, the inner-product between them is large and the overlap is expected to be high at dimension m. In Appendix E.2 we give a more detailed explanation that explains why the overlap before rank-m grows linearly. We also make a more general explanation and account for some cases where this argument does not hold.

5.2 Dominating Eigenvectors of Layer-wise Hessian are Low Rank

Let $\boldsymbol{h}_i$ be the i-th eigenvector of a layer-wise Hessian. The rank of $\mathrm{Mat}(\boldsymbol{h}_i)$ can be considered as an indicator of the complexity of the eigenvector. Consider the case that $\boldsymbol{h}_i$ is one of the top

eigenvectors. From Section 5.1, we have $\boldsymbol{h}_i \approx \boldsymbol{u}_i \otimes \hat{\mathbb{E}}[\boldsymbol{x}]$. Thus, $\mathrm{Mat}(\boldsymbol{h}_i) \approx \boldsymbol{u}_i \hat{\mathbb{E}}[\boldsymbol{x}]^T$ and is approximately rank 1. Fig. 6 shows first singular values of $\mathrm{Mat}(\boldsymbol{h}_i)$ divided by its Frobenius Norm for i from 1 to 200. We can see the top eigenvectors are very close to rank 1.

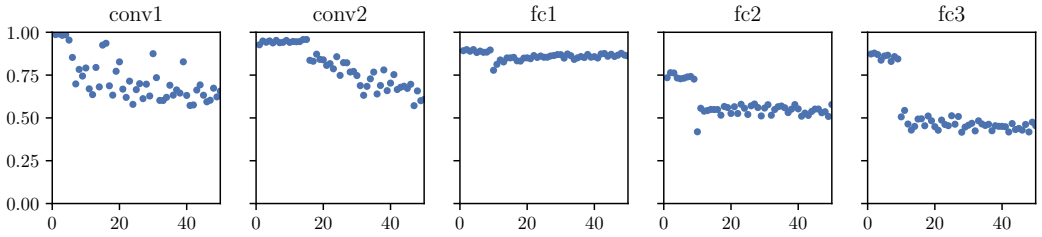


Figure 6: Ratio between top singular value and Frobenius norm. (LeNet5 on CIFAR10)

5.3 Batch Normalization and Zero-mean Input

According to our explanation, the good approximation and high overlap of top eigenspace both depend on the low rank structure of $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$. Also, the low rank structure is caused by the fact that $\mathbb{E}[\boldsymbol{x}]\mathbb{E}[\boldsymbol{x}]^T$ dominates $\boldsymbol{\Sigma}_{\boldsymbol{x}}$ in most cases. Therefore, it's natural to conjecture that models trained using Batch Normalization (BN) (Ioffe & Szegedy, 2015) will change these phenomena as for those models $\mathbb{E}[\boldsymbol{x}]$ will be zero and $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T] = \boldsymbol{\Sigma}_{\boldsymbol{x}}$. Indeed, as shown in Ghorbani et al. (2019), BN suppresses the outliers in the Hessian eigenspectrum.

We experiment on our networks with BN. The results are shown in Appendix E.3. We found that $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ is no longer close to rank 1 for models trained with BN. However, $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ still have a few large eigenvalues. In this case, all the previous structures (c outliers, high eigenspace overlap, low rank eigenvectors) become weaker. The decoupling conjecture itself also becomes less accurate. However, the approximation still gives meaningful information.

6 Tighter PAC-Bayes Bound with Hessian Information

Theorem 6.1 (PAC-Bayes Bound)**.** (McAllester, 1999)(Langford & Seeger, 2001) With the hypothesis space $\mathcal{H}$ parametrized by model parameters. For any prior distribution P in $\mathcal{H}$ that is chosen independently from the training set S, and any posterior distribution Q in $\mathcal{H}$ whose choice may inference S, with probability $1-\delta$,

$$D_{\mathrm{KL}}\left(\hat{e}(Q)||e(Q)\right) \leq \frac{D_{\mathrm{KL}}(Q||P) + \log\frac{|S|}{\delta}}{|S|-1}, \tag{11}$$

where $e(Q)$ is the expected classification error for the posterior over the underlying data distribution and $\hat{e}(Q)$ is the expected classification error for the posterior over the training set.

PAC-Bayes bound is commonly used to derive upper bounds on the generalization error. Intuitively, if one can find a posterior distribution Q that both has low loss on the training set, and is close to the prior P, then the generalization error on Q must be small. Dziugaite & Roy (2017) uses optimization techniques to find an optimal posterior in the family of Gaussians with diagonal covariance. They showed that the bound can be nonvacuous for several neural network models.

We follow Dziugaite & Roy (2017) to set the prior P to be a multi-variant Gaussian. The covariance is a multiple of identity. Thus, it is invariant with respect to the change of basis. For the posterior, when the variance in one direction is larger, the distance with the prior decreases; however this also has the risk of increasing the empirical loss over the posterior. In general, one would expect the variance to be larger along a flatter direction in the loss landscape. However, since the covariance matrix of Q is fixed to be diagonal in Dziugaite & Roy (2017), the search of optimal deviation happens in standard basis vectors which are not aligned with the local loss landscape.

Using the Kronecker factorization as in Eq. (6), we can approximate the layer-wise Hessian's eigenspace. We set Q to be a Gaussian whose covariance is diagonal in the eigenbasis of the layer-

wise Hessians. We expect the alignment of sharp and flat directions will result in a better optimized posterior Q and thus resulting in a tighter bound on classification error.

We perform the same optimization process as proposed by Dziugaite & Roy (2017). Our algorithm is called *Approx Hessian* when we fix the layer-wise Hessian eigenbasis to the one at the minimum and *Iterative Hessian* when we update the eigenbasis dynamically with the mean of the Gaussian. We used identical dataset, network structures and experiment settings as in Dziugaite & Roy (2017), with a few adjustments in hyperparameters. T-n^m and R-n^m represents network F-n^m trained on true labels and random labels, respectively. We also added T-200^2 used in Section 4. T-600_{10} and T-200^2_{10} are trained on standard MNIST while all others are trained on MNIST-2 (see Appendix B.1). The results are shown in Table 1 with a confidence of 0.965.

We also plotted the final posterior variance, $\boldsymbol{s}$ for network T-200^2_{10} in Fig. 7. For Approximated Hessian and Iterative Hessian, we can see that direction associated with larger eigenvalue has a smaller variance. This agrees with our presumption that top eigenvectors are aligned with sharper directions and should have smaller variance after optimization.

Table 1: Optimized PAC-Bayes bounds using different methods.

Experiments	T-600	T-1200	T-300^2	T-600^2	R-600	T-600_{10}	T-200^2_{10}
Test Error	0.0153	0.0161	0.0150	0.0148	0.4925	0.0180	0.0208
Vanilla	0.1540	0.1754	0.1686	0.1921	0.6046	0.2879	0.4165
Approx Hessian	0.1464	0.1726	0.1417	0.1712	**0.5653**	0.2424	0.2725
Iterative Hessian	**0.1198**	**0.1417**	**0.1249**	**0.1456**	0.5681	**0.2132**	**0.2145**

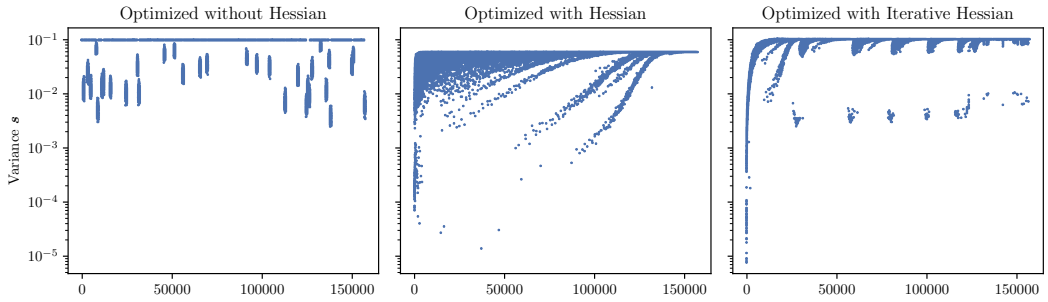


Figure 7: Optimized posterior variance, $\boldsymbol{s}$. (fc1:T-200^2, trained on MNIST), the horizontal axis is eigenbasis ordered with decreasing eigenvalues.

Detailed algorithm description and experiment results are shown in Appendix F.

7 Conclusions

In this paper we identified two new surprising structures in the dominating eigenspace of layerwise Hessians for neural networks. Specifically, the eigenspace overlap reveals a novel similarity between different models. We showed that under a decoupling conjecture, these structures can be explained by a Kronecker factorization of the layerwise Hessians. As a proof of concept, we showed that these structures can be used to find better explicit generalization bounds.

We hope this would be a starting point towards formally proving the structures in the Hessians of neural networks (in particular, why the decoupling conjecture is correct and why the output Hessian matrix is often low rank). Also, previous works suggest that dominating eigenspace of weight Hessian, which is the sharpest directions on the loss landscape, plays an important role in both generalization (Keskar et al., 2017; Jiang et al., 2020) and optimization (Gur-Ari et al., 2018). Therefore, we believe our new understanding of the common structures of this subspace can benefit both fields.

REFERENCES

Jimmy Ba, Roger B. Grosse, and James Martens. Distributed second-order optimization using kronecker-factored approximations. In *5th International Conference on Learning Representations, ICLR*, 2017.

Olivier Catoni. Pac-bayesian supervised classification: the thermodynamics of statistical learning. *arXiv preprint arXiv:0712.0248*, 2007.

Felix Dangel, Stefan Harmeling, and Philipp Hennig. Modular block-diagonal curvature approximations for feedforward architectures. In *International Conference on Artificial Intelligence and Statistics*, pp. 799–808, 2020.

Gintare Karolina Dziugaite and Daniel M. Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. In *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI*, 2017.

Stanislav Fort and Surya Ganguli. Emergent properties of the local geometry of neural loss landscapes. *arXiv preprint arXiv:1910.05929*, 2019.

Stanislav Fort, Paweł Krzysztof Nowak, Stanislaw Jastrzebski, and Srini Narayanan. Stiffness: A new perspective on generalization in neural networks. *arXiv preprint arXiv:1901.09491*, 2019.

Thomas George, César Laurent, Xavier Bouthillier, Nicolas Ballas, and Pascal Vincent. Fast approximate natural gradient descent in a kronecker factored eigenbasis. In *Advances in Neural Information Processing Systems*, pp. 9550–9560, 2018.

Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An investigation into neural net optimization via hessian eigenvalue density. In *International Conference on Machine Learning*, pp. 2232–2241, 2019.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

Noah Golmant, Zhewei Yao, Amir Gholami, Michael Mahoney, and Joseph Gonzalez. pytorch-hessian-eigentings: efficient pytorch hessian eigendecomposition, 2018. URL `https://github.com/noahgolmant/pytorch-hessian-eigenthings`.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Roger Grosse and James Martens. A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, pp. 573–582, 2016.

Guy Gur-Ari, Daniel A Roberts, and Ethan Dyer. Gradient descent happens in a tiny subspace. *arXiv preprint arXiv:1812.04754*, 2018.

Tom Heskes. On "natural" learning and pruning in multilayered perceptrons. *Neural Computation*, 12(4):881–901, 2000.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456, 2015.

Arthur Jacot, Franck Gabriel, and Clément Hongler. The asymptotic spectrum of the hessian of DNN throughout training. In *8th International Conference on Learning Representations, ICLR*, 2020.

Stanislaw Jastrzebski, Zachary Kenton, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos J. Storkey. On the relation between the sharpest directions of DNN loss and the SGD step length. In *7th International Conference on Learning Representations, ICLR*, 2019.

Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic generalization measures and where to find them. In *International Conference on Learning Representations*, 2020.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *5th International Conference on Learning Representations, ICLR*, 2017.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

John Langford and Matthias Seeger. Bounds for averaging classifiers. Technical report, 2001.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Xinyan Li, Qilong Gu, Yingxue Zhou, Tiancong Chen, and Arindam Banerjee. Hessian based analysis of sgd for deep nets: Dynamics and generalization. In *Proceedings of the 2020 SIAM International Conference on Data Mining*, pp. 190–198. SIAM, 2020.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pp. 2408–2417, 2015.

James Martens, Jimmy Ba, and Matt Johnson. Kronecker-factored curvature approximations for recurrent neural networks. In *International Conference on Learning Representations*, 2018.

David A McAllester. Some pac-bayesian theorems. *Machine Learning*, 37(3):355–363, 1999.

Kazuki Osawa, Yohei Tsuji, Yuichiro Ueno, Akira Naruse, Rio Yokota, and Satoshi Matsuoka. Large-scale distributed second-order optimization using kronecker-factored approximate curvature for deep convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 12359–12367, 2019.

Vardan Papyan. The full spectrum of deepnet hessians at scale: Dynamics with sgd training and sample size. *arXiv preprint arXiv:1811.07062*, 2018.

Vardan Papyan. Measurements of three-level hierarchical structure in the outliers in the spectrum of deepnet hessians. In *International Conference on Machine Learning*, pp. 5012–5021, 2019.

Vardan Papyan. Traces of class/cross-class structure pervade deep learning spectra. *arXiv preprint arXiv:2008.11865*, 2020.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. Technical report, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Levent Sagun, Leon Bottou, and Yann LeCun. Eigenvalues of the hessian in deep learning: Singularity and beyond. *arXiv preprint arXiv:1611.07476*, 2016.

Levent Sagun, Utku Evci, V. Ugur Güney, Yann N. Dauphin, and Léon Bottou. Empirical analysis of the hessian of over-parametrized neural networks. In *6th International Conference on Learning Representations, ICLR 2018, Workshop Track Proceedings*, 2018.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR*, 2015.

