WHY BIGGER IS NOT ALWAYS BETTER: ON FINITE AND INFINITE NEURAL NETWORKS

Laurence Aitchison *
Janelia Research Campus
19700 Helix Drive,
Ashburn, VA 20147, USA
laurence.aitchison@gmail.com

ABSTRACT

Recent work has shown that the outputs of convolutional neural networks become Gaussian process (GP) distributed when we take the number of channels to infinity. In principle, these infinite networks should perform very well, both because they allow for exact Bayesian inference, and because widening networks is generally thought to improve (or at least not diminish) performance. However, Bayesian infinite networks perform poorly in comparison to finite networks, and our goal here is to explain this discrepancy. We note that the high-level representation induced by an infinite network has very little flexibility; it depends only on network hyperparameters such as depth, and as such cannot learn a good high-level representation of data. In contrast, finite networks correspond to a rich prior over high-level representations, corresponding to kernel hyperparameters. We analyse this flexibility from the perspective of the prior (looking at the structured prior covariance of the top-level kernel), and from the perspective of the posterior, showing that the representation in a learned, finite deep linear network slowly transitions from the kernel induced by the inputs towards the kernel induced by the outputs, both for gradient descent, and for Langevin sampling. Finally, we explore representation learning in deep, convolutional, nonlinear networks, showing that learned representations differ dramatically from the corresponding infinite network.

One approach to understanding and improving neural networks is to perform Bayesian inference in an infinitely wide network, which can be done both for fully connected (Lee et al., 2018; Matthews et al., 2018) and convolutional networks (Garriga-Alonso et al., 2019; Novak et al., 2019). In this limit the outputs become Gaussian process distributed, enabling efficient and exact reasoning about uncertainty, and giving a means of interpretation using the parameter-free kernel function (which depends only on network hyperparameters such as depth). However, the performance of Bayesian infinite networks lags considerably behind state-of-the-art finite networks trained using SGD (e.g. compare performance in Garriga-Alonso et al. (2019), Novak et al. (2019) and Arora et al. (2019) against He et al. (2016) and Chen et al. (2018)). This seems surprising, because, to our knowledge, there are no reports of wider networks degrading classification performance (indeed, the opposite is sometimes argued; see Zagoruyko & Komodakis, 2016), and because exact Bayesian inference is provably optimal, if the prior accurately describes our beliefs (Ramsey, 1926). Indeed, recent work on the Neural Tangent Kernel (NTK) (Jacot et al., 2018) has suggested that deterministic gradient descent in an infinite network gives slighly lower performance than Bayesian inference in the same network (Arora et al., 2019).

Our hypothesis is that the poor performance of Bayesian infinite networks arises because the top-layer representation (equivalent to the kernel), is fixed by the network hyperparameters, and thus cannot be learned from data. This breaks many of our key intuitions about why deep networks are effective. For instance in transfer learning we use a large-scale task such as ImageNet to a learn a good high-level representation, then apply this representation to other tasks where less data is available. However, transfer learning is impossible in Bayesian neural networks, because the top-

*University of Bristol, University of Cambridge

layer representation is fixed by the network hyperparameters and so cannot be learned using e.g. ImageNet.

To understand these issues, we analysed finite networks using tools inspired by the analysis of infinite networks. We were able to show that, within the Bayesian setting, finite networks allow the flexibility to learn useful high-level representations of data. We begin by giving a toy, two-layer example, which demonstrates that finite networks offer potential benefits under conditions of model-mismatch. We then characterised the flexibility offered by finite networks in two ways. First, we considered the prior viewpoint, i.e. the covariance in the top-layer kernel induced by randomness in the lower-layer weights. In particular, we showed that narrower, deeper networks offer more flexibility, and that CNNs offer more flexibility than locally connected networks (LCNs) when the input is spatially structured. Second, we considered the posterior viewpoint, showing that under both MAP inference and sampling, the representations in learned neural networks slowly transition from being similar to the input kernel (i.e. the inner product of the inputs) to being similar to the output kernel (i.e. the inner product of one-hot vectors representing targets). We found an important difference between MAP inference and sampling: for MAP inference, the learned representations transition from the input to the output kernel, irrespective of the network width. In contrast, sampled networks behave similarly when the network width and the number of output channels have the same scale, but as the network width increases, the learned representations become increasingly dominated by the prior, and insensitive to the outputs. Finally, we considered representation learning in finite, deep, non-linear, convolutional networks, confirming the intuition that the networks learn a top-layer representation similar to the output kernel, and showing that the learned intermediate representations differ dramatically from those under the prior.

1 RESULTS

1.1 TOY EXAMPLE

In the introduction, we noted that infinite Bayesian networks perform worse than standard neural networks trained using stochastic gradient descent. Thus, as we make finite neural networks wider, there should be some point at which performance begins to degrade. We considered a simple, two-layer, fully-connected linear network with the full set of 20 4-dimensional inputs denoted $\mathbf{X}$, hidden unit activations denoted $\mathbf{H}$, and 10-dimensional outputs denoted $\mathbf{Y}$,

$$\mathbf{H} = \mathbf{X}\mathbf{W} \qquad\qquad \mathbf{Y} = \mathbf{H}\mathbf{V} + \sigma\boldsymbol{\Xi} \tag{1}$$

where $\boldsymbol{\Xi}$ is IID standard Gaussian noise, $\mathbf{W}$ is the input-to-hidden weight matrix and $\mathbf{V}$ is the hidden-to-output weight matrix, whose columns, $\mathbf{w}_\mu$ and $\mathbf{v}_\nu$ are generated IID from,

$$\mathrm{P}\left(\mathbf{w}_\mu\right) = \mathcal{N}\left(\mathbf{w}_\mu; \mathbf{0}, \tfrac{1}{X}\mathbf{I}\right) \qquad\qquad \mathrm{P}\left(\mathbf{v}_\nu\right) = \mathcal{N}\left(\mathbf{v}_\nu; \mathbf{0}, \tfrac{1}{N}\mathbf{I}\right), \tag{2}$$

and where the variance of the weights is normalised by the number of inputs to that layer, X for the 4-dimensional input, and N for the width of the hidden layer.

In the first example (Fig. 1 left), we generated targets for supervised learning using a second neural network with weights generated as described above, and $H_{\text{gen}} \in \{1, 2, 4\}$ hidden units. We evaluated the Bayesian model-evidence for networks with many different numbers of hidden units (x-axis). Bayesian reasoning would suggest that the model evidence for the true model (i.e. with a matched number of hidden units) should be higher than the model evidence for any other model, as indeed we found (Fig. 1 top left), and these patterns held true for the predictive probability, or equivalently test performance (Fig. 1 bottom left). While these results give an example where smaller networks perform better, they do not necessarily help us to understand the behaviour of neural networks on real datasets, where the true generative process for the data is not known, and is certainly not in our model class. As such, we considered two further examples where the neural network generating the targets lay outside of our model class. First, we first used the same neural network to generate the targets (with $H_{\text{gen}} = 4$), but multiplying the inputs by 100 (Fig. 1 middle). Second, we modified the inputs by zeroing-out all but the first input unit (Fig. 1 right). In both of these experiments, there was an optimium number of hidden units, after which performance degraded as more hidden units were included.

To understand why this might be the case, it is insightful to consider the methods we used to evaluate the model evidence and generate these results. In particular, note that conditioned on $\mathbf{H}$, the output

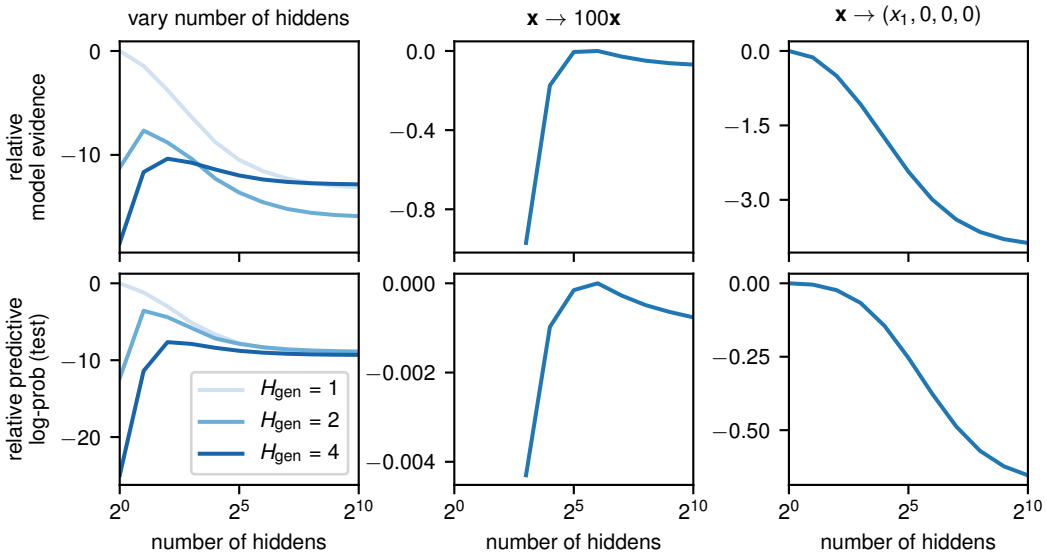

Figure 1: A toy fully-connected, two-layer Bayesian linear network showing situations in which smaller networks perform better than larger networks. Left: training data generated from the prior with H_{gen} hidden units. Middle: training data generated from the prior with $H_{\text{gen}} = 4$ but where we scale-up the inputs by a factor of 100. Right: training data generated from the prior with $H_{\text{gen}} = 4$, but where we zero-out all but the first input dimension. Top: Bayesian model evidence. Bottom: predictive log-probability, or equivalently test-error.

for any given channel, $\mathbf{y}_\nu$, is IID and depends only on the corresponding column of the output weights, $\mathbf{v}_\nu$,

$$\mathrm{P}\left(\mathbf{Y}|\mathbf{V},\mathbf{H}\right) = \prod_\nu \mathrm{P}\left(\mathbf{y}_\nu|\mathbf{v}_\nu,\mathbf{H}\right) = \prod_\nu \mathcal{N}\left(\mathbf{y}_\nu;\mathbf{H}\mathbf{v}_\nu,\sigma^2\mathbf{I}\right). \tag{3}$$

Thus, we can integrate over the output weights, $\mathbf{v}_\nu$, to obtain a distribution over $\mathbf{Y}$ conditioned on $\mathbf{H}$,

$$\mathrm{P}\left(\mathbf{Y}|\mathbf{H}\right) = \prod_\nu \mathrm{P}\left(\mathbf{y}_\nu|\mathbf{H}\right) = \prod_\nu \mathcal{N}\left(\mathbf{y}_\nu;\mathbf{0},\tfrac{1}{N}\mathbf{H}\mathbf{H}^T + \sigma^2\mathbf{I}\right), \tag{4}$$

which is the classical Gaussian process representation of Bayesian linear regression (Rasmussen & Williams, 2006). Remembering that the hidden activities, $\mathbf{H}$, is a deterministic function of the weights, $\mathbf{W}$, and inputs, $\mathbf{X}$, we can write this distribution as,

$$\mathrm{P}\left(\mathbf{y}_\nu|\mathbf{H}\right) = \mathrm{P}\left(\mathbf{y}_\nu|\mathbf{W},\mathbf{X}\right) = \mathcal{N}\left(\mathbf{y}_\nu;\mathbf{0},\tfrac{1}{H}\mathbf{X}\mathbf{W}\mathbf{W}^T\mathbf{X}^T + \sigma^2\mathbf{I}\right). \tag{5}$$

Thus, the first-layer weights, $\mathbf{W}$, act as kernel hyperparameters in a Gaussian process: they control the covariance of the outputs, $\mathbf{y}_\nu$. To evaluate the model evidence we need to integrate over $\mathbf{W}$,

$$\mathrm{P}\left(\mathbf{Y}|\mathbf{X}\right) = \int \mathbf{dW}\, \mathrm{P}\left(\mathbf{W}\right) \prod_\nu \mathrm{P}\left(\mathbf{y}_\nu|\mathbf{W},\mathbf{X}\right) = \mathop{\mathbb{E}}_{\mathrm{P}(\mathbf{W})}\left[\prod_\nu \mathrm{P}\left(\mathbf{y}_\nu|\mathbf{W},\mathbf{X}\right)\right] \tag{6}$$

and we estimate this integral by drawing $64,000$ samples from the prior, $\mathrm{P}\left(\mathbf{W}\right)$. Importantly, while $\mathbf{W}$ provides flexibility in the kernel in finite networks, this flexibility gradually disappears as we consider larger networks. In particular,

$$\lim_{N\to\infty} \tfrac{1}{N}\mathbf{W}\mathbf{W}^T = \lim_{N\to\infty} \tfrac{1}{N}\sum_{\mu=1}^{N} \mathbf{w}_\mu\mathbf{w}_\mu^T = \mathbb{E}\left[\mathbf{w}_\mu\mathbf{w}_\mu^T\right] = \tfrac{1}{X}\mathbf{I}. \tag{7}$$

Therefore, in this limit, the distribution over $\mathbf{Y}$ converges to,

$$\lim_{N\to\infty} \mathrm{P}\left(\mathbf{Y}|\mathbf{X}\right) = \prod_\nu \mathcal{N}\left(\mathbf{y}_\nu;\mathbf{0},\tfrac{1}{X}\mathbf{X}\mathbf{X}^T + \sigma^2\mathbf{I}\right). \tag{8}$$

This is exactly the distribution we would expect from Bayesian linear regression in a one-layer network. Thus, by taking the infinite limit, we have eliminated the additional flexibility afforded by the two-layer network, and we can see that the superior performance of smaller networks in Fig. 1 emerges because they give additional flexibility in the covariance of the outputs, which gradually disappears as network size increases.

1.2 KERNEL REPRESENTATIONS FOR FINITE NETWORKS

In the previous section, we considered the simplest networks in which these phenomena emerge: a two-layer, linear network. In this section, we setup a finite deep nonlinear network and show that activity flowing through this network can be understood entirely in terms of kernel matricies.

Consider a single layer within a fully-connected network, where the activity at the previous layer, $\mathbf{H}_{\ell-1}$, corresponding to a batch containing all inputs, is multiplied by a weight matrix, $\mathbf{W}_\ell$, to give activations, $\mathbf{A}_\ell$. This activation matrix, $\mathbf{A}_\ell$ is multiplied by another matrix, $\mathbf{M}_\ell$, to give an updated activation matrix, $\mathbf{A}'_\ell$. Critically, this multiplication leaves the representation unchanged (Fig. 2): $\mathbf{M}_\ell$ simply helps to ensure that we can exactly compute the kernel in nonlinear finite networks (see below). Finally, the activations, $\mathbf{A}'_\ell$, are passed through a non-linearity, ϕ, to give the activity at this layer, $\mathbf{H}_\ell$,

$$\mathbf{A}_\ell = \mathbf{H}_{\ell-1}\mathbf{W}_\ell \qquad \mathbf{A}'_\ell = \mathbf{A}_\ell\mathbf{M}_\ell \qquad \mathbf{H}_\ell = \phi\left(\mathbf{A}'_\ell\right), \tag{9}$$

where,

$$\mathbf{H}_\ell \in \mathbb{R}^{P\times M_\ell} \quad \mathbf{A}_\ell \in \mathbb{R}^{P\times N_\ell} \quad \mathbf{A}'_\ell \in \mathbb{R}^{P\times M_\ell} \quad \mathbf{W}_\ell \in \mathbb{R}^{M_{\ell-1}\times N_\ell} \quad \mathbf{M}_\ell \in \mathbb{R}^{N_\ell\times M_\ell} \tag{10}$$