Sahil Singla, Eric Wallace, Shi Feng, and Soheil Feizi. Understanding impacts of high-order loss approximations and features in deep learning interpretation. In *International Conference on Machine Learning*, pp. 5848–5856, 2019.

Maciej Skorski. Chain rules for hessian and higher derivatives made easy by tensor calculus. *arXiv preprint arXiv:1911.13292*, 2019.

Suvrit Sra, Sebastian Nowozin, and Stephen J Wright. *Optimization for machine learning*. Mit Press, 2012.

Chaoqi Wang, Roger Grosse, Sanja Fidler, and Guodong Zhang. Eigendamage: Structured pruning in the kronecker-factored eigenbasis. In *International Conference on Machine Learning*, pp. 6566–6575, 2019.

Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, pp. 5279–5288, 2017.

Zhewei Yao, Amir Gholami, Qi Lei, Kurt Keutzer, and Michael W Mahoney. Hessian-based analysis of large batch training and robustness to adversaries. In *Advances in Neural Information Processing Systems*, pp. 4949–4959, 2018.

Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael Mahoney. Pyhessian: Neural networks through the lens of the hessian. *arXiv preprint arXiv:1912.07145*, 2019.

Guodong Zhang, Shengyang Sun, David Duvenaud, and Roger Grosse. Noisy natural gradient as variational inference. In *International Conference on Machine Learning*, pp. 5852–5861, 2018.

Wenda Zhou, Victor Veitch, Morgane Austern, Ryan P Adams, and Peter Orbanz. Non-vacuous generalization bounds at the imagenet scale: a pac-bayesian compression approach. In *International Conference on Learning Representations*, 2019.

APPENDIX ROADMAP

1. In Appendix A, we provide the detailed derivations of Hessian for fully-connected and convolutional layers.
2. In Appendix B, we give the detailed experiment setups, including the datasets, network structures, and the training settings we use.
3. In Appendix C, we explain how we compute the eigenvalues and eigenvectors of full and layer-wise Hessian numerically.
4. In Appendix D, we provide detailed experimental results that are not fully included in the main text.
5. In Appendix E, we provide additional and more general explanations of the phenomena we found.
6. In Appendix F, we give a detailed description of the PAC-Bayes bound that we optimize and the algorithm we use to optimize the bound.
7. In Appendix G, we give a brief description of the source code.

A DETAILED DERIVATIONS

A.1 DERIVATION OF HESSIAN

For an input $\boldsymbol{x}$ with label $\boldsymbol{y}$, we define the Hessian of single input loss with respect to vector $\boldsymbol{v}$ as

$$\boldsymbol{H}_\ell(\boldsymbol{v}, \boldsymbol{x}) = \nabla^2_{\boldsymbol{v}} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}) = \nabla^2_{\boldsymbol{v}} \ell(\boldsymbol{z}_{\boldsymbol{x}}, \boldsymbol{y}). \tag{12}$$

We define the Hessian of loss with respect to $\boldsymbol{v}$ for the entire training sample as

$$\boldsymbol{H}_{\mathcal{L}}(\boldsymbol{v}) = \nabla^2_{\boldsymbol{v}} \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \nabla^2_{\boldsymbol{v}} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i) = \sum_{i=1}^{N} \boldsymbol{H}_\ell(\boldsymbol{v}, \boldsymbol{x}_i) = \mathbb{E}\left[\boldsymbol{H}_\ell(\boldsymbol{v}, \boldsymbol{x})\right]. \tag{13}$$

We now derive the Hessian for a fixed input label pair $(\boldsymbol{x}, \boldsymbol{y})$. Following the definition and notations in Section 3, we also denote output as $\boldsymbol{z} = f_{\boldsymbol{\theta}}(\boldsymbol{x})$. We fix a layer p for the layer-wise Hessian. Here the layer-wise weight Hessian is $\boldsymbol{H}_\ell(\boldsymbol{w}^{(p)}, \boldsymbol{x})$. We also have the output for the layer as $\boldsymbol{z}^{(p)}$. Since $\boldsymbol{w}^{(p)}$ only appear in the layer but not the subsequent layers, we can consider $\boldsymbol{z} = f_{\boldsymbol{\theta}}(\boldsymbol{x}) = g_{\boldsymbol{\theta}}(\boldsymbol{z}^{(p)}(\boldsymbol{w}, \boldsymbol{x}))$ where $g_{\boldsymbol{\theta}}$ only contains the layers after the p-th layer and does not depend on $\boldsymbol{w}^{(p)}$. Thus, using the Hessian Chain rule (Skorski, 2019), we have

$$\boldsymbol{H}_\ell(\boldsymbol{w}^{(p)}, \boldsymbol{x}) = \left(\frac{\partial \boldsymbol{z}^{(p)}}{\partial \boldsymbol{w}^{(p)}}\right)^T \boldsymbol{H}_\ell(\boldsymbol{z}^{(p)}, \boldsymbol{x}) \left(\frac{\partial \boldsymbol{z}^{(p)}}{\partial \boldsymbol{w}^{(p)}}\right) + \sum_{i=1}^{m^{(p)}} \frac{\partial \ell(\boldsymbol{z}, \boldsymbol{y})}{\partial z_i^{(p)}} \nabla^2_{\boldsymbol{w}^{(p)}} z_i^{(p)}, \tag{14}$$

where $z_i^{(p)}$ is the ith entry of $\boldsymbol{z}^{(p)}$ and $m^{(p)}$ is the number of neurons in p-th layer (size of $\boldsymbol{z}^{(p)}$).

Since $\boldsymbol{z}^{(p)} = \boldsymbol{W}^{(p)} \boldsymbol{x}^{(p)} + \boldsymbol{b}^{(p)}$ and $\boldsymbol{w}^{(p)} = \mathrm{vec}(\boldsymbol{W}^{(p)})$ we have

$$\frac{\partial \boldsymbol{z}^{(p)}}{\partial \boldsymbol{w}^{(p)}} = \boldsymbol{I}_{m^{(p)}} \otimes \boldsymbol{x}^{(p)T}. \tag{15}$$

Since $\frac{\partial \boldsymbol{z}^{(p)}}{\partial \boldsymbol{w}^{(p)}}$ does not depend on $\boldsymbol{w}^{(p)}$,

$$\nabla^2_{\boldsymbol{w}^{(p)}} z_i^{(p)} = 0, \ \forall i. \tag{16}$$

Thus,

$$\boldsymbol{H}_\ell(\boldsymbol{w}^{(p)}, \boldsymbol{x}) = \left(\boldsymbol{I}_{m^{(p)}} \otimes \boldsymbol{x}^{(p)}\right) \boldsymbol{H}_\ell(\boldsymbol{z}^{(p)}, \boldsymbol{x}) \left(\boldsymbol{I}_{m^{(p)}} \otimes \boldsymbol{x}^{(p)T}\right). \tag{17}$$

We define $\boldsymbol{M}_{\boldsymbol{x}}^{(p)} = \boldsymbol{H}_\ell(\boldsymbol{z}^{(p)}, \boldsymbol{x})$ as in Section 3 so that

$$\boldsymbol{H}_\ell(\boldsymbol{w}^{(p)}, \boldsymbol{x}) = \left(\boldsymbol{I}_{m^{(p)}} \otimes \boldsymbol{x}^{(p)}\right) \boldsymbol{M}_{\boldsymbol{x}}^{(p)} \left(\boldsymbol{I}_{m^{(p)}} \otimes \boldsymbol{x}^{(p)T}\right) = \boldsymbol{M}_{\boldsymbol{x}}^{(p)} \otimes \boldsymbol{x}^{(p)} \boldsymbol{x}^{(p)T}. \tag{18}$$

We now look into $\boldsymbol{M}_x^{(p)} = \boldsymbol{H}_\ell(\boldsymbol{z}^{(p)}, \boldsymbol{x})$. We again have $\boldsymbol{z} = g_{\boldsymbol{\theta}}(\boldsymbol{z}^{(p)})$ and can use chain rule here,

$$\boldsymbol{H}_\ell(\boldsymbol{z}^{(p)}, \boldsymbol{x}) = \left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{z}^{(p)}}\right)^T \boldsymbol{H}_\ell(\boldsymbol{z}, \boldsymbol{x}) \left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{z}^{(p)}}\right) + \sum_{i=1}^{c} \frac{\partial \ell(\boldsymbol{z}, \boldsymbol{y})}{\partial z_i} \nabla^2_{\boldsymbol{z}^{(p)}} z_i \tag{19}$$

By letting $\boldsymbol{p} := \mathrm{softmax}(\boldsymbol{z})$ be the output confidence vector, we define the Hessian with respect to output logit $\boldsymbol{z}$ as $\boldsymbol{A}_{\boldsymbol{x}}$ and have

$$\boldsymbol{A}_{\boldsymbol{x}} := \boldsymbol{H}_\ell(\boldsymbol{z}, \boldsymbol{x}) = \nabla^2_{\boldsymbol{z}} l(\boldsymbol{z}, \boldsymbol{y}) = \mathrm{diag}(\boldsymbol{p}) - \boldsymbol{p}\boldsymbol{p}^T, \tag{20}$$

according to Singla et al. (2019).

We also define the Jacobian of $\boldsymbol{z}$ with respect to $\boldsymbol{z}^{(p)}$ (informally logit gradient for layer p) as $\boldsymbol{G}_{\boldsymbol{x}}^{(p)} := \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{z}^{(p)}}$. For FC layers with ReLUs, we can consider ReLU after the p-th layer as multiplying $\boldsymbol{z}^{(p)}$ by an indicator function $\mathbf{1}_{\boldsymbol{z}^{(p)}>0}$. To use matrix multiplication, we can turn the indicator function into a diagonal matrix and define it as $\boldsymbol{D}^{(p)}$ where

$$\boldsymbol{D}^{(p)} := \mathrm{diag}\left(\mathbf{1}_{\boldsymbol{z}^{(p)}>0}\right). \tag{21}$$

Thus, we have the input of the next layer as $\boldsymbol{x}^{(p+1)} = \boldsymbol{D}^{(p)}\boldsymbol{z}^{(p)}$. The FC layers can then be considered as a sequential matrix multiplication and we have the final output as

$$\boldsymbol{z} = \boldsymbol{W}^{(L)}\boldsymbol{D}^{(L-1)}\boldsymbol{W}^{(L-1)}\boldsymbol{D}^{(L-2)}\cdots\boldsymbol{D}^{(p)}\boldsymbol{z}^{(p)}. \tag{22}$$

Thus,

$$\boldsymbol{G}_{\boldsymbol{x}}^{(p)} = \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{z}^{(p)}} = \boldsymbol{W}^{(L)}\boldsymbol{D}^{(L-1)}\boldsymbol{W}^{(L-1)}\boldsymbol{D}^{(L-2)}\cdots\boldsymbol{D}^{(p)}. \tag{23}$$

Since $\boldsymbol{G}_{\boldsymbol{x}}^{(p)}$ is independent of $\boldsymbol{z}^{(p)}$, we have

$$\nabla^2_{\boldsymbol{z}^{(p)}} z_i = 0, \forall i. \tag{24}$$

Thus,

$$\boldsymbol{M}_{\boldsymbol{x}}^{(p)} = \boldsymbol{H}_\ell(\boldsymbol{z}^{(p)}, \boldsymbol{x}) = \boldsymbol{G}_{\boldsymbol{x}}^{(p)T}\boldsymbol{A}_{\boldsymbol{x}}\boldsymbol{G}_{\boldsymbol{x}}^{(p)}. \tag{25}$$

Moreover, loss Hessian with respect to the bias term $\boldsymbol{b}^{(p)}$ equals to that with respect to the output of that layer $\boldsymbol{z}^{(p)}$. We thus have

$$\boldsymbol{H}_\ell(\boldsymbol{b}^{(p)}, \boldsymbol{x}) = \boldsymbol{M}_{\boldsymbol{x}}^{(p)} = \boldsymbol{G}_{\boldsymbol{x}}^{(p)T}\boldsymbol{A}_{\boldsymbol{x}}\boldsymbol{G}_{\boldsymbol{x}}^{(p)}. \tag{26}$$

The Hessians of loss for the entire training sample are simply the empirical expectations of the Hessian for single input. We have the formula as the following:

$$\boldsymbol{H}_{\mathcal{L}}(\boldsymbol{w}^{(p)}) = \mathbb{E}\left[\boldsymbol{H}_\ell(\boldsymbol{w}^{(p)}, \boldsymbol{x})\right] = \mathbb{E}\left[\boldsymbol{M}_{\boldsymbol{x}}^{(p)} \otimes \boldsymbol{x}^{(p)}\boldsymbol{x}^{(p)T}\right], \tag{27}$$

$$\boldsymbol{H}_{\mathcal{L}}(\boldsymbol{b}^{(p)}) = \boldsymbol{H}_{\mathcal{L}}(\boldsymbol{z}^{(p)}) = \mathbb{E}\left[\boldsymbol{M}_{\boldsymbol{x}}^{(p)}\right] = \mathbb{E}\left[\boldsymbol{G}_{\boldsymbol{x}}^{(p)T}\boldsymbol{A}_{\boldsymbol{x}}\boldsymbol{G}_{\boldsymbol{x}}^{(p)}\right]. \tag{28}$$

Note that we can further decompose $\boldsymbol{A}_{\boldsymbol{x}} = \boldsymbol{Q}_{\boldsymbol{x}}^T\boldsymbol{Q}_{\boldsymbol{x}}$, where

$$\boldsymbol{Q}_{\boldsymbol{x}} = \mathrm{diag}\left(\sqrt{\boldsymbol{p}}\right)\left(\boldsymbol{I}_c - \mathbf{1}_c\boldsymbol{p}^T\right), \tag{29}$$

with $\mathbf{1}_c$ is a all one vector of size c, proved in Papyan (2019). We can further extend the close form expression to off diagonal blocks and the bias entries to get the full Gauss-Newton term of Hessian. Let

$$\boldsymbol{F}_{\boldsymbol{x}} = \begin{pmatrix} \boldsymbol{Q}_{\boldsymbol{x}}\boldsymbol{G}_{\boldsymbol{x}}{}^{(1)} \otimes \boldsymbol{x}^{(1)} \\ \boldsymbol{Q}_{\boldsymbol{x}}\boldsymbol{G}_{\boldsymbol{x}}{}^{(1)} \\ \vdots \\ \boldsymbol{Q}_{\boldsymbol{x}}\boldsymbol{G}_{\boldsymbol{x}}{}^{(L)} \otimes \boldsymbol{x}^{(n)} \\ \boldsymbol{Q}_{\boldsymbol{x}}\boldsymbol{G}_{\boldsymbol{x}}{}^{(L)} \end{pmatrix}. \tag{30}$$

The full Hessian is given by

$$\boldsymbol{H}_{\mathcal{L}}(\boldsymbol{\theta}) = \mathbb{E}\left[\boldsymbol{F}_{\boldsymbol{x}}^T\boldsymbol{F}_{\boldsymbol{x}}\right] + \mathbb{E}\left[\sum_{i=1}^{c} \frac{\partial \ell(\boldsymbol{z}, \boldsymbol{y})}{z_i} \nabla^2_{\boldsymbol{\theta}} z_i\right]. \tag{31}$$

A.2 APPROXIMATING WEIGHT HESSIAN OF CONVOLUTIONAL LAYERS

The approximation of weight Hessian of convolutional layer is a trivial extension from the approximation of Fisher information matrix of convolutional layer by Grosse & Martens (2016).

Consider a two dimensional convolutional layer of neural network with m input channels and n output channels. Let its input feature map $\mathbf{X}$ be of shape (n, X_1, X_2) and output feature map $\mathbf{Z}$ be of shape (m, P_1, P_2). Let its convolution kernel be of size $K_1 \times K_2$. Then the weight $\mathbf{W}$ is of shape (m, n, K_1, K_2), and the bias $\boldsymbol{b}$ is of shape (m). Let P be the number of patches slide over by the convolution kernel, we have $P = P_1 P_2$.

Follow Dangel et al. (2020), we define $\boldsymbol{Z} \in \mathbb{R}^{m \times P}$ as the reshaped matrix of $\mathbf{Z}$ and $\boldsymbol{W} \in \mathbb{R}^{m \times nK_1K_2}$ as the reshaped matrix of $\mathbf{W}$. Define $\boldsymbol{B} \in \mathbb{R}^{m \times P}$ by broadcasting $\boldsymbol{b}$ to P dimensions. Let $\boldsymbol{X} \in \mathbb{R}^{nK_1K_2 \times P}$ be the unfolded $\mathbf{X}$ with respect to the convolutional layer. The unfold operation (Paszke et al., 2019) is commonly used in computation to model convolution as matrix operations.

After the above transformation, we have the linear expression of the p-th convolutional layer similar to FC layers:

$$\boldsymbol{Z}^{(p)} = \boldsymbol{W}^{(p)} \boldsymbol{X}^{(p)} + \boldsymbol{B}^{(p)} \tag{32}$$

We still omit superscription of (p) for dimensions for simplicity. We also denote $\boldsymbol{z}^{(p)}$ as the vector form of $\boldsymbol{Z}^{(p)}$ and has size mP. Similar to fully connected layer, we have analogue of Eq. (18) for convolutional layer as

$$\boldsymbol{H}_\ell(\boldsymbol{w}^{(p)}, \boldsymbol{X}) = \left(\boldsymbol{I}_m \otimes \boldsymbol{X}^{(p)}\right) \boldsymbol{M}_{\boldsymbol{x}}^{(p)} \left(\boldsymbol{I}_m \otimes \boldsymbol{X}^{(p)T}\right), \tag{33}$$

where $\boldsymbol{M}_{\boldsymbol{x}}^{(p)} = \boldsymbol{H}_\ell(\boldsymbol{z}^{(p)}, \boldsymbol{X})$ and is a $mP \times mP$ matrix. Also, since convolutional layers can also be considered as linear operations (matrix multiplication with reshape) together with FC layers and ReLUs, Eq. (24) still holds. Thus, we still have

$$\boldsymbol{H}_\ell(\boldsymbol{z}^{(p)}, \boldsymbol{X}) = \boldsymbol{M}_{\boldsymbol{x}}^{(p)} = \boldsymbol{G}_{\boldsymbol{x}}^{(p)T} \boldsymbol{A}_{\boldsymbol{x}} \boldsymbol{G}_{\boldsymbol{x}}^{(p)}, \tag{34}$$

where $\boldsymbol{G}_{\boldsymbol{x}}^{(p)} = \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{z}^{(p)}}$ and has dimension $c \times mP$, although is cannot be further decomposed as direct multiplication of weight matrices as in the FC layers.

However, for convolutional layers, $\boldsymbol{X}^{(p)}$ is a matrix instead of a vector. Thus, we cannot make Eq. (33) into the form of a Kronecker product as in Eq. (18).

Despite this, it is still possible to have a Kronecker factorization of the weight Hessian in the form

$$\boldsymbol{H}_\ell(\boldsymbol{w}^{(p)}, \boldsymbol{X}) \approx \tilde{\boldsymbol{M}}_{\boldsymbol{x}}^{(p)} \otimes \boldsymbol{X}^{(p)} \boldsymbol{X}^{(p)T}, \tag{35}$$

using further approximation motivated by Grosse & Martens (2016). Note that $\tilde{\boldsymbol{M}}_{\boldsymbol{x}}^{(p)}$ need to have a different shape $(m \times m)$ from $\boldsymbol{M}_{\boldsymbol{x}}^{(p)}$ $(mP \times mP)$, since $\boldsymbol{H}_\ell(\boldsymbol{w}^{(p)}, \boldsymbol{X})$ is $mnK1K2 \times mnK1K2$ and $\boldsymbol{X}^{(p)} \boldsymbol{X}^{(p)T}$ is $nK1K2 \times nK1K2$.

Since we can further decompose $\boldsymbol{A}_{\boldsymbol{x}} = \boldsymbol{Q}_{\boldsymbol{x}}{}^T \boldsymbol{Q}_{\boldsymbol{x}}$, we then have

$$\boldsymbol{M}_{\boldsymbol{x}}{}^{(p)} = \boldsymbol{G}_{\boldsymbol{x}}^{(p)T} \boldsymbol{A}_{\boldsymbol{x}} \boldsymbol{G}_{\boldsymbol{x}}^{(p)} = \left(\boldsymbol{Q}_{\boldsymbol{x}} \boldsymbol{G}_{\boldsymbol{x}}{}^{(p)}\right)^T \left(\boldsymbol{Q}_{\boldsymbol{x}} \boldsymbol{G}_{\boldsymbol{x}}{}^{(p)}\right). \tag{36}$$

We define $\boldsymbol{N}_{\boldsymbol{x}}^{(p)} = \boldsymbol{Q}_{\boldsymbol{x}} \boldsymbol{G}_{\boldsymbol{x}}{}^{(p)}$. Here $\boldsymbol{Q}_{\boldsymbol{x}}$ is $c \times c$ and $\boldsymbol{G}_{\boldsymbol{x}}{}^{(p)}$ is $c \times mP$ so that $\boldsymbol{N}_{\boldsymbol{x}}^{(p)}$ is $c \times mP$. We can reshape $\boldsymbol{N}_{\boldsymbol{x}}^{(p)}$ into a $cP \times m$ matrix $\tilde{\boldsymbol{N}}_{\boldsymbol{x}}^{(p)}$. We then reduce $\boldsymbol{M}_{\boldsymbol{x}}^{(p)}$ $(mP \times mP)$ into a $m \times m$ matrix as

$$\tilde{\boldsymbol{M}}_{\boldsymbol{x}}^{(p)} = \frac{1}{P} \tilde{\boldsymbol{N}}_{\boldsymbol{x}}^{(p)T} \tilde{\boldsymbol{N}}_{\boldsymbol{x}}^{(p)}. \tag{37}$$

The scalar $\frac{1}{P}$ is a normalization factor since we squeeze a dimension of size P into size 1.

Thus, we can have similar Kronecker factorization approximation as

$$\boldsymbol{H}_{\mathcal{L}}(\boldsymbol{w}^{(p)}) = \mathbb{E}\left[\boldsymbol{H}_\ell(\boldsymbol{w}^{(p)}, \boldsymbol{X})\right] = \mathbb{E}\left[\left(\boldsymbol{I}_m \otimes \boldsymbol{X}^{(p)}\right) \boldsymbol{M}_{\boldsymbol{x}}^{(p)} \left(\boldsymbol{I}_m \otimes \boldsymbol{X}^{(p)T}\right)\right] \tag{38}$$

$$\approx \mathbb{E}\left[\tilde{\boldsymbol{M}}_{\boldsymbol{x}}^{(p)} \otimes \boldsymbol{X}^{(p)} \boldsymbol{X}^{(p)T}\right] \approx \mathbb{E}\left[\tilde{\boldsymbol{M}}_{\boldsymbol{x}}^{(p)}\right] \otimes \mathbb{E}\left[\boldsymbol{X}^{(p)} \boldsymbol{X}^{(p)T}\right]. \tag{39}$$

B DETAILED EXPERIMENT SETUP

B.1 DATASETS

We conduct experiment on CIFAR-10 (Krizhevsky, 2009) (`https://www.cs.toronto.edu/~kriz/cifar.html`) and MNIST (LeCun et al., 1998) (`http://yann.lecun.com/exdb/mnist/`). We used their default splitting of training and testing set.

To compare our work on PAC-Bayes bound with the work of Dziugaite & Roy (2017), we created a custom dataset MNIST-2 by setting the label of images 0-4 to 0 and 5-9 to 1. We also created random-labeled datasets MNIST-R and CIFAR-10-R by randomly labeling the images from the training set of MNIST and CIFAR-10.

We summarize the dataset information in Table 2

Table 2: Datasets

Dataset	# Data Points Train	Test	Input Size	# Classes	Label
CIFAR-10	50000	10000	$3 \times 32 \times 32$	10	True
CIFAR-10-R	50000	10000	$3 \times 32 \times 32$	10	Random
MNIST	60000	10000	28×28	10	True
MNIST-2	60000	10000	28×28	2	True
MNIST-R	60000	10000	28×28	10	Random

B.2 NETWORK STRUCTURES

Fully Connected Network: We used several different fully connected networks varying in the number of hidden layers and the number of neurons for each hidden layer. The output of all layers except the last layer are passed into ReLU before feeding into the subsequent layer. As described in Section 4.1, we denote a fully connected network with m hidden layers and n neurons each hidden layer by F-n^m. For networks without uniform layer width, we denote them by a sequence of numbers (e.g. for a network with three hidden layers, where the first two layers has 200 neurons each and the third has 100 neurons, we denote it as F-200^2-100). Take F-200^2 trained on MNIST as an example, it has complete structure:

Table 3: Structure of F-200^2 on MNIST

#	Name	Module	In Shape	Out Shape
1		Flatten	(28,28)	784
2	fc1	Linear(784, 200)	784	200
3		ReLU	200	200
4	fc2	Linear(200, 200)	200	200
5		ReLU	200	200
6	fc3	Linear(200, 10)	200	10
		output		

LeNets: We adopted the LeNet5 structure proposed by LeCun et al. (1998) for MNIST, and slightly modified to adapt the input of CIFAR-10 dataset. We call the structure defined in Table 5 LeNet5. Based on LeNet5, we further modified the dimension of fc1 and conv2 to create several variants for the experiment in Section 5.1. Take the model whose fc1 layer is adjusted to have 80 neurons as an example, we denote it as LeNet5-(fc1-80).

Table 4: Structure of LeNet5 on CIFAR-10

#	Name	Module	In Shape	Out Shape
1	conv1	Conv2D(3, 6, 5, 5)	(3, 32, 32)	(6, 28, 28)
2		ReLU	(6, 28, 28)	(6, 28, 28)
3	maxpool1	MaxPooling2D(2,2)	(6, 28, 28)	(6, 14, 14)
4	conv2	Conv2D(6, 16, 5, 5)	(6, 14, 14)	(16, 10, 10)
5		ReLU	(16, 10, 10)	(16, 10, 10)
6	maxpool2	MaxPooling2D(2,2)	(16, 10, 10)	(16, 5, 5)
7		Flatten	(16, 5, 5)	400
8	fc1	Linear(400, 120)	400	120
9		ReLU	120	120
10	fc2	Linear(120, 84)	120	84
11		ReLU	84	84
12	fc3	Linear(84, 10)	84	10
		output		

VGG11: To verify if our results apply to larger networks, we trained a variant of VGG11 (originally called VGG-A in the paper, but commonly refered as VGG11) proposed by Simonyan & Zisserman (2015). To adapt the structure, which is originally designed for the $3 \times 224 \times 224$ input of imageNet, to $3 \times 32 \times 32$ input of CIFAR-10, no change were made to the convolutional layers since they are insensitive to the input shape, and the final classification layer is changed to a linear layer with 512 dimension input and 10 dimension output.