In a standard neural network, we would set $\mathbf{M}_\ell = \mathbf{I}$, such that $\mathbf{A}'_\ell = \mathbf{A}_\ell$ and $M_\ell = N_\ell$. For a fully-connected network, the columns of $\mathbf{W}_\ell$, denoted $\mathbf{w}^\ell_\mu$ are generated IID from a Gaussian distribution,

$$\mathrm{P}\left(\mathbf{W}_\ell\right) = \prod_{\lambda=1}^{N_\ell} \mathrm{P}\left(\mathbf{w}^\ell_\lambda\right) = \prod_{\lambda=1}^{N_\ell} \mathcal{N}\left(\mathbf{w}^\ell_\lambda; \mathbf{0}, \tfrac{1}{N_{\ell-1}}\mathbf{I}\right) \tag{11}$$

where the normalization constants, $1/N_{\ell-1}$, ensures that activations remain normalized.

We now define the activation kernel and activity kernel,

$$\mathbf{K}_\ell \equiv \tfrac{1}{N_\ell}\mathbf{A}_\ell\mathbf{A}_\ell^T = \tfrac{1}{M_\ell}\mathbf{A}'_\ell\mathbf{A}'^T_\ell \qquad \mathbf{L}_\ell \equiv \tfrac{1}{N_\ell}\mathbf{H}_\ell\mathbf{H}_\ell^T. \tag{12}$$

where the kernels for $\mathbf{A}_\ell$ and $\mathbf{A}'_\ell$ are equivalent, as we always use $\mathbf{M}_\ell$ for which $\mathbf{M}_\ell\mathbf{M}_\ell^T = \mathbf{I}$. The only remaining object is the covariance, $\mathbf{J}_\ell$. As each channel (column) of the activations is a linear function of the corresponding channel of the weights, $\mathbf{a}^\ell_\lambda = \mathbf{H}_{\ell-1}\mathbf{w}^\ell_\lambda$, the activations are Gaussian and IID conditioned on the activity at the previous layer,

$$\mathrm{P}\left(\mathbf{A}_\ell|\mathbf{H}_{\ell-1}\right) = \prod_{\mu=1}^{N_\ell} \mathrm{P}\left(\mathbf{a}^\ell_\mu|\mathbf{H}_{\ell-1}\right) = \prod_{\mu=1}^{N_\ell} \mathcal{N}\left(\mathbf{a}^\ell_\mu; \mathbf{0}, \mathbf{J}_\ell\right) = \mathrm{P}\left(\mathbf{A}_\ell|\mathbf{J}_\ell\right), \tag{13}$$

with covariance $\mathbf{J}_\ell$. For a fully connected network, the covariance, $\mathbf{J}_\ell$, is equal to the previous layer's activity-kernel, $\mathbf{L}_\ell$,

$$\mathbf{J}_\ell = \mathbf{L}_{\ell-1} = \tfrac{1}{N_{\ell-1}}\mathbf{H}_{\ell-1}\mathbf{H}_{\ell-1}^T, \tag{14}$$

but the relationship is more complex in convolutional architectures (Garriga-Alonso et al., 2019; Novak et al., 2019) (Appendix A.2).

In order to work entirely in the kernel domain, we need to be able to transform $\mathbf{L}_{\ell-1}$ to $\mathbf{J}_\ell$ to $\mathbf{K}_\ell$, and back to $\mathbf{L}_\ell$ (Fig. 2). The first transformation, from the activity kernel at the previous layer, $\mathbf{L}_{\ell-1}$, to the covariance, $\mathbf{J}_\ell$ is described above. To perform the next transformation, from the covariance, $\mathbf{J}_\ell$, to the activity kernel, $\mathbf{K}_\ell$, we sample activations, $\mathbf{A}_\ell$, from a Gaussian with covariance $\mathbf{J}_\ell$ (Eq. 13), then directly compute the activation kernel from the activations (Eq. 12). Both of these transformations can be performed in either finite or infinite networks. However, the key difficulty comes when we try to transform the activation kernel, $\mathbf{K}_\ell$, into the activity kernel, $\mathbf{L}_\ell$, in finite

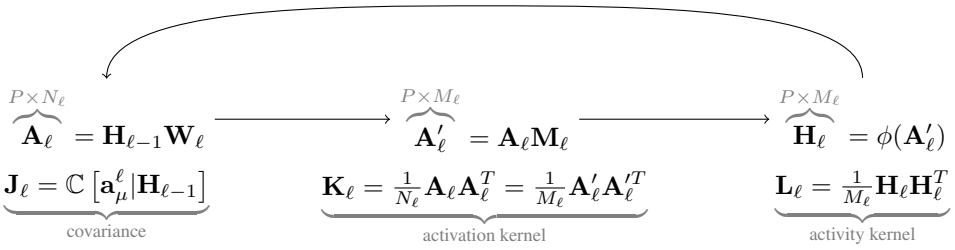

Figure 2: The relationships between the feature-space and kernel representations of the neural network. For a typical finite neural network, $\mathbf{M}_\ell = \mathbf{I}$, so $M_\ell = N_\ell$. For a finite-infinite network (which allows us to compute $\mathbf{L}_\ell$ from $\mathbf{K}_\ell$), we send $M_\ell \to \infty$, and draw the elements of $\mathbf{M}_\ell$ IID from a Gaussian distribution with zero mean and variance $1/M_\ell$.

networks. For deep linear networks, which are useful for analytical if not practical purposes, this issue does not emerge as the activity kernel *is* the activation kernel,

$$\mathbf{M}_\ell = \mathbf{I} \qquad \mathbf{H}_\ell = \mathbf{A}_\ell \qquad \mathbf{L}_\ell = \mathbf{K}_\ell. \tag{15}$$

However, to compute $\mathbf{L}_\ell$ for nonlinear networks, we need the activations, $\mathrm{P}\left(\mathbf{A}'_\ell|\mathbf{K}_\ell\right)$ to be Gaussian distributed, so that we can apply results from Cho & Saul (2009). In a standard finite network, where we take $\mathbf{M}_\ell = \mathbf{I}$, the distribution $\mathrm{P}\left(\mathbf{A}'_\ell|\mathbf{K}_\ell\right)$ cannot be Gaussian, as the activations are constrained to $\mathbf{K}_\ell = \frac{1}{M_\ell}\mathbf{A}'_\ell\mathbf{A}'^T_\ell$. This is the reason we require $\mathbf{M}_\ell$: if we allow $\mathbf{M}_\ell$ to be Gaussian distributed,

$$\mathrm{P}\left(\mathbf{M}_\ell\right) = \prod_{\lambda=1}^{M_\ell} \mathrm{P}\left(\mathbf{m}^\ell_\lambda\right) = \prod_{\lambda=1}^{M^\ell_\lambda} \mathcal{N}\left(\mathbf{m}^\ell_\lambda; \mathbf{0}, \tfrac{1}{N_\ell}\right) \tag{16}$$

and take the limit of $M_\ell \to \infty$,

$$\lim_{M_\ell\to\infty} \tfrac{1}{M_\ell}\mathbf{M}_\ell\mathbf{M}_\ell^T = \tfrac{1}{N_\ell}\mathbf{I} \qquad \lim_{M_\ell\to\infty} \tfrac{1}{M_\ell}\mathbf{A}'_\ell\left(\mathbf{A}'_\ell\right)^T = \tfrac{1}{N_\ell}\mathbf{A}_\ell\mathbf{A}_\ell^T = \mathbf{K}_\ell, \tag{17}$$

then we simultaneously have Gaussian $\mathrm{P}\left(\mathbf{A}'_\ell|\mathbf{K}_\ell\right)$, and we have left the activation kernel unchanged, $\mathbf{K}_\ell = \frac{1}{M_\ell}\mathbf{A}'_\ell\mathbf{A}'^T_\ell$. As this network alternates between finite N_ℓ and infinite M_ℓ layers, we call it a finite-infinite network.