Batch Normalizations: In Appendix E.3 we conducted several experiments regarding the effect of batch normalization on our results. For those experiments, we use the existing structures and add batch normalization layer for each intermediate output after it passes the ReLU module. In order for the Hessian to be well-defined, we fix the running statistics of batch normalization and treat it as a linear layer during inference. We also turn off the learnable parameters θ and β(Ioffe & Szegedy, 2015) for simplification of the model. We use X-BN to denote the model X added with batch normalization layers.

Table 5: Structure of LeNet5-BN on CIFAR-10

#	Name	Module	In Shape	Out Shape
1	conv1	Conv2D(3, 6, 5, 5)	(3, 32, 32)	(6, 28, 28)
2		ReLU	(6, 28, 28)	(6, 28, 28)
3		BatchNorm2D	(6, 28, 28)	(6, 28, 28)
4	maxpool1	MaxPooling2D(2,2)	(6, 28, 28)	(6, 14, 14)
5	conv2	Conv2D(6, 16, 5, 5)	(6, 14, 14)	(16, 10, 10)
6		ReLU	(16, 10, 10)	(16, 10, 10)
7		BatchNorm2D	(16, 10, 10)	(16, 10, 10)
8	maxpool2	MaxPooling2D(2,2)	(16, 10, 10)	(16, 5, 5)
9		Flatten	(16, 5, 5)	400
10	fc1	Linear(400, 120)	400	120
11		ReLU	120	120
12		BatchNorm1D	120	120
13	fc2	Linear(120, 84)	120	84
14		ReLU	84	84
15		BatchNorm1D	84	84
16	fc3	Linear(84, 10)	84	10
		output		

B.3 TRAINING SETTINGS

Most of the models were trained using batched stochastic gradient descent (SGD) with batch-size 128 and fixed learning rate 0.01 for 1000 epochs. No momentum and weight decay regularization term were used. The loss objective converges by the end of training, so we may assume that the final models are at local minima.

For generality we also used a training scheme with fixed learning rate at 0.001, and a training scheme with fixed learning rate at 0.01 with momentum=0.9 and 0.0005 weight-decay factor. Models trained with these settings will be explicitly stated. Otherwise we assume they were trained with the default scheme mentioned above.

Follow the default initialization scheme of PyTorch(Paszke et al., 2019), the weights of linear layers and convolutional layers are initialized using the Xavier method proposed by Glorot & Bengio (2010), and bias of each layer are initialized to be zero.

C COMPUTATION OF EIGENVECTORS/VALUES OF LAYER-WISE HESSIAN

For Hessian approximated using Kronecker factorization, we compute $\mathbb{E}[\boldsymbol{M}]$ and $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ explicitly. Let $\boldsymbol{m}$ and $\boldsymbol{v}$ be an eigenvector of $\mathbb{E}[\boldsymbol{M}]$ and $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ respectively, with corresponding eigenvalues $\lambda_{\boldsymbol{m}}$ and $\lambda_{\boldsymbol{v}}$. Since both matrices are positive semi-definite, $\boldsymbol{m} \otimes \boldsymbol{v}$ is an eigenvector of $\mathbb{E}[\boldsymbol{M}] \otimes \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ with eigenvalue $\lambda_{\boldsymbol{m}}\lambda_{\boldsymbol{v}}$. In this way, since $\mathbb{E}[\boldsymbol{M}]$ has m eigenvectors and $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ has n eigenvectors, we can approximate all mn eigenvectors for the layer-wise Hessian. All these calculation can be done directly.

However, it is almost prohibitive to calculate the true Hessian explicitly. Thus, we use numerical methods with automatic differentiation(Paszke et al., 2017) to calculate them. The packages we use is Golmant et al. (2018) and we use the Lanczos method in most of the calculations. We also use package in Yao et al. (2019) as a reference.

For layer-wise Hessian, we modified the Golmant et al. (2018) package. In particular, the package relies on the calculation of Hessian-vector product $\boldsymbol{H}\boldsymbol{v}$, where $\boldsymbol{v}$ is a vector with the same size as parameter θ. To calculate eigenvalues and eigenvectors for layer-wise Hessian at the p-th layer, we cut the $\boldsymbol{v}$ into different layers. Then, we only leave the part corresponding to weights of the p-th layer and set all other entries to 0. Note that the dimension does not change. We let the new vector be $\boldsymbol{v}^{(p)}$ and get the value of $\boldsymbol{u} = \boldsymbol{H}\boldsymbol{v}^{(p)}$ using auto differentiation. Then, we do the same operation to $\boldsymbol{u}$ and get $\boldsymbol{u}^{(p)}$.

D ADDITIONAL EXPERIMENT RESULTS

D.1 LOW RANK STRUCTURE OF AUTO-CORRELATION MATRIX $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$

We have briefly discussed about the autocorrelation matrix $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ being approximately rank 1 in Section 4.3, where $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T] \approx \mathbb{E}[\boldsymbol{x}]\mathbb{E}[\boldsymbol{x}^T]$. Here are some empirical results supporting that claim.

Define $\hat{\mathbb{E}}[\boldsymbol{x}]$ as the normalized expectation of $\boldsymbol{x}$, namely $\frac{\mathbb{E}[\boldsymbol{x}]}{\|\mathbb{E}[\boldsymbol{x}]\|}$ for fc layers, and the first left singular vector of $\mathbb{E}[\boldsymbol{X}]$ for conv layers, with size nK_1K_2, as described in Appendix A.2. Also, $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ means $\mathbb{E}[\boldsymbol{X}\boldsymbol{X}^T]$ for conv layers.

In addition, let λ_1 be the first eigenvalue of $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ and λ_2 be the second. We have

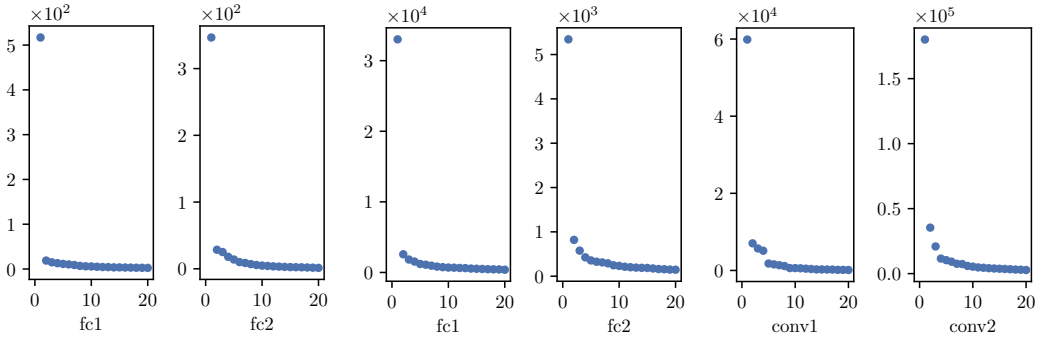
$$\frac{\lambda_1}{\lambda_2} \geq \frac{\|\mathbb{E}[\boldsymbol{x}]\mathbb{E}[\boldsymbol{x}]^T\| - \|\boldsymbol{\Sigma}_{\boldsymbol{x}}\|}{\|\boldsymbol{\Sigma}_{\boldsymbol{x}}\|} = \frac{\|\mathbb{E}[\boldsymbol{x}]\mathbb{E}[\boldsymbol{x}]^T\|}{\|\boldsymbol{\Sigma}_{\boldsymbol{x}}\|} - 1, \tag{40}$$

where $\|\cdot\|$ is the spectral norm. Thus, the spectral norm of $\mathbb{E}[\boldsymbol{x}]\mathbb{E}[\boldsymbol{x}]^T$ divided by that of $\boldsymbol{\Sigma}_{\boldsymbol{x}}$ gives a lower bound to $\frac{\lambda_1}{\lambda_2}$. In the experiment, we usually have $\frac{\lambda_1}{\lambda_2} \geq \frac{\|\mathbb{E}[\boldsymbol{x}]\mathbb{E}[\boldsymbol{x}]^T\|}{\|\boldsymbol{\Sigma}_{\boldsymbol{x}}\|}$.

The values for F-200^2 and MNIST are average of 5 different models. From Table 6 and Fig. 8 we can see that the $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ matrix is close to rank 1 for all the models (without batch normalization) we experiment on, and the principle component is approximately $\hat{\mathbb{E}}[x]$ from Table 7.

Table 6: The spectral norm of $\mathbb{E}[\boldsymbol{x}]\mathbb{E}[\boldsymbol{x}]^T$ divided by that of $\boldsymbol{\Sigma}_{\boldsymbol{x}}$

		Layer					
Dataset	Network	conv1	conv2	fc1	fc2	fc3	fc4
CIFAR-10	LeNet5	5.474	7.647	6.885	6.136	5.707	
CIFAR-10	LeNet5-(fc1-80)	5.474	7.072	6.895	7.135	5.224	
CIFAR-10	LeNet5-(conv2-25)	5.474	7.639	5.249	6.531	5.755	
CIFAR-10	VGG11	5.474	4.737				
MNIST	F-200^2			25.142	13.991	10.237	
MNIST	F-200^2-100			25.142	18.335	20.508	8.359
MNIST	T-600^2_{10}			6.857	9.104	10.706	
MNIST	T-600_{10}			6.857	6.586		
MNIST-2	T-600^2			6.857	6.011	1.735	
MNIST-2	T-300^2			6.857	6.128	1.588	
MNIST-2	T-600			6.857	4.971		
MNIST-2	T-1200			6.857	5.410		



(a) fc1:F-200^2 (MNIST) (b) fc1:LeNet5 (CIFAR-10) (c) conv1:LeNet5 (CIFAR-10)

Figure 8: Eigenspectrum of $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ for different layers in different models. All are close to rank 1.

Table 7: Absolute Inner product between $\hat{\mathbb{E}}[\boldsymbol{x}]$ and the first eigenvector of $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$

		Layer					
Dataset	Network	conv1	conv2	fc1	fc2	fc3	fc4
CIFAR-10	LeNet5	0.9999	0.9994	0.9991	0.9991	0.9988	
CIFAR-10	LeNet5-(fc1-80)	0.9999	0.9991	0.9995	0.9995	0.9978	
CIFAR-10	LeNet5-(conv2-25)	0.9999	0.9993	0.9988	0.9993	0.9989	
CIFAR-10	VGG11	0.9999	0.9986				
MNIST	F-200^2			0.9999	0.9999	0.9998	
MNIST	F-200^2-100			0.9999	1.0000	1.0000	0.9996
MNIST	F-600^2_{10}			0.9992	0.9995	0.9996	
MNIST	F-600_{10}			0.9992	0.9990		
MNIST-2	F-600^2			0.9992	0.9987	0.9993	
MNIST-2	F-300^2			0.9992	0.9988	0.9993	
MNIST-2	F-600			0.9992	0.9967		
MNIST-2	F-1200			0.9992	0.9973		

D.2 Eigenspace Overlap Between Different Models

The non trivial overlap between top eigenspaces of layer-wise Hessians is one of our interesting observations that had been discusses in Section 5.1. Here we provide more related empirical results. Some will further verify our claim in Section 5.1 and some will appear to be challenge that. Both results will be explained discussed more extensively in Appendix E.

Overlap preserved when varying hyper-parameters: We first verify that the overlap also exists for a set of models trained with the different hyper-parameters. Using the LeNet5 (defined in Table 5) as the network structure. We train 6 models using the default training scheme (SGD, lr=0.01, momentum=0), 5 models using a smaller learning rate (SGD, lr=0.001, momentum=0), and 5 models using a combination of optimization tricks (SGD, lr=0.01, momentum=0.9, weight decay=0.0005).

Given this set of 16 models, we compute the pairwise eigenspace overlap of their layer-wise Hessians (120 pairs in total), and plot their average in Fig. 9. As we can see, the pattern of overlap is clearly preserved, and the position of the peak roughly agrees with the output dimension m, demonstrating that the phenomenon is caused by a common structure instead of similarities in training process.

However, note that for layer fc3, we are not observing a linear growth starting from 0 like the other layers. This phenomenon can be explained by the lack of neuron arbitrary permutation, which will be discussed along with the reason for the linear growth pattern for other layers in Appendix E.2.

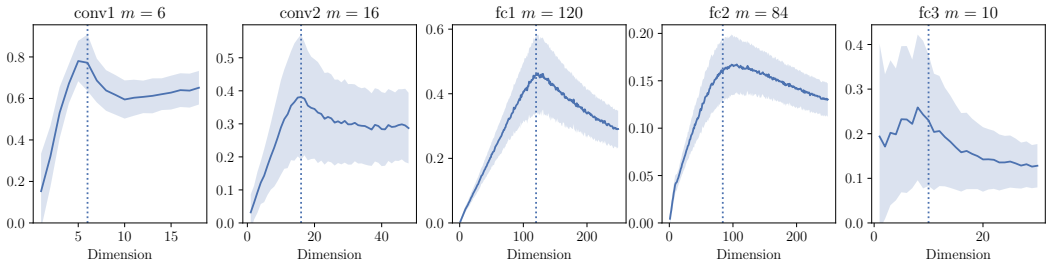


Figure 9: Eigenspace overlap of different models of LeNet5 trained with different hyper parameters.

Overlap may exhibits a early peak for some layers: For the eigenspace overlap plot of F-200^2 (Fig. 10), we see that the overlap for fc2 is significantly lower than the other layers, and there exists a small peak near dimension $k = 10$. This is because the top hessian eigenspace is not completely spanned by $\mathbb{E}[\boldsymbol{x}]$. This phenomenon will also be elaborated in Appendix E.2 with the help of correspondence matrices.