1.3 Deep neural networks as deep Gaussian processes

Given the above setup, we can see that even a finite network with $\mathbf{M}_\ell = \mathbf{I}$ is a deep Gaussian process. In particular, in a deep Gaussian process, the activations at layer ℓ, denoted $\mathbf{A}_\ell$, consist of N_ℓ IID channels that are Gaussian-process distributed (Eq. 13), with a kernel/covariance determined by the activations at the previous layer. For a fully connected network,

$$\mathbf{J}_\ell = \mathbf{L}_{\ell-1} = \tfrac{1}{N_{\ell-1}}\phi\left(\mathbf{A}_{\ell-1}\right)\phi^T\left(\mathbf{A}_{\ell-1}\right). \tag{18}$$

The relationship between finite neural networks and Deep GPs is worth noting, because the same intuition, of the lower-layers shaping the top-layer kernel, arises in both senarios (e.g. Bui et al., 2016), and because there is potential for applying GP inference methods for neural networks, and vice versa.

1.4 Kernel flexibility: Prior viewpoint: Analytical

We can analyse how flexibility in the kernel emerges, by looking at the variability (i.e. the variance and covariance) of $\mathbf{J}_\ell$, $\mathbf{K}_\ell$ and $\mathbf{L}_\ell$. In the appendix, we derive recursive updates for deep, linear, convolutional networks,

$$\mathbb{C}\left[J^\ell_{ir,jr}, J^\ell_{ku,lu}|\mathbf{L}^0\right] = \tfrac{1}{D^2_{\ell-1}}\sum_{dd'}\mathbb{C}\left[L^{\ell-1}_{i(r+d),j(r+d)}, L^{\ell-1}_{k(u+d'),l(u+d')}|\mathbf{L}^0\right] \tag{19a}$$

$$\begin{aligned}\mathbb{C}\left[K^\ell_{ir,jr}, K^\ell_{ku,lu}|\mathbf{L}^0\right] \approx{}& \mathbb{C}\left[J^\ell_{ir,jr}, J^\ell_{ku,lu}|\mathbf{L}^0\right] \\ &+ \tfrac{1}{N_\ell}\left(\left\langle J^\ell_{ir,ku}\right\rangle\left\langle J^\ell_{jr,lu}\right\rangle + \left\langle J^\ell_{ir,lu}\right\rangle\left\langle J^\ell_{jr,ku}\right\rangle\right) + \mathcal{O}(1/N^2_\ell)\end{aligned} \tag{19b}$$

$$\mathbb{C}\left[L^\ell_{ir,jr}, L^\ell_{ku,lu}|\mathbf{L}^0\right] = \mathbb{C}\left[K^\ell_{ir,jr}, K^\ell_{ku,lu}|\mathbf{L}^0\right] \tag{19c}$$

where,

$$\left\langle J^{\ell}_{ir,jr} \right\rangle = \mathbb{E}\left[J^{\ell}_{ir,jr} | \mathbf{L}^0 \right] \tag{19d}$$

and where i, j, k and l index datapoints, whereas r and u index spatial locations, and where fully connected networks are a special case where there is only one spatial location in the inputs and hidden layers.

For fully connected networks, this expression predicts that the variance of the kernel is proportional to the depth (including the last layer; $L+1$) and inversely proportional to the width, N,

$$\mathbb{C}\left[K^{\ell}_{is,jr}, K^{\ell}_{ku,lv} | \mathbf{L}^0 \right] \approx \tfrac{L+1}{N} \left(L^0_{ik} L^0_{jl} + L_{il^0} L^0_{jk} \right). \tag{20}$$

This expression is so simple because, for a fully-connected linear network, the expected covariance at each layer is the same. For more nonlinear, convolutional or locally-connected networks the covariance is still proportional to N, but the depth-dependence becomes more complex, as the covariance changes as it propagates through layers.

1.5 Kernel flexibility: Prior viewpoint: Experimental

To check the validity of these expressions, we sampled 10,000 neural networks from the prior, and evaluated the variance of the kernel for a single input (Fig. 3). These inputs were either spatially unstructured (i.e. white noise), or spatially structured, in which case the inputs were the same across the whole image. For fully connected networks, we confirmed that the variance of the kernel is proportional to the depth including the last layer, $L+1$, and inversely proportional to width, N (Fig. 3A). For locally connected networks, we found that structured and unstructured inputs gave the same kernel variance, which is expected as any spatial structured is destroyed after the first layer (Fig. 3B). Further, for convolutional networks with structured input, the variance of the kernel was proportional to network depth (Fig. 3C bottom), but whenever that spatial structure was absent, either because it was absent in the inputs or because it was eliminated by an LCN (Fig. 3BC bottom) the variance of the kernel was almost constant with depth (see Appendix A.2.1). The large decrease in kernel flexibility for locally connected networks might be one reason behind the result in Novak et al. (2019) that locally connected networks have performance that is very similar to an infinite-width network, in which all flexibility has been eliminated. Finally, as the spatial input size, S, increases, for convolutional networks with spatially structured inputs, the variance of the kernel is constant, whereas for locally connected or spatially unstructured inputs, the variance falls (Fig. 3D).

1.6 Kernel flexibility: posterior viewpoint: analytical

An alternative approach to understanding flexibility in finite neural networks is to consider the posterior viewpoint: how learning shapes top-level representations. To obtain analytical insights, we considered maximum a-posteriori and sampling based inference in a deep, fully-connected, linear network. In both cases, we found that learned neural networks shift the representation from being close to the input kernel, defined by,

$$\mathbf{K}_0 = \mathbf{L}_0 = \tfrac{1}{N_0} \mathbf{X}\mathbf{X}^T, \tag{21}$$

to being close the output kernel, defined by,

$$\mathbf{K}_{L+1} = \tfrac{1}{N_{L+1}} \mathbf{Y}\mathbf{Y}^T. \tag{22}$$

In particular, under MAP inference, the shape of the kernel smoothly transitions from the input to the output kernel (Appendix B.2),

$$\mathbf{K}_{\ell} = \left(\frac{N_{\ell <}}{N_{\leq \ell}} \right)^{\frac{\ell(L+1-\ell)}{L+1}} \left(\mathbf{K}_{L+1} \mathbf{K}_0^{-1} \right)^{\ell/(L+1)} \mathbf{K}_0, \tag{23}$$

where $N_{\ell <}$ is the geometric average of the width in layers $\ell+1$ to $L+1$, and $N_{\leq \ell}$ is the geometric average of the width in layers 1 to ℓ. Thus, the kernels (and the underlying weights) at each layer can be made arbitrarily large or small by changing the width, despite the prior distribution being chosen specifically to ensure that the scale of the kernels was invariant to network width. This is an issue

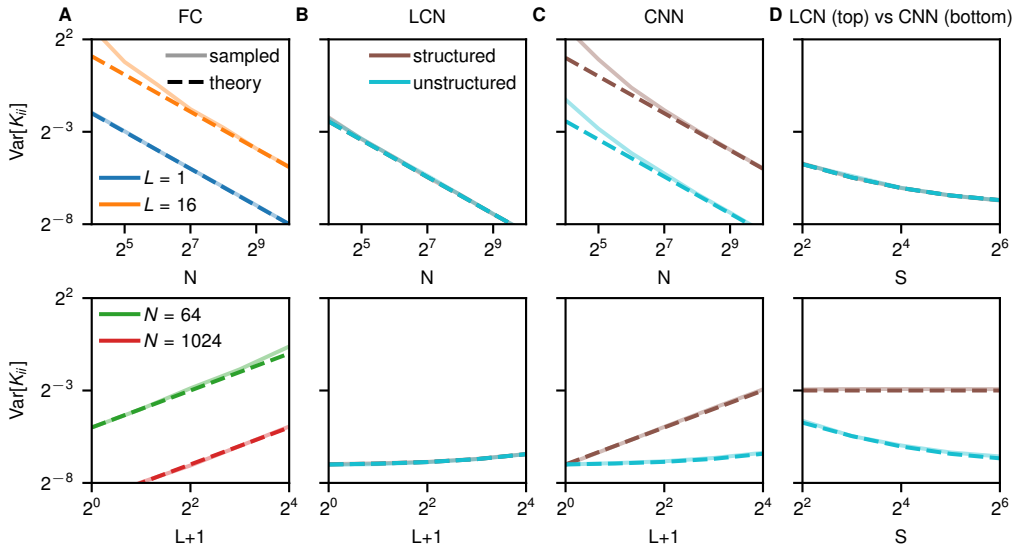

Figure 3: The variance of the kernel for linear, fully connected and convolutional networks, with spatially structured and unstructured inputs. The dashed lines in all plots display the theoretical approximation (Eq. 19) which is valid when the width is much greater than the number of layers. The solid lines display the empirical variance of the kernel from 10,000 simulations. **A** The variance of the kernel for fully connected networks, plotted against network width, N, for shallow (blue; $L+1=1$), and deep (orange; $L+1=16$) networks (top) and plotted against network depth, $L+1$, for narrow (green; $N=64$) and wide (red; $N=1024$) networks. **B** The variance of the kernel for locally connected networks with spatially structured and unstructured inputs, plotted against the number of channels, N, and against network depth, $L+1$. Note that the structured line lies underneath the unstructured line. The inputs are 1-dimensional with $S=32$ spatial locations, and 100 input channels. **C** As in **B**, but for convolutional networks. **D** The variance of the kernel as a function of the input spatial size, S, for deep ($L+1=16$) LCNs (top) and CNNs (bottom) with spatially strutured and unstructured inputs.

inherent to the use of MAP inference, which often finds modes that give a poor characterisation of the Bayesian posterior. In contrast, if we sample the weights using Langevin sampling (Appendix C), and set all the intermediate weights, from N_1 to N_L to N, then we get a similar recusion,

$$\mathbf{K}_\ell = \left(\mathbf{K}_L \mathbf{K}_0^{-1}\right)^{\ell/L} \mathbf{K}_0, \tag{24}$$

but where the top-layer representation, K_L depends on the ratio between the network width, N, and the number of output units, $Y = N_{L+1}$. In particular, if $Y = N$, then we get a relationship very similar to that for MAP inference,

$$\mathbf{K}_\ell = \left(\mathbf{K}_{L+1} \mathbf{K}_0^{-1}\right)^{\ell/(L+1)} \mathbf{K}_0, \tag{25}$$

as $\mathbf{K}_L = \left(\mathbf{K}_{L+1} \mathbf{K}_0^{-1}\right)^{L/(L+1)} \mathbf{K}_0$. However, as the network width grows very large, the prior begins to dominate, and the posterior becomes dominated by the prior,

$$\lim_{N/Y \to \infty} \mathbf{K}_\ell = \mathbf{K}_0, \tag{26}$$

as $\mathbf{K}_L = \mathbf{K}_0$. Finally, if the network width is small in comparison to the number of units,

$$\lim_{N/Y \to 0} \mathbf{K}_\ell = \left(\mathbf{K}_{L+1} \mathbf{K}_0^{-1}\right)^{\ell/L} \mathbf{K}_0, \tag{27}$$

as the top-layer kernel converges to the output, $\mathbf{K}_L = \mathbf{K}_{L+1}$.

1.7 KERNEL FLEXIBILITY: POSTERIOR VIEWPOINT: EMPIRICAL

The above results suggest that finite neural networks perform well by giving flexibility to interpolate between the input kernel and output kernel. To see how this happens in real neural networks, we considered a 34-layer ResNet without batchnorm corresponding to the infinite network in Garriga-Alonso et al. (2019) trained on CIFAR-10. We began by computing the correlation between elements of the finite and infinite kernel (Fig. 4A top) as we go through network layers (x-axis), and as we go through training (blue lines). As expected, the randomly initialized, untrained network retains a high correlation with the infinite kernel at all layers, though the correlation is somewhat smaller for higher layers, as there have been more opportunities for discrepancies to build up. However, for trained networks, this correspondence between the finite and infinite networks is far weaker: even at the first layer, the correlation is only around 0.5, and as we go through layers, the correlation decreases to almost zero. To understand whether this decrease in correlation indicated that the kernels were being actively shaped, we computed the correlation between the kernel for the finite network and the output kernel, defined by taking the inner product of vectors representing the one-hot class labels (Fig. 4A bottom). We found that while the correlation for the untrained network decreased across layers, training gives strong positive correlations with the output kernel, and these correlations increase as we move through network layers. While correlation is a useful simple measure of similarity, there are other measures of similarity that take into account the special structure of kernel matricies. In particular, we considered the marginal likelihood for the one-hot outputs corresponding to the class label, which is also a measure of the similarity of the kernel and the kernel formed by the one-hot outputs. To compute this measure, we maximized the marginal likelihood using a covariance that is a scaled sum of the kernel defined by that layer of the network and the identity (see Appendix D). As we would hope, we found that for the infinite network, the marginal likelihood increased somewhat as we moved through network layers, and the untrained finite network had similar performance, except that there was a fall in performance at the last layer. In contrast, the marginal likelihood for the finite, trained networks was initially very close to the infinite networks, but grows rapidly as we move through network layers.

To gain an insight into how training shaped the neural network kernels, we computed their eigenvalue spectrum. For the infinite network (Fig. 4C top), we found that the eigenvalue spectrum at all levels decayed as a -1 power law. This is expected at the lowest level due to the well known $1/f$ power spectrum of images (Van der Schaaf & van Hateren, 1996), but is not necessarily the case at higher-levels. Given the power-spectrum of the output kernel is just a small set of equal-sized eigenvalues corresponding to the class labels (Fig. 4C bottom, green line), we might expect the eigenspectrum of finite networks to gradually get steeper as we move through network layers. In fact, we find the opposite: for intermediate layers, the eigenvalue spectrum becomes flatter, which can be interpreted as the network attempting to retain as much information as possible about all aspects of the image. It is only at the last layer where the relevant information is selected, giving an eigenvalue spectrum with around 10 large and roughly equally-sized eigenvalues, followed by much smaller eigenvalues, which mirrors the spectrum of the output kernel.

2 CONCLUSIONS

We have shown that finite Bayesian neural networks have more flexibility than infinite networks, and that this may explain the superior performance of finite networks. We assessed flexibility from two perspectives. First, we looked at the prior viewpoint: the variability in the top-layer kernel induced by the prior over a finite neural network. Second, we looked at the posterior viewpoint: the ability of the learning process to shape the top-layer kernel. Under both MAP inference and sampling in finite networks, learning gradually shaped top-layer representations so as to match the output-kernel. But, as Bayesian (sampling) networks are made wider, the kernels become gradually less flexible, eliminating the possibility for learning to shape the kernel. In contrast, for MAP inference, the degree of kernel shaping is not affected by network width, and this additional flexibility might be an avenue for overfitting.