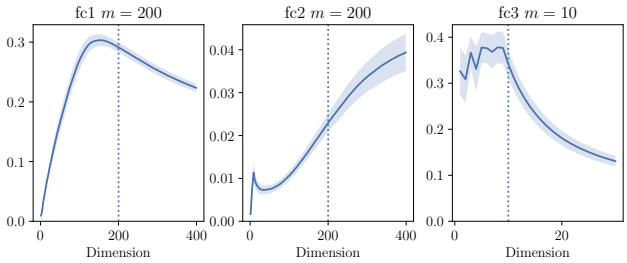
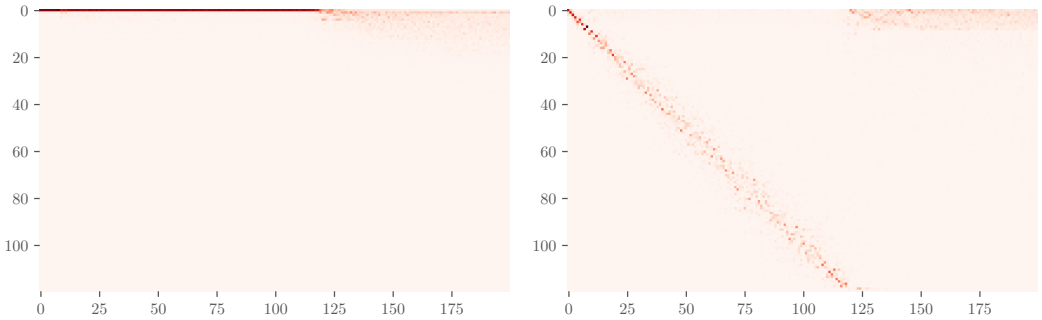


Figure 10: Eigenspace overlap of different models of F-200^2.

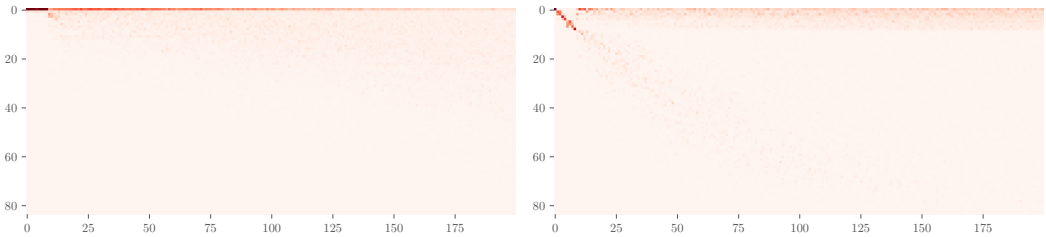
D.3 Eigenvector Correspondence

Here we present the correspondence matrix for fc1, fc2, conv1, and conv2 layer of LeNet5. The top eigenvectors for all layers shows a strong correlation with the first eigenvector of $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ (which is approximately $\hat{\mathbb{E}}[\boldsymbol{x}]$). However, the diagonal pattern in the correspondence matrix with $\mathbb{E}[\boldsymbol{M}]$ for fc2 is not as clear as the one for fc1.



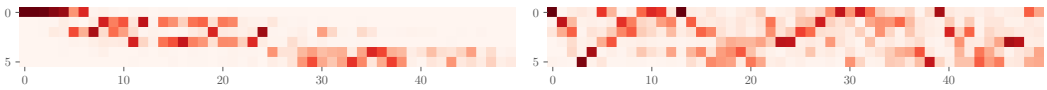
(a) Correspondence with $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$. (b) Correspondence with $\mathbb{E}[\boldsymbol{M}]$.

Figure 11: Eigenvector Correspondence for fc1:LeNet5. (m=120)



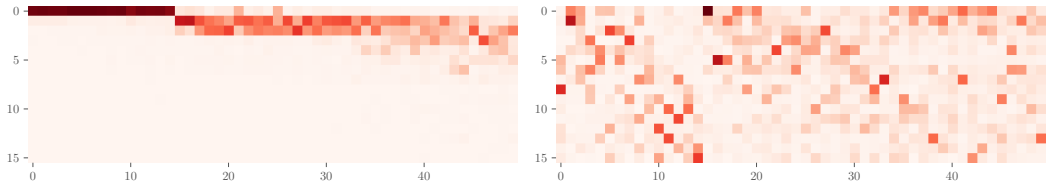
(a) Correspondence with $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$. (b) Correspondence with $\mathbb{E}[\boldsymbol{M}]$.

Figure 12: Eigenvector Correspondence for fc2:LeNet5. (m=84)



(a) Correspondence with $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$. (b) Correspondence with $\mathbb{E}[\boldsymbol{M}]$.

Figure 13: Eigenvector Correspondence for conv1:LeNet5. (m=6)



(a) Correspondence with $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$.

(b) Correspondence with $\mathbb{E}[\boldsymbol{M}]$.

Figure 14: Eigenvector Correspondence for conv2:LeNet5. (m=16)

For VGG11 we also observe a strong correlation with the first eigenvector of $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$. However the phenomenon is not as strong as the ones exhibited by LeNet5.

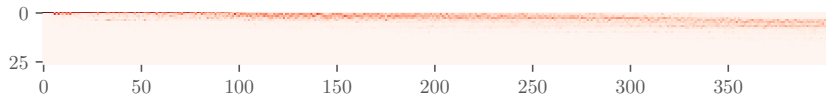


Figure 15: Eigenvector Correspondence with $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ for conv1:VGG11. (m=64)

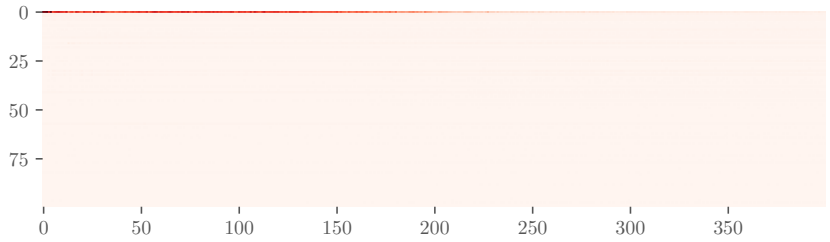


Figure 16: Eigenvector Correspondence with $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ for conv2:VGG11. (m=128)

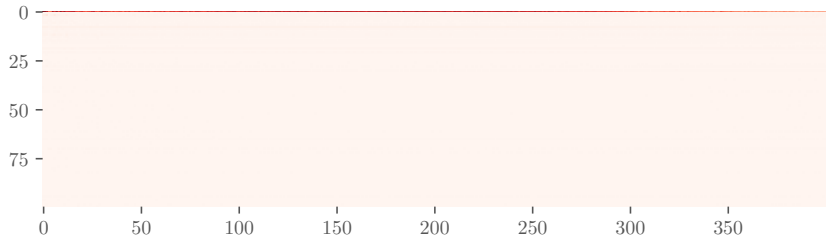


Figure 17: Eigenvector Correspondence with $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ for conv3:VGG11. (m=256)

D.4 Structure of $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ and $\mathbb{E}[\boldsymbol{M}]$ During Training

We observed the pattern of $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ matrix and $\mathbb{E}[\boldsymbol{M}]$ matrix along the training trajectory (Fig. 18, Fig. 19). It shows that $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ is always approximately rank 1, and $\mathbb{E}[\boldsymbol{M}]$ always have around c large eigenvalues. According to our analysis, since the nontrivial eigenspace overlap is likely to be a consequence of a approximately rank 1 $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$, we would conjecture that the overlap phenomenon is likely to happen on the training trajectory as well.

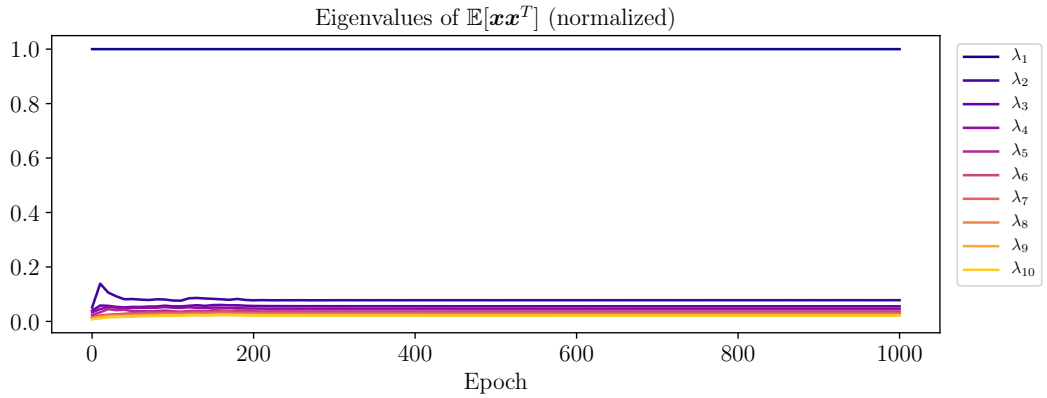


Figure 18: Top eigenvalues of $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ along training trajectory. (fc1:LeNet5)

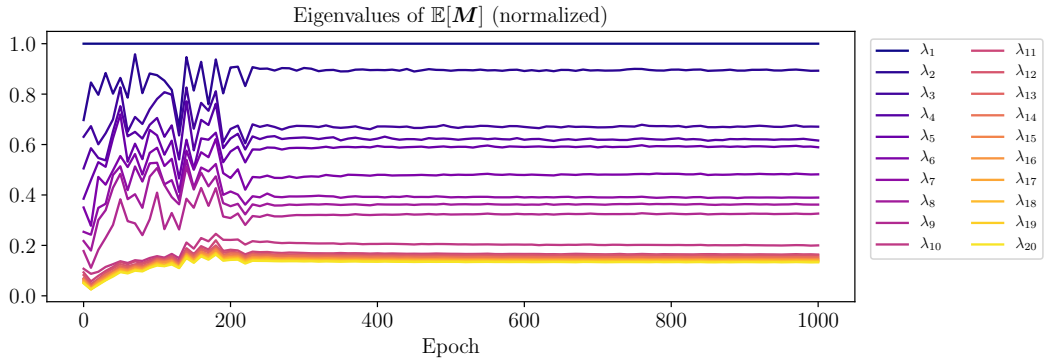


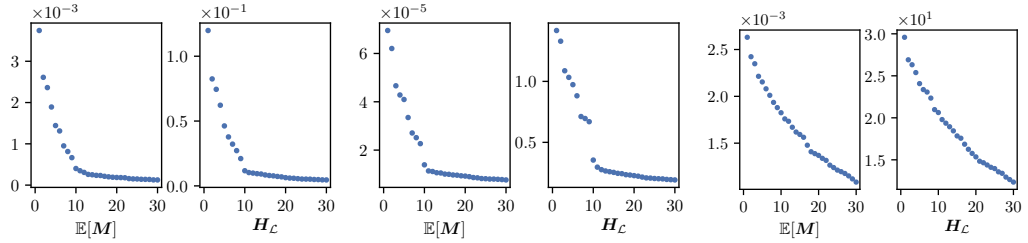
Figure 19: Top eigenvalues of $\mathbb{E}[\boldsymbol{M}]$ along training trajectory. (fc1:LeNet5)

E ADDITIONAL EXPLANATIONS

E.1 STRUCTURE OF $\mathbb{E}[\boldsymbol{M}]$ AND OUTLIERS IN HESSIAN EIGENSPECTRUM

One characteristic of Hessian that has been mentioned by many is the outliers in the spectrum of eigenvalues. Sagun et al. (2018) suggests that there is a gap in Hessian eigenvalue distribution around the number of classes c in most cases, where $c = 10$ in our case. A popular theory for such gap is the clustering of gradient within the same class (Fort & Ganguli, 2019; Fort et al., 2019). Papyan (2019) attempted further explanation for the c outliers using class clustering.

Fig. 20 is the same figure as Fig. 4 in the main text. We have shown that the eigenspectrum of $\mathbb{E}[\boldsymbol{M}]$ and the layer-wise Hessian $\boldsymbol{H}_{\mathcal{L}}(\boldsymbol{w}^{(p)})$ is similar when $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ is close to rank 1. In this section, we investigate why there are nontrivial outliers in the spectrum of $\mathbb{E}[\boldsymbol{M}]$ and $\boldsymbol{H}_{\mathcal{L}}(\boldsymbol{w}^{(p)})$.



(a) fc1:LeNet5 at initialization. (CIFAR-10)

(b) fc1:LeNet5 at minimum. (CIFAR-10)

(c) fc1:LeNet5 at minimum. (CIFAR-10 Random Labeled)

Figure 20: Eigenspectrum of $\mathbb{E}[\boldsymbol{M}]$ and $\boldsymbol{H}_{\mathcal{L}}(\boldsymbol{w}^{(p)})$.

Papyan (2019) provides explanation for the gap in eigenspectrum using class clustering. Following their methods, we use t-SNE to visualize the clustering of $\boldsymbol{\Delta}$ and $\boldsymbol{\Gamma}$. $\boldsymbol{\Delta}$ has the same definition as in their paper, except it is for the layer-wise Hessian. Fix a layer p, we have

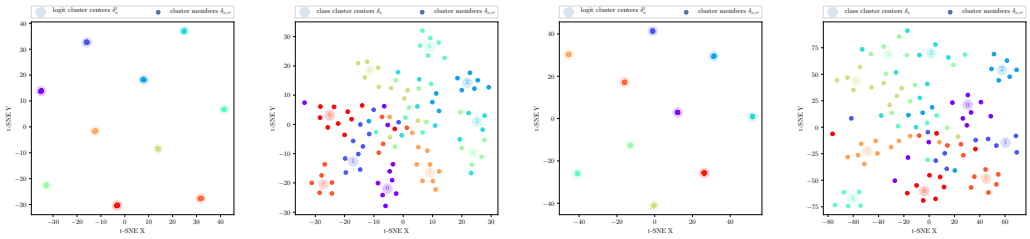
$$\boldsymbol{\Delta}_{i,c}^T = \boldsymbol{Q}_{\boldsymbol{x}_{i,c}} \frac{\partial \boldsymbol{z}_{i,c}}{\partial \boldsymbol{w}^{(p)}}, \tag{41}$$

where $\boldsymbol{Q}_{\boldsymbol{x}}$ is defined in Eq. (29), c is the class (label) of $\boldsymbol{x}$, and i is the index inside the class. The layer-wise Hessian is thus $\boldsymbol{H}_{\mathcal{L}}(\boldsymbol{w}^{(p)}) = \mathbb{E}[\boldsymbol{\Delta}_{i,c}\boldsymbol{\Delta}_{i,c}^T]$.

We then define $\boldsymbol{\Gamma}$ for $\mathbb{E}[\boldsymbol{M}]$ similarly as

$$\boldsymbol{\Gamma}_{i,c}^T = \boldsymbol{Q}_{\boldsymbol{x}_{i,c}} \frac{\partial \boldsymbol{z}_{i,c}}{\partial \boldsymbol{z}_{i,c}^{(p)}} = \boldsymbol{Q}_{\boldsymbol{x}_{i,c}} \boldsymbol{G}_{\boldsymbol{x}_{i,c}}, \tag{42}$$

so that $\mathbb{E}[\boldsymbol{M}] = \mathbb{E}[\boldsymbol{\Gamma}_{i,c}\boldsymbol{\Gamma}_{i,c}^T]$.



(a) Clustering of $\boldsymbol{\Delta}$ with respect to logit at initialization.

(b) Weak clustering of $\boldsymbol{\Delta}$ with respect to class at minimum.

(c) Clustering of $\boldsymbol{\Gamma}$ with respect to logit at initialization.

(d) Weak clustering of $\boldsymbol{\Gamma}$ with respect to class at minimum.

Figure 21: Logit clustering behavior of $\boldsymbol{\Delta}$ and $\boldsymbol{\Gamma}$ at initialization and minimum. (fc1:LeNet5)

Although we reproduced their results on their networks, there is no clear class clustering for both $\boldsymbol{H}_{\mathcal{L}}(\boldsymbol{w}^{(p)})$ and $\mathbb{E}[\boldsymbol{M}]$ at the Minimum for networks we experiment on, as shown in Fig. 21b and Fig. 21d. The reason is unclear but we conjecture that class clustering is only significant for very large networks.

The outliers at initialization, however, are easier to explain. Similar to Papyan (2019) suggests for full Hessian $\boldsymbol{H}_{\mathcal{L}}(\boldsymbol{\theta})$, we observe logit clustering in both $\boldsymbol{H}_{\mathcal{L}}(\boldsymbol{w}^{(p)})$ and $\mathbb{E}[\boldsymbol{M}]$, as shown in Fig. 21a and Fig. 21c. Since there are 10 logits, we would expect there are around 10 outliers at the initialization. This agrees with Fig. 20a.

E.2 Eigenspace Overlap of Different Models

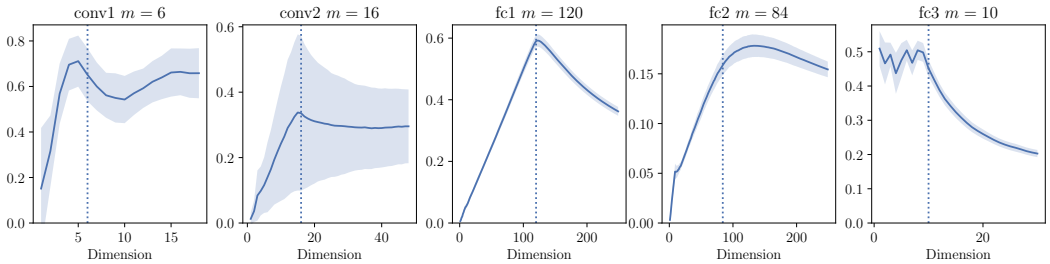


Figure 22: Eigenspace overlap of different models of LeNet5

Fig. 22 shows the average pairwise overlap between the top eigenspaces of the layer-wise Hessian of 5 different LeNet5 models. Together with Fig. 5, we can see that our approximation and explanation stated in Section 5.1 is approximately correct but may not be so accurate for some layers.

We now present a more general explanation which also explains why the overlap before rank-m grows linearly. We also explain some exceptional cases like the layer fc3 and possible discrepancies of our approximation.

Let $\boldsymbol{h}_i$ be the i-th eigenvector of the layer-wise Hessian $\boldsymbol{H}_{\mathcal{L}}(\boldsymbol{w}^{(p)})$. Following the explanation in Section 5.1, for $i \leq m$, we can approximate the $\boldsymbol{h}_i$ as $\boldsymbol{u}_i \otimes \hat{\mathbb{E}}[\boldsymbol{x}]$, where $\boldsymbol{u}_i$ is the ith eigenvector of $\mathbb{E}[\boldsymbol{M}]$. If this approximation is reasonably accurate, the eigenspace overlap pattern is explained. However, this is actually not a necessary condition for this pattern and we can loose this requirement while still explaining the phenomenon.

According to Fig. 3 and Appendix D.3, for the top m eigenvectors of layer-wise Hessian, we can see that the approximation is usually more accurate for the $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ part than the $\mathbb{E}[\boldsymbol{M}]$ part and $\boldsymbol{h}_i$ usually have a high correspondance with the top eigenvector of $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$. Indeed, this is the only condition we need. We can then have this theorem, with $\operatorname{Corr}(\boldsymbol{t}, \boldsymbol{h}_i) = \| \operatorname{Mat}(\boldsymbol{h}_i)\boldsymbol{t}\|^2$ as in Section 4.2.

Theorem E.1. Let there be 2 different models with the same network structure and dataset. Fix a layer p which is not the last layer. Let $\boldsymbol{h}_i$ be the ith eigenvector of the layer-wise Hessian for the 1st model $\boldsymbol{H}_{\mathcal{L}}(\boldsymbol{w}^{(p)})_1$, and $\boldsymbol{g}_i$ be that of the 2nd model $\boldsymbol{H}_{\mathcal{L}}(\boldsymbol{w}^{(p)})_2$. Let $\boldsymbol{t}_1, \boldsymbol{t}_2$ be 2 vectors in $\mathbb{R}^n$ (the dimension of $\mathbb{E}[\boldsymbol{x}]$).

When $\operatorname{Corr}(\boldsymbol{t}_1, \boldsymbol{h}_i) \approx 1$ and $\operatorname{Corr}(\boldsymbol{t}_2, \boldsymbol{g}_i) \approx 1$ for all $i \leq m$ holds for 2 models, the overlap of top eigenspace between $\boldsymbol{H}_{\mathcal{L}}(\boldsymbol{w}^{(p)})_1$ and $\boldsymbol{H}_{\mathcal{L}}(\boldsymbol{w}^{(p)})_2$ will be approximately $\frac{k}{m}(\boldsymbol{t}_1 \cdot \boldsymbol{t}_2)^2$. It will show a linear growth before dimension m. The peak at m is high if the squared dot product $(\boldsymbol{t}_1 \cdot \boldsymbol{t}_2)^2$ is large. For the last layer satisfying these conditions, the overlap will stay high before dimension m and be approximately $(\boldsymbol{t}_1 \cdot \boldsymbol{t}_2)^2$.

Proof. Consider the 1st model. For $i \leq m$, since $\operatorname{Corr}(\boldsymbol{t}_1, \boldsymbol{h}_i) \approx 1$, we have $\operatorname{Mat}(\boldsymbol{h}_i) \approx \boldsymbol{r}_i^T \boldsymbol{t}_1$ for some $\boldsymbol{r}_i \in \mathbb{R}^m$. Thus, $\boldsymbol{h}_i \approx \boldsymbol{r}_i \otimes \boldsymbol{t}_1$. Thus, for any $k \leq m$, we can approximate its top k eigenspace as $\boldsymbol{U}_k \otimes \boldsymbol{t}_1$, where $\boldsymbol{U}_k$ has columns $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_k$. Similarly, we can approximate $\boldsymbol{g}_i \approx \boldsymbol{s}_i \otimes \boldsymbol{t}_2$ and the top k eigenspace of the 2nd model as $\boldsymbol{V}_k \otimes \boldsymbol{t}_2$, where $\boldsymbol{V}_k$ has columns $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k$.

The eigenspace overlap of the 2 models at dimension k is thus

$$\text{Overlap}\left(\boldsymbol{U}_k \otimes \boldsymbol{t}_1, \boldsymbol{V}_k \otimes \boldsymbol{t}_2\right) = \frac{1}{k}\left\|\boldsymbol{U}_k^T \boldsymbol{V}_k \otimes \boldsymbol{t}_1^T \boldsymbol{t}_2\right\|_F^2 = \left(\boldsymbol{t}_1 \cdot \boldsymbol{t}_2\right)^2 \text{Overlap}\left(\boldsymbol{U}_k, \boldsymbol{V}_k\right). \tag{43}$$

Moreover, $\boldsymbol{r}_i, \boldsymbol{s}_i \in \mathbb{R}^n$, the space corresponding to the neurons. Note that output neurons (channels for convolutional layers) can be permuted to give equivalent models while changing eigenvectors. For $\boldsymbol{h}_i \approx \boldsymbol{r}_i \otimes \boldsymbol{t}_1$, permuting neurons will permute entries in $\boldsymbol{r}_i$. Thus, it is reasonable to assume that $\boldsymbol{r}_i$ and $\boldsymbol{s}_i$ are not correlated and thus have an expected inner product of $\frac{1}{\sqrt{m}}$.

It follows from Definition 4.1 that $\mathbb{E}[\text{Overlap}(\boldsymbol{U}_k, \boldsymbol{V}_k)] = \sum_{i=1}^{k} \mathbb{E}[(\boldsymbol{r}_i \cdot \boldsymbol{s}_i)^2] = k(\frac{1}{m}) = \frac{k}{m}$ and thus the eigenspace overlap of at dimension k would be approximately $\frac{k}{m}(\boldsymbol{t}_1 \cdot \boldsymbol{t}_2)^2$. This explains the peak at dimension m and the linear growth before it.

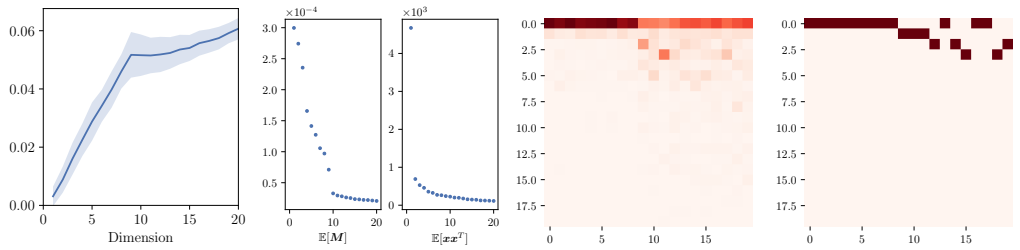
Note that layer fc3 as shown in Fig. 22 does not have a linear growth before dimension m but still have a peak at dimension m. This is because it is the last layer so that the neurons corresponds to classes. Thus, neurons cannot be permuted. In this case, the overlap will be approximately $(\boldsymbol{t}_1 \cdot \boldsymbol{t}_2)^2$ for all dimension $k \leq m$. □

From our results in Section 4.3 and Appendix D.1, we can often take $\hat{\mathbb{E}}[\boldsymbol{x}]_1$ and $\hat{\mathbb{E}}[\boldsymbol{x}_2]$ as $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$. When $k = m$, the overlap is approximately $(\hat{\mathbb{E}}[\boldsymbol{x}]_1 \cdot \hat{\mathbb{E}}[\boldsymbol{x}]_2)^2$. Since $\hat{\mathbb{E}}[\boldsymbol{x}]_1$ and $\hat{\mathbb{E}}[\boldsymbol{x}]_2$ are the same for the input layer, the overlap is expected to be very high at dimension m. For other layers, $\boldsymbol{x}$ are output of ReLU and thus non-negative. In this case, 2 non-negative vectors $\hat{\mathbb{E}}[\boldsymbol{x}]_1$ and $\hat{\mathbb{E}}[\boldsymbol{x}]_2$ should still have relatively large dot product and thus the overlap is also expected to be high.

Then, consider the $(m+1)$th eigenvector of the first model. Since top m eigenvectors span the subspace $\boldsymbol{I}_m \otimes \hat{\mathbb{E}}[\boldsymbol{x}]_1$, it will be orthogonal to this space. It will also have low overlap with $\boldsymbol{I}_m \otimes \hat{\mathbb{E}}[\boldsymbol{x}]_2$ since $(\hat{\mathbb{E}}[\boldsymbol{x}]_1 \cdot \hat{\mathbb{E}}[\boldsymbol{x}]_2)^2$ is large. This explains the immediate drop in eigenspace overlap at dimension $m+1$.

The conditions in Theorem E.1 is often satisfied in our experiments but not always. For example, we find fc2 in LeNet5 has a low peak at dimension obviously larger than m as shown in Fig. 22. It also does not show a linear growth before dimension m.