REFERENCES

Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Ruslan Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. *arXiv preprint arXiv:1904.11955*, 2019.

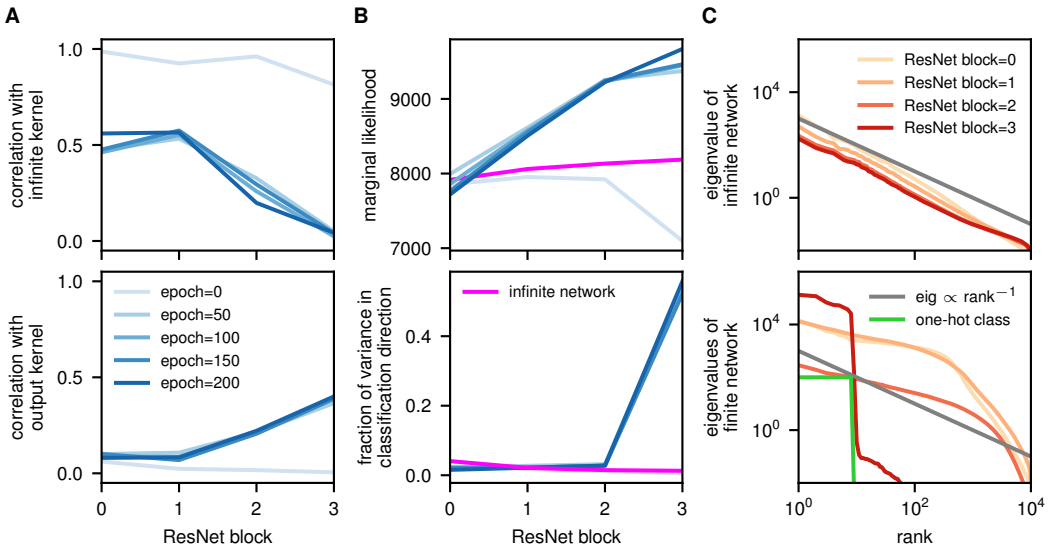

Figure 4: Comparison of kernels for finite and infinite neural networks at different layers. All kernels are computed on test data. **A** (top) Correlation (coefficient) between the kernel defined by the infinite network, and kernel defined by a finite network after different numbers of training epochs. **A** (bottom) Correlation (coefficient) between the kernel defined by the infinite network, and the output kernel defined by taking the inner product of one-hot vectors representing the class label. **B** (top) The Gaussian process marginal likelihood for the one-hot class labels. **B** (bottom) The fraction of variance in the direction of the one-hot output class labels. **C** (top) The eigenvalues of the kernel defined by the infinite network as we progress through layers, and compared to a -1 power law (grey). **C** (top) The eigenvalues of the kernel defined by the finite network after 200 training epochs, as we progress through network layers.

Thang Bui, Daniel Hernández-Lobato, Jose Hernandez-Lobato, Yingzhen Li, and Richard Turner. Deep gaussian processes for regression using approximate expectation propagation. In *International Conference on Machine Learning*, pp. 1472–1481, 2016.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pp. 6571–6583, 2018.

Youngmin Cho and Lawrence K Saul. Kernel methods for deep learning. *NeurIPS*, 2009.

Adrià Garriga-Alonso, Carl Edward Rasmussen, and Laurence Aitchison. Deep convolutional networks as shallow Gaussian processes. *ICLR*, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pp. 8571–8580, 2018.

Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as Gaussian processes. *ICLR*, 2018.

AGDG Matthews, M Rowland, J Hron, RE Turner, and Z Ghahramani. Gaussian process behaviour in wide deep neural networks. 2018.

Roman Novak, Lechao Xiao, Yasaman Bahri, Jaehoon Lee, Greg Yang, Jiri Hron, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Bayesian deep convolutional networks with many channels are Gaussian processes. *ICLR*, 2019.

Frank P Ramsey. Truth and probability. In *Readings in Formal Epistemology*, pp. 21–45. Springer, 1926.

Carl Edward Rasmussen and Christopher KI Williams. *Gaussian processes for machine learning*. MIT press, 2006.

van A Van der Schaaf and JH van van Hateren. Modelling the power spectra of natural images: statistics and information. *Vision research*, 36(17):2759–2770, 1996.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

A KERNEL FLEXIBILITY: PRIOR VIEWPOINT

To compute the covariance of the kernel for a deep network, we consider a recursion where we start with $\mathbb{C}\left[L_{ij}^{\ell-1}, L_{kl}^{\ell-1}|\mathbf{L}^0\right]$, then compute the resulting $\mathbb{C}\left[J_{ij}^{\ell}, J_{kl}^{\ell}|\mathbf{L}^0\right]$, then compute the resulting $\mathbb{C}\left[K_{ij}^{\ell}, K_{kl}^{\ell}|\mathbf{L}^0\right]$. In particular, we apply the law of total covariance for $\mathbf{K}_\ell|\mathbf{J}_\ell$, and we consider linear networks for which $\mathbf{L}_\ell = \mathbf{K}_\ell$,

$$\mathbb{C}\left[J_{ij}^{\ell}, J_{kl}^{\ell}|\mathbf{L}^0\right] = ? \tag{28a}$$

$$\mathbb{C}\left[K_{ij}^{\ell}, K_{kl}^{\ell}|\mathbf{L}^0\right] = \mathbb{C}\left[\mathbb{E}\left[K_{ij}^{\ell}|\mathbf{J}^{\ell}\right], \mathbb{E}\left[K_{kl}^{\ell}|\mathbf{J}^{\ell}\right]|\mathbf{L}^0\right] + \mathbb{E}\left[\mathbb{C}\left[K_{ij}^{\ell}, K_{kl}^{\ell}|\mathbf{J}^{\ell}\right]|\mathbf{L}^0\right] \tag{28b}$$

$$\mathbb{C}\left[L_{ij}^{\ell}, L_{kl}^{\ell}|\mathbf{L}^0\right] = \mathbb{C}\left[K_{ij}^{\ell}, K_{kl}^{\ell}|\mathbf{L}^0\right] \tag{28c}$$

The first equation is different for fully connected and convolutional networks, so we give its form later.

The expression for $\mathbf{K}_\ell|\mathbf{J}_\ell$ always behaves in the same way for linear and nonlinear, fully connected and convolutional networks so we consider this first. In particular, we always have $\mathbb{E}\left[\mathbf{K}_\ell|\mathbf{J}_\ell\right] = \mathbf{J}_\ell$, so the first term in Eq. (28b) is

$$\mathbb{C}\left[\mathbb{E}\left[K_{ij}^{\ell}|\mathbf{J}^{\ell}\right], \mathbb{E}\left[K_{kl}^{\ell}|\mathbf{J}^{\ell}\right]|\mathbf{L}^0\right] = \mathbb{C}\left[J_{ij}^{\ell}, J_{kl}^{\ell}|\mathbf{L}^0\right]. \tag{29}$$

For the second term in Eq. (28b), we substitute the definition of $\mathbf{K}_\ell$ (Eq. 12),

$$\begin{aligned}\mathbb{C}\left[K_{ij}^{\ell}, K_{kl}^{\ell}|\mathbf{J}^{\ell}\right] = \tfrac{1}{N_\ell^2}\sum_{\mu=1}^{N_\ell}\sum_{\nu=1}^{N_\ell}\mathbb{E}\left[a_{\mu i}^{\ell}a_{\mu j}^{\ell}a_{\nu k}^{\ell}a_{\nu l}^{\ell}|\mathbf{J}^{\ell}\right] \\ - \left(\tfrac{1}{N_\ell}\sum_{\mu=1}^{N_\ell}\mathbb{E}\left[a_{\mu i}^{\ell}a_{\mu j}^{\ell}|\mathbf{J}^{\ell}\right]\right)\left(\tfrac{1}{N_\ell}\sum_{\nu=1}^{N_\ell}\mathbb{E}\left[a_{\nu k}^{\ell}a_{\nu l}^{\ell}|\mathbf{J}^{\ell}\right]\right).\end{aligned} \tag{30}$$