We can explain this case by looking at the eigenvector correspondence matrices shown in Fig. 23. The correspondence between eigenvectors of layer-wise Hessian and the top eigenvector of $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ is only close to 1 for the top 9 eigenvectors. Thus, Theorem E.1 does not apply here. However, we can observe a small peak at dimension 9 and linear growth before it, as shown in Fig. 23a. This is because our approximation can still be applied before dimension 9. Demonstrating that eigenspace overlap of different models can be predicted using eigenvector correspondence matrices.



(a) Eigenspace overlap (zoom in) (b) Eigenspectrum of $\mathbb{E}[\boldsymbol{M}]$ and $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ (c) True Hessian with $E[\boldsymbol{x}\boldsymbol{x}^T]$ (d) Approximated Hessian with $E[\boldsymbol{x}\boldsymbol{x}^T]$

Figure 23: Eigenspace overlap, Eigenspectrum, and Eigenvector correspondance matrices for fc2:LeNet5

In addition, Fig. 23d shows the correspondance matrix for the approximated Hessian using Kronekecker factorization $\mathbb{E}[\boldsymbol{M}] \otimes \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$. Let $\boldsymbol{u}_i$ be that of $\mathbb{E}[\boldsymbol{M}]$ with corresponding eigenvalue λ_i. Let $\boldsymbol{v}_i$ be the ith eigenvector of $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ with corresponding eigenvalue μ_i. This approximation shows that the top 9 eigenvectors can be approximated as $\boldsymbol{u}_i \otimes \boldsymbol{v}_1$ for some $\boldsymbol{u}_i$ but the 10th eigenvector cannot. Since the eigenvectors are ranked according to the magnitude of their corresponding

eigenvalues, which are approximated using products of eigenvalues of $\mathbb{E}[\boldsymbol{M}]$ and $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$. This is equivalent to having $\lambda_{10}\mu_1 < \lambda_1\mu_2$. From Fig. 23b, we can see that there is a significant gap between λ_9 and λ_{10}, suggesting why we have this inequality.

This further shows Kronecker factorizatoin can be used to predict when our conditions in Theorem E.1 fails and also predict the condition can be satisfied up to which dimension.

E.3 Batch Normalization and Zero-mean Input

In this section, we show the results on networks with using Batch normalization (BN) (Ioffe & Szegedy, 2015). For layers after BN, we have $\mathbb{E}[\boldsymbol{x}] \approx 0$ so that $\mathbb{E}[\boldsymbol{x}]\mathbb{E}[\boldsymbol{x}]^T$ no longer dominates $\boldsymbol{\Sigma}_{\boldsymbol{x}}$ and the low rank structure of $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ should disappear. Thus, we can further expect that the overlap between top eigenspace of layer-wise Hessian among different models will not have a peak.

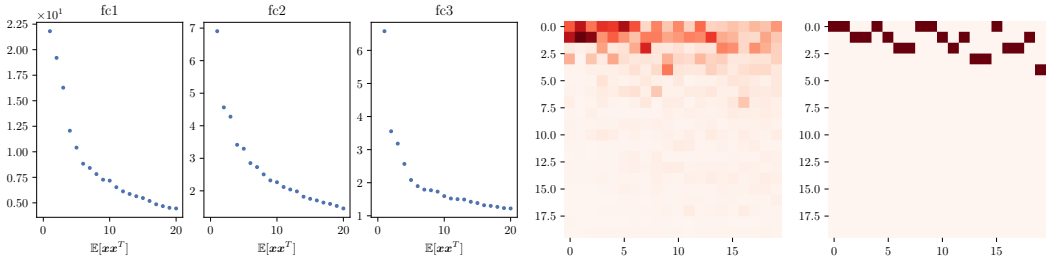
Table 8 shows the same experiments done in Table 7 (Inner product) and Table 6 (Spectral ratio). Inner product is the absolute value of inner product between $\hat{\mathbb{E}}[\boldsymbol{x}]$ and the first eigenvector of $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$. Spectral ratio is the ratio of spectral norms between $\mathbb{E}[\boldsymbol{x}]\mathbb{E}[\boldsymbol{x}]^T$ and $\boldsymbol{\Sigma}_{\boldsymbol{x}}$. The values for each network are the average of 3 different models. It is clear that the high inner product and large spectral ratio both do not hold here, except for the first layer where there is no normalization applied. Note that we had channel-wise normalization (zero-mean for each channel but not zero-mean for $\boldsymbol{x}$) for conv1 in LeNet5 so that the spectral ratio is also small.

Table 8: Structure of $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ for BN networks

			Layer				
Metric	Dataset	Network	conv1	conv2	fc1	fc2	fc3
Inner product	CIFAR-10	LeNet5-BN	0.9675	0.5179	0.0837	0.1805	0.2107
Inner product	MNIST	F-200^2-BN			0.9999	0.2414	0.2677
Spectral ratio	CIFAR-10	LeNet5-BN	0.020	0.045	0.016	0.019	0.013
Spectral ratio	MNIST	F-200^2-BN			25.142	0.010	0.011

Fig. 24a shows that $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ is no longer close to rank 1 when having BN. This is as expected. However, $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ still has a few large eigenvalues.

Fig. 24b shows the eigenvector correspondance matrix of True Hessian with $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ for fc1:LeNet5. Because $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ is no longer close to rank 1, only very few eigenvectors of the layer-wise Hessian will have high correspondance with the top eigenvector of $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$, as expected. This directly leads to the disappearance of peak in top eigenspace overlap of different models, as shown in Fig. 25. The peak still exists in conv1 because BN is not applied to the input.



(a) Eigenspectrum for $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ (b) True Hessian (fc1) (c) Approx Hessian (fc1)

Figure 24: Eigenspectrum and Eigenvector correspondance matrices with $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ for LeNet5-BN.

Comparing Fig. 24b and Fig. 24c, we can see that the Kronecker factorization still gives a reasonable approximation for the eigenvector correspondance matrix with $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$, although worse than the cases without BN (Fig. 3).

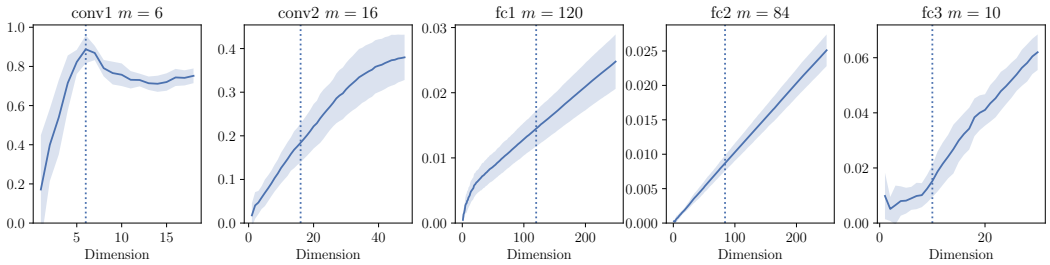
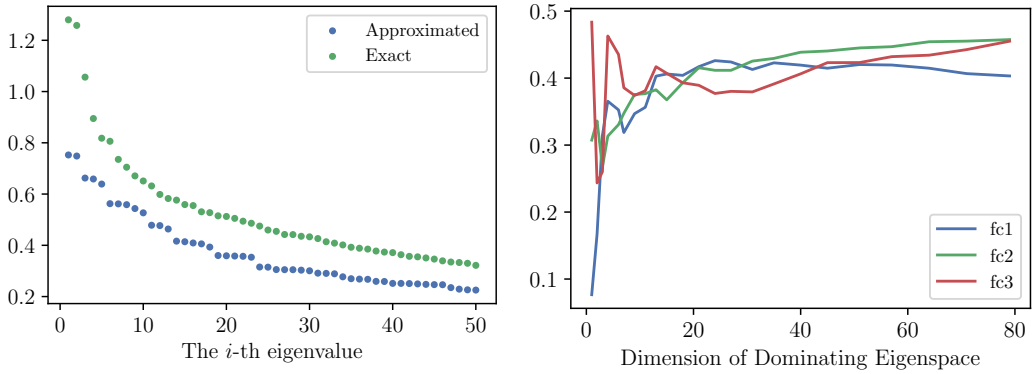


Figure 25: Eigenspapce overlap of different models of LeNet5-BN.

Fig. 26 compare the eigenvalues and top eigenspaces of the approximated Hessian and the true Hessian for LeNet5 with BN. The approximation using Kronecker factorization is also worse than the case without BN (Fig. 2). However, the approximation still gives meaningful information as the overlap of top eigenspace is still highly nontrivial.



(a) Top eigenvalues of approximated and exact layer-wise Hessian for fc2.

(b) Top eigenspace overlap between approximated and true layer-wise Hessian.

Figure 26: Comparison between the true and approximated layer-wise Hessians for LeNet5-BN.

F COMPUTING PAC-BAYES BOUNDS WITH HESSIAN APPROXIMATION

Given a model parameterized with θ and an input-label pair $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^d \times \mathbb{R}^c$, the classification error of θ over the input sample $\boldsymbol{x}$ is $\breve{l}(\theta, \boldsymbol{x}) := \mathbf{1}[\arg\max f_\theta(\boldsymbol{x}) = \arg\max \boldsymbol{y}]$. With the underlying data distribution D and training set S i.i.d. sampled from D, we define

$$e(\theta) := \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim D}[\breve{l}(\theta, \boldsymbol{x})], \qquad \hat{e}(\theta) := \frac{1}{N}\sum_{i=1}^{N}[\breve{l}(\theta, \boldsymbol{x}_i)] \tag{44}$$

as the expected and empirical classification error of θ, respectively. We define the measurable hypothesis space of parameters $\mathcal{H} := \mathbb{R}^P$. For any probabilistic measure P in $\mathcal{H}$, let $e(P) = \mathbb{E}_{\theta\sim P} e(\theta)$, $\hat{e}(P) = \mathbb{E}_{\theta\sim P}\hat{e}(\theta)$, and $\breve{e}(P) = \mathbb{E}_{\theta\sim P}\mathcal{L}(\theta)$. Here $\breve{e}(P)$ serves as a differentiable convex surrogate of $\hat{e}(P)$.

Theorem F.1 (Pac-Bayes Bound)**.** (McAllester, 1999)(Langford & Seeger, 2001) For any prior distribution P in $\mathcal{H}$ that is chosen independently from the training set S, and any posterior distribution Q in $\mathcal{H}$ whose choice may inference S, with probability $1-\delta$,

$$D_{\mathrm{KL}}\left(\hat{e}(Q) \| e(Q)\right) \leq \frac{D_{\mathrm{KL}}(Q\|P) + \log\frac{|S|}{\delta}}{|S|-1}. \tag{45}$$

Fix some constant $b, c \geq 0$ and $\theta_0 \in \mathcal{H}$ as a random initialization, Dziugaite & Roy (2017) shows that when setting $Q = \mathcal{N}(\boldsymbol{w}, \mathrm{diag}(\boldsymbol{s}))$, $P = \mathcal{N}(\theta_0, \lambda \boldsymbol{I}_P)$, where $\boldsymbol{w}, \boldsymbol{s} \in \mathcal{H}$ and $\lambda = c\exp(-j/b)$ for some $j \in \mathbb{N}$, and solve the optimization problem

$$\min_{\boldsymbol{w},\boldsymbol{s},\lambda} \breve{e}(Q) + \sqrt{\frac{D_{\mathrm{KL}}(Q\|P) + \log\frac{|S|}{\delta}}{2(|S|-1)}}, \tag{46}$$

with initialization $\boldsymbol{w} = \theta$, $\boldsymbol{s} = \theta^2$, one can achieved a nonvacous PAC-Bayes bound by Eq. (45).

In order to avoid discrete optimization for $j \in \mathcal{N}$, Dziugaite & Roy (2017) uses the B_{RE} term to replace the bound in Eq. (45). The B_{RE} term is defined as

$$B_{\mathrm{RE}}(\boldsymbol{w}, \boldsymbol{s}, \lambda; \delta) = \frac{D_{\mathrm{KL}}(P\|Q) + 2\log(b\log\frac{c}{\lambda}) + \log\frac{\pi^2|S|}{6\delta}}{|S|-1}, \tag{47}$$

where $Q = \mathcal{N}(\boldsymbol{w}, \mathrm{diag}(\boldsymbol{s}))$, $P = \mathcal{N}(\theta_0, \lambda\boldsymbol{I}_P)$. The optimization goal actually used in the implementation is thus

$$\min_{\boldsymbol{w}\in\mathbb{R}^P, \boldsymbol{s}\in\mathbb{R}^P_+, \lambda\in(0,c)} \breve{e}(Q) + \sqrt{\frac{1}{2}B_{\mathrm{RE}}(\boldsymbol{w}, \boldsymbol{s}, \lambda; \delta)}. \tag{48}$$

Algorithm 1 shows the algorithm for Iterative Hessian PAC-Bayes Optimization. If we set $\eta = T$, the algorithm will be come Approximate Hessian PAC-Bayes Optimization. It is based on Algorithm 1 in Dziugaite & Roy (2017). The initialization of $\boldsymbol{w}$ is different from Dziugaite & Roy (2017) because we believe what they wrote, $\mathrm{abs}(\boldsymbol{w})$ is a typo and $\log[\mathrm{abs}(\boldsymbol{w})]$ is what they actually means. It is more reasonable to initialize the variance $\boldsymbol{s}$ as $\boldsymbol{w}^2$ instead of $\exp[2\,\mathrm{abs}(\boldsymbol{w})]$.

In the algorithm, HESSIANCALC($\boldsymbol{w}$) is the process to calculate Hessian information with respect to the posterior mean $\boldsymbol{w}$ in order to produce the Hessian eigenbasis to perform the change of basis. For very small networks, we can calculate Hessian explicitly but it is prohibitive for most common networks. However, efficient approximate change of basis can be performed using our approximated layer-wise Hessians. In this case, we would just need to calculate the full eigenspace of $\mathbb{E}[\boldsymbol{M}]$ and that of $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ for each layer. For pth layer, we denote them as $\boldsymbol{U}^{(p)}$ and $\boldsymbol{V}^{(p)}$ respectively with eigenvectors as columns. We can also store the corresponding eigenvalues by doing pairwise multiplications between eigenvalues of $\mathbb{E}[\boldsymbol{M}]$ and $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$.

After getting the eigenspaces, we can perform the change of basis. Note that we perform change of basis on vectors with the same dimensionality as the parameter vector (or the posterior mean). TOHESSIAN($\boldsymbol{u}$) is the process to put a vector $\boldsymbol{u}$ in the standard basis to the Hessian eigenbasis. We first break $\boldsymbol{u}$ into different layers and let $\boldsymbol{u}^{(p)}$ be the vector for the pth layer. We then define $\mathrm{Mat}^{(p)}$

Algorithm 1 PAC-Bayes bound optimization using layer-wise Hessian eigenbasis

Input:

$\boldsymbol{w}_0 \in \mathbb{R}^P$ ▷ Network parameters (Initialization)
$\boldsymbol{w} \in \mathbb{R}^P$ ▷ Network parameters (SGD solution)
S ▷ Training examples
$\delta \in (0,1)$ ▷ Confidence parameter
$b \in \mathbb{N}, c \in (0,1)$ ▷ Precision and bound for λ
$\tau \in (0,1), T \in \mathbb{N}$ ▷ Learning rate; No. of iterations
$\eta \in \mathbb{N}$ ▷ Epoch interval for Hessian calculation

Output

$\boldsymbol{w}$ ▷ Optimized network parameters
$\boldsymbol{s}$ ▷ Optimized posterior variances in Hessian eigenbasis
λ ▷ Optimized prior variancce

procedure ITERATIVE-HESSIAN-PAC-BAYES
  $\boldsymbol{\varsigma} \leftarrow \log[\mathrm{abs}(\boldsymbol{w})]$ ▷ where $\boldsymbol{s}(\boldsymbol{\varsigma}) = \exp(2\boldsymbol{\varsigma})$
  $\varrho \leftarrow -3$ ▷ where $\lambda(\varrho) = \exp(2\varrho)$
  $R(\boldsymbol{w}, \boldsymbol{s}, \lambda) = \sqrt{\frac{1}{2} B_{\mathrm{RE}}(\boldsymbol{w}, \boldsymbol{s}, \lambda; \delta)}$ ▷ BRE term
  $B(\boldsymbol{w}, \boldsymbol{s}, \lambda, \boldsymbol{w}') = \mathcal{L}(\boldsymbol{w}') + R(\boldsymbol{w}, \boldsymbol{s}, \lambda)$ ▷ Optimization goal
  for $t = 0 \to T-1$ **do** ▷ Run SGD for T iterations
    if $t \mod \eta == 0$ **then**
      HESSIANCALC(w)
    end if
    Sample $\boldsymbol{\xi} \sim \mathcal{N}(0,1)^P$
    $\boldsymbol{w}'(\boldsymbol{w}, \boldsymbol{\varsigma}) = \boldsymbol{w} + \text{TOSTANDARD}\,(\boldsymbol{\xi} \odot \exp(\boldsymbol{\varsigma}))$ ▷ Generate noisy parameter for SNN
    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \tau\left[\nabla_{\boldsymbol{w}} R(\boldsymbol{w}, \boldsymbol{s}, \lambda) + \nabla_{\boldsymbol{w}'} \mathcal{L}(\boldsymbol{w}')\right]$
    $\boldsymbol{\varsigma} \leftarrow \boldsymbol{\varsigma} - \tau\left[\nabla_{\boldsymbol{\varsigma}} R(\boldsymbol{w}, \boldsymbol{s}(\boldsymbol{\varsigma}), \lambda) + \text{TOHESSIAN}\,(\nabla_{\boldsymbol{w}'} \mathcal{L}(\boldsymbol{w}')) \odot \boldsymbol{\xi} \odot \exp(\boldsymbol{\varsigma})\right]$
    $\varrho \leftarrow \varrho - \tau \nabla_{\varrho} R(\boldsymbol{w}, \boldsymbol{s}, \lambda(\varrho))$ ▷ Gradient descent
  end for
  return $w, s(\boldsymbol{\varsigma}), \lambda(\varrho)$
end procedure

as the reshape of a vector to the shape of the parameter matrix $\boldsymbol{W}^{(p)}$ of that layer. We have the new vector $\boldsymbol{v}^{(p)}$ in Hessian basis as

$$\boldsymbol{v}^{(p)} = \mathrm{vec}\left[\boldsymbol{U}^{(p)T}\,\mathrm{Mat}^{(p)}(\boldsymbol{u}^{(p)})\boldsymbol{V}^{(p)}\right]. \tag{49}$$

The new vector $\boldsymbol{v} = \text{TOHESSIAN}(\boldsymbol{u})$ is thus the concatenation of all the $\boldsymbol{v}^{(p)}$.

$\text{TOSTANDARD}(\boldsymbol{v})$ is the process to put a vector $\boldsymbol{v}$ in the Hessian eigenbasis to the standard basis. It is the reverse process to TOHESSIAN. We also break $\boldsymbol{v}$ into layers and let the vector for the pth layer be $\boldsymbol{v}^{(p)}$. Then, the new vector $\boldsymbol{u}^{(p)}$ is

$$\boldsymbol{u}^{(p)} = \mathrm{vec}\left[\boldsymbol{U}^{(p)}\,\mathrm{Mat}^{(p)}(\boldsymbol{v}^{(p)})\boldsymbol{V}^{(p)T}\right], \tag{50}$$

The new vector $\boldsymbol{u} = \text{TOSTANDARD}(\boldsymbol{v})$ is thus the concatenation of all $\boldsymbol{u}^{(p)}$.

After getting optimized $\boldsymbol{w}, \boldsymbol{s}, \lambda$, we compute the final bound using Monte Carlo methods same as in Dziugaite & Roy (2017).

Note that the prior P is invariant with respect to the change of basis, since its covariance matrix is a multiple of identity $\lambda \boldsymbol{I}_P$. Thus, the KL divergence can be calculate in the Hessian eigenbasis without changing the value of λ.

We followed the experiment setting proposed by Dziugaite & Roy (2017) in general. In all the results we present, we first trained the models from Gaussian random initialization w_0 to the initial posterior mean estimate w using SGD (lr=0.01) with batch-size 128 and epoch number 1000.

We then optimize the posterior mean and variance with layer-wise Hessian information using Algorithm 1, where $\delta = 0.025$, $b = 100$, and $c = 0.1$. We train for 2000 epochs, with learning

rate τ initialized at 0.001 and decays with ratio 0.1 every 400 epochs. For Approximated Hessian algorithm, we set $\eta = 1$. For Iterative Hessian algorithm, we set $\eta = 10$. We also tried η with the same decay schedule as learning rate (multiply η by 10 every time the learning rate is multiplied by 0.1) and the results are similar to those without decay. We also used the same Monte Carlo method as in Dziugaite & Roy (2017) to calculate the final PAC-Bayes bound. Except that we used 50000 iterations instead of 150000 iterations because extra iterations do not further tighten the bound significantly. We use sample frequency 100 and $\delta' = 0.01$ as in that paper.

The complete experiment results are listed in Table 9. We follow the same naming convention as in Dziugaite & Roy (2017) except adding T-200^2 we introduced in Section 4. T-600_{10}, T-600^2_{10}, and T-200^2_{10} are trained on standard MNIST with 10 classes, and others are trained on MNIST-2 (see Appendix B.1), in which we combined class 0-4 and class 5-9.

In Table 9, Prev means the previous results in Dziugaite & Roy (2017), Approx-H means Approximated Hessian, Iter-H means Iterative Hessian, Iter-H(decay) means Iterative Hessian with decaying η. Those without parentheses are vanilla PAC-Bayes optimization as in the previous paper.

Table 9: Full PAC-Bayes bound optimization results

Network	**Method**	PAC-Bayes Bound	KL Divergence	SNN loss	λ (prior)	Test Error
T-600	Prev	0.161	5144	0.028	-	0.017
	Vanilla	0.154	4612.6	0.03373	-1.3313	0.0153
	Approx-H	0.1432	3980.6	0.03417	-1.6063	0.0153
	Iter-H	**0.1198**	3766.1	0.02347	-1.2913	0.0153
	Iter-H (decay)	0.1199	3751.1	0.02366	-1.2913	0.0153
T-600^2	Prev	0.186	6534	0.028	-	0.016
	Vanilla	0.1921	6966.6	0.03262	-1.4163	0.0148
	Approx-H	0.1658	5176.1	0.03468	-2.0963	0.0148
	Iter-H	0.1456	5086.5	0.02473	-1.7963	0.0148
	Iter-H (decay)	**0.1443**	4956.8	0.02523	-1.7963	0.0148
T-1200	Prev	0.179	5977	0.027	-	0.016
	Vanilla	0.1754	5917.6	0.03295	-1.5463	0.0161
	Approx-H	0.1725	5318.8	0.03701	-1.8313	0.0161
	Iter-H	0.1417	5071	0.02292	-1.4763	0.0161
	Iter-H (decay)	**0.1413**	5021.1	0.02316	-1.4763	0.0161
T-300^2	Prev	0.17	5791	0.027	-	0.015
	Vanilla	0.1686	5514.9	0.03329	-1.1513	0.015
	Approx-H	0.1434	4105.4	0.03296	-1.8063	0.015
	Iter-H	0.1249	3873.2	0.02514	-1.4763	0.015
	Iter-H (decay)	**0.1244**	3833.7	0.02526	-1.4763	0.015
R-600	Prev	1.352	201131	0.112	-	0.501
	Vanilla	0.6046	1144.8	0.507	-1.8263	0.4925
	Approx-H	**0.5653**	390.25	0.5066	-2.4713	0.4925
	Iter-H (decay)	0.5681	431.62	0.5066	-2.4513	0.4925
T-200^2_{10}	Vanilla	0.4165	21896	0.04706	-1.1513	0.0208
	Approx-H	0.2621	11068	0.0366	-1.4213	0.0208
	Iter-H	**0.2145**	9821	0.02229	-1.1513	0.0208
	Iter-H (decay)	0.2311	9758.5	0.03071	-1.1513	0.0208
T-600_{10}	Vanilla	0.2879	12674	0.03854	-1.1513	0.018
	Approx-H	0.2424	9095.8	0.04159	-1.6013	0.018
	Iter-H	**0.2132**	8697.9	0.02947	-1.3063	0.018
T-600^2_{10}	Vanilla	0.3472	17212	0.03884	-1.1513	0.0186
	Approx-H	0.2896	11618	0.04723	-2.0563	0.0186
	Iter-H	**0.2431**	10568	0.03057	-1.5713	0.0186

We also plotted the final posterior variance, $\boldsymbol{s}$. Fig. 27 shown below is for T-200^2_{10}. For posterior variance optimized with Approximated Hessian and Iterative Hessian, we can see that direction associated with larger eigenvalue has a smaller variance. This agrees with our presumption that top eigenvectors are aligned with sharper directions and should have smaller variance after optimization. The effect is more significant and consistent for Iterative Hessian, where the PAC-Bayes bound is also tighter.

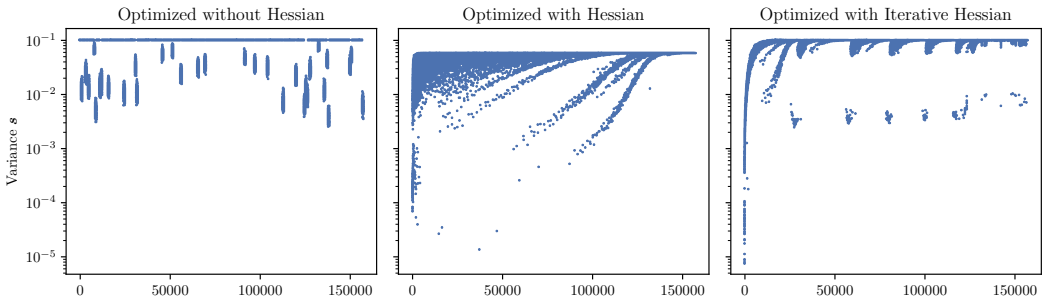


Figure 27: Optimized posterior variance, $\boldsymbol{s}$. (fc1:T-200^2, trained on MNIST), the horizontal axis is ordered with decreasing eigenvalues.

G CODES DESCRIPTION

We include 3 packages in the source codes: *closeformhessian*, *hessian_eigenthings*, and *eigen_pac_bayes*.

closeformhessian gives the methods to calculate matrices we used in our experiments, like $\mathbb{E}[\boldsymbol{M}]$ and $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$, and others mentioned in Appendix A. Most of the computational functions are in decomp.py. The names should be self-explanatory, except that we use some old namings (U for $\boldsymbol{G}$, UTAU for $\boldsymbol{M}$).

hessian_eigenthings is used to calculate eigenvalues and eigenvectors as described in Appendix C, modified from Golmant et al. (2018).

eigen_pac_bayes is the package to compute the PAC-Bayes bounds described in Section 6 and Appendix F. The *hpb* folder contains the core files for the package and the *algos* folder contains packages described above. Files in the main folder are example codes to run this package. Files ended with run.py are used for datasets with ± 1 labels like MNIST-2. Files ended with run_10class.py are used for datasets with multiple classes like MNIST. They do not actually require the number of classes to be 10.