The a's are jointly Gaussian, so their expectations are,

$$\mathbb{E}\left[a_{\mu i}^{\ell}a_{\mu j}^{\ell}a_{\nu k}^{\ell}a_{\nu l}^{\ell}|\mathbf{J}^{\ell}\right] = J_{ij}^{\ell}J_{kl}^{\ell} + \delta_{\mu\nu}\left(J_{ik}^{\ell}J_{jl}^{\ell} + J_{il}^{\ell}J_{jk}^{\ell}\right) \tag{31a}$$

$$\mathbb{E}\left[a_{\mu i}^{\ell}a_{\mu j}^{\ell}|\mathbf{J}^{\ell}\right] = J_{ij}^{\ell} \tag{31b}$$

$$\mathbb{E}\left[a_{\nu k}^{\ell}a_{\nu l}^{\ell}|\mathbf{J}^{\ell}\right] = J_{kl}^{\ell}. \tag{31c}$$

Thus, we can write the covariance of the kernels as,

$$\mathbb{C}\left[K_{ij}^{\ell}, K_{kl}^{\ell}|\mathbf{J}_\ell\right] = \tfrac{1}{N_\ell}\left(J_{ik}^{\ell}J_{jl}^{\ell} + J_{il}^{\ell}J_{jk}^{\ell}\right), \tag{32}$$

Substituting this into the second term in Eq. (28b) and writing the expected product in terms of the product of expectations,

$$\mathbb{E}\left[\mathbb{C}\left[K_{ij}^{\ell}, K_{kl}^{\ell}|\mathbf{J}_\ell\right]|\mathbf{L}_0\right] = \tfrac{1}{N_\ell}\mathbb{E}\left[J_{ik}^{\ell}J_{jl}^{\ell} + J_{il}^{\ell}J_{jk}^{\ell}|\mathbf{L}_0\right] \tag{33}$$

$$= \tfrac{1}{N_\ell}\left\langle J_{ik}^{\ell}\right\rangle\left\langle J_{jl}^{\ell}\right\rangle + \left\langle J_{il}^{\ell}\right\rangle\left\langle J_{jk}^{\ell}\right\rangle + \tfrac{1}{N_\ell}\left(\mathbb{C}\left[J_{ik}^{\ell}, J_{jl}^{\ell}|\mathbf{L}_0\right] + \mathbb{C}\left[J_{ik}^{\ell}, J_{jl}^{\ell}|\mathbf{L}_0\right]\right) \tag{34}$$

where,

$$\left\langle J_{ik}^{\ell}\right\rangle = \mathbb{E}\left[J_{ik}^{\ell}|\mathbf{L}_0\right]. \tag{35}$$

Thus, Eq. (28b) can be written,

$$\mathbb{C}\left[K^\ell_{ij}, K^\ell_{kl}|\mathbf{L}^0\right] = \mathbb{C}\left[J^\ell_{ij}, J^\ell_{kl}|\mathbf{L}^0\right] + \tfrac{1}{N_\ell}\left(\left\langle J^\ell_{ik}\right\rangle\left\langle J^\ell_{jl}\right\rangle + \left\langle J^\ell_{il}\right\rangle\left\langle J^\ell_{jk}\right\rangle\right) + \tfrac{1}{N_\ell}\left(\mathbb{C}\left[J^\ell_{ik}, J^\ell_{jl}|\mathbf{L}_0\right] + \mathbb{C}\left[J^\ell_{ik}, J^\ell_{jl}|\mathbf{L}_0\right]\right) \quad (36)$$

We could directly use this expression but it is somewhat unwieldy as the right-hand-side contains three covariance terms. Instead, note that it is possible to approximate this expression as the final two covariance terms are $\mathcal{O}(1/N^2)$, in comparison to $\mathcal{O}(1/N)$ for the other terms, and as such they become negligable as N grows,

$$\mathbb{C}\left[K^\ell_{ij}, K^\ell_{kl}|\mathbf{L}^0\right] \approx \mathbb{C}\left[J^\ell_{ij}, J^\ell_{kl}|\mathbf{L}^0\right] + \tfrac{1}{N_\ell}\left(\left\langle J^\ell_{ik}\right\rangle\left\langle J^\ell_{jl}\right\rangle + \left\langle J^\ell_{il}\right\rangle\left\langle J^\ell_{jk}\right\rangle\right). \quad (37)$$

Thus, the recursive updates become,

$$\mathbb{C}\left[J^\ell_{ij}, J^\ell_{kl}|\mathbf{L}^0\right] = ? \quad (38a)$$

$$\mathbb{C}\left[K^\ell_{is,jr}, K^\ell_{kl}|\mathbf{L}^0\right] \approx \mathbb{C}\left[J^\ell_{ij}, J^\ell_{kl}|\mathbf{L}^0\right] + \tfrac{1}{N_\ell}\left(\left\langle J^\ell_{ik}\right\rangle\left\langle J^\ell_{jl}\right\rangle + \left\langle J^\ell_{il}\right\rangle\left\langle J^\ell_{jk}\right\rangle\right) \quad (38b)$$

$$\mathbb{C}\left[L^\ell_{ij}, L^\ell_{kl}|\mathbf{L}^0\right] = \mathbb{C}\left[K^\ell_{ij}, K^\ell_{kl}|\mathbf{L}^0\right] \quad (38c)$$

A.1 Fully connected network

For a fully connected, network,

$$a^\ell_{\lambda,i} = \sum_\mu h^{\ell-1}_{i,\mu} W^\ell_{\mu,\lambda} \quad (39)$$

where the weights are drawn from a correlated, zero-mean Gaussian, with covariance

$$E\left[W^\ell_{\mu,\lambda} W^\ell_{\nu,\lambda}\right] = \tfrac{1}{N_\ell - 1}\delta_{\mu,\nu} \quad (40)$$

Thus, $\mathbf{a}^\ell_\lambda$ has distribution,

$$\mathrm{P}\left(\mathbf{a}^\ell_\lambda\right) = \mathcal{N}\left(\mathbf{a}^\ell_\lambda; \mathbf{0}, \mathbf{J}^\ell\right), \quad (41)$$

where $\mathbf{J}^\ell$ is given by,

$$J^\ell_{ij} = \mathbb{C}\left[a^\ell_{i,\lambda}, a^\ell_{j,\lambda}\right] = \mathbb{E}\left[a^\ell_{i,\lambda} a^\ell_{j,\lambda}\right] \quad (42)$$

$$= \mathbb{E}\left[\left(\sum_\mu h^{\ell-1}_{i,\mu} W^\ell_{\mu,\lambda}\right)\left(\sum_\nu h^{\ell-1}_{j,\nu} W^\ell_{\nu,\lambda}\right)\right] \quad (43)$$

$$= \sum_{\mu\nu} h^{\ell-1}_{i,\mu} h^{\ell-1}_{j,\nu}\, \mathbb{E}\left[W^\ell_{\mu,\lambda} W^\ell_{\nu,\lambda}\right] \quad (44)$$

substituting for the expectation (Eq. 40), and identifying the activity kernel (Eq. 12),

$$= \tfrac{1}{N_\ell - 1} \sum_\mu h^{\ell-1}_{i,\mu} h^{\ell-1}_{j,\mu} = L^{\ell-1}_{ij} \quad (45)$$

Thus,

$$\mathbb{C}\left[J^\ell_{ij}, J^\ell_{kl}|\mathbf{K}^0\right] = \mathbb{C}\left[L^{\ell-1}_{ij}, L^{\ell-1}_{kl}|\mathbf{K}^0\right] \quad (46)$$

which can be substituted into the recursion Eq. (38a) allowing the updates to be directly computed,

$$\mathbb{C}\left[J^\ell_{ij}, J^\ell_{kl}|\mathbf{L}^0\right] = \mathbb{C}\left[L^{\ell-1}_{ij}, L^{\ell-1}_{kl}|\mathbf{K}^0\right] \quad (47a)$$

$$\mathbb{C}\left[K^\ell_{is,jr}, K^\ell_{kl}|\mathbf{L}^0\right] \approx \mathbb{C}\left[J^\ell_{ij}, J^\ell_{kl}|\mathbf{L}^0\right] + \tfrac{1}{N_\ell}\left(\left\langle J^\ell_{ik}\right\rangle\left\langle J^\ell_{jl}\right\rangle + \left\langle J^\ell_{il}\right\rangle\left\langle J^\ell_{jk}\right\rangle\right) \quad (47b)$$

$$\mathbb{C}\left[L^\ell_{ij}, L^\ell_{kl}|\mathbf{L}^0\right] = \mathbb{C}\left[K^\ell_{ij}, K^\ell_{kl}|\mathbf{L}^0\right] \quad (47c)$$

A.2 CONVOLUTIONAL NETWORK

For locally connected and convolutional networks, we introduce spatial structure into the activations, and we use spatial indicies, r, s, u and v. Thus, the activations for datapoint i at layer ℓ, spatial location r and channel λ are given by,

$$a^\ell_{i,r\lambda} = \sum_{r'\mu} h^{\ell-1}_{i,r'\mu} W^\ell_{r'\mu,r\lambda}. \tag{48}$$

Note that for many purposes, these higher-order tensors can be treated as vectors and matrices, if we combine indicies (e.g. using a "reshape" or "view" operation). The commas in the index list are used to denote how to combine indicies for this particular operation, such that it can be understood as a standard matrix/vector operation. For the above equation, the activations, $\mathbf{a}^\ell \in \mathbb{R}^{P\times SN_\ell}$ are given by the matrix product of the activities from the previous layer, $\mathbf{h}^{\ell-1} \in \mathbb{R}^{P\times SN_{\ell-1}}$, and the weights, $\mathbf{W}_\ell \in \mathbb{R} \in^{SN_{\ell-1}\times SN_\ell}$, where remember that S is the number of spatial locations in the input.

For a convolutional neural network, the weights are the same if we consider the same input-to-output channels, and the same spatial displacement, d, and are uncorrelated otherwise,

$$E\left[W^\ell_{r'\mu,r\lambda} W^\ell_{s'\nu,s\lambda}\right] = \tfrac{1}{N_{\ell-1}D_{\ell-1}}\delta_{\mu,\nu} \sum_{d\in\mathcal{D}_{\ell-1}} \delta_{r',(r+d)}\delta_{s',(s+d)}. \tag{49}$$

where $\mathcal{D}_{\ell-1}$ is the set of all valid spatial displacements for the convolution, and $D_{\ell-1} = |\mathcal{D}_{\ell-1}|$ is the number of valid spatial displacements (i.e. the size of the convolutional patch). For a locally-connected network, the only additional requirement is that the output spatial locations are the same,

$$E\left[W^\ell_{r'\mu,r\lambda} W^\ell_{s'\nu,s\lambda}\right] = \tfrac{1}{N_{\ell-1}D_{\ell-1}}\delta_{\mu,\nu}\delta_{r,s} \sum_{d\in\mathcal{D}_{\ell-1}} \delta_{r',(r+d)}\delta_{s',(s+d)}. \tag{50}$$

Now we can compute the covariance of the activations, $\mathbf{J}^\ell$, for a convolutional network,

$$J^\ell_{ir,js} = \mathbb{E}\left[a^\ell_{ir,\lambda} a^\ell_{js,\lambda} | \mathbf{L}_{\ell-1}\right] \tag{51}$$

$$= \mathbb{E}\left[\left(\sum_{r'\mu} h^{\ell-1}_{i,r'\mu} W^\ell_{r'\mu,r\lambda}\right)\left(\sum_{s'\nu} h^{\ell-1}_{i,s'\nu} W^\ell_{s'\nu,r\lambda}\right)\middle|\, \mathbf{L}_{\ell-1}\right] \tag{52}$$

$$= \sum_{\mu\nu r's'} h^{\ell-1}_{ir',\mu} h^{\ell-1}_{is',\nu}\, \mathbb{E}\left[W^\ell_{r'\mu,r\lambda} W^\ell_{s'\nu,r\lambda}\right] \tag{53}$$

substituting the covariance of the weights (Eq. 50), and noting that the product of h's forms the definition of the activity kernel (Eq. 12),

$$J^\ell_{ir,js} = \tfrac{1}{D_{\ell-1}} \sum_{d\in\mathcal{D}_{\ell-1}} L^{\ell-1}_{i(r+d),j(s+d)} \tag{54}$$

For locally connected intermediate layers, the derivation here is the same as previously, except that the output locations must be the same for there to be any covariance in the weights,

$$J^\ell_{ir,js} = \tfrac{1}{D_{\ell-1}}\delta_{r,s} \sum_{d\in\mathcal{D}_{\ell-1}} L^{\ell-1}_{i(r+d),j(s+d)} \tag{55}$$

Substituting this into Eq. (38a),

$$\mathbb{C}\left[J^\ell_{ir,js}, J^\ell_{ku,lv} | \mathbf{K}^0\right] = \mathbb{C}\left[\tfrac{1}{D_{\ell-1}} \sum_{d\in\mathcal{D}_{\ell-1}} L^{\ell-1}_{i(r+d),j(s+d)}, \tfrac{1}{D_{\ell-1}} \sum_{d\in\mathcal{D}_{\ell-1}} L^{\ell-1}_{k(u+d),l(v+d)}\right]. \tag{56}$$

Now, we can put together full recursive updates for convolutional networks, by pulling the sum out of the covariance above, and by taking the indicies in Eq. (38), as indexing both a datapoint and a spatial location (i.e. $i \to i, s$),

$$\mathbb{C}\left[J^\ell_{ir,js}, J^\ell_{ku,lv} | \mathbf{L}^0\right] = \tfrac{1}{D^2_{\ell-1}} \sum_{dd'} \mathbb{C}\left[L^{\ell-1}_{i(r+d),j(s+d)}, L^{\ell-1}_{k(u+d'),l(v+d')} | \mathbf{L}^0\right] \tag{57a}$$

$$\mathbb{C}\left[K^\ell_{ir,js}, K^\ell_{ku,lv} | \mathbf{L}^0\right] \approx \mathbb{C}\left[J^\ell_{ir,js}, J^\ell_{ku,lv} | \mathbf{L}^0\right] + \tfrac{1}{N_\ell}\left(\left\langle J^\ell_{ir,ku}\right\rangle \left\langle J^\ell_{js,lv}\right\rangle + \left\langle J^\ell_{ir,lv}\right\rangle \left\langle J^\ell_{js,ku}\right\rangle\right) \tag{57b}$$

$$\mathbb{C}\left[L^\ell_{ir,js}, L^\ell_{ku,lv} | \mathbf{L}^0\right] = \mathbb{C}\left[K^\ell_{ir,js}, K^\ell_{ku,lv} | \mathbf{L}^0\right] \tag{57c}$$

Finally, to compute these terms, note that we can compute the recursions for $r = s$ and $u = v$,

$$\mathbb{C}\left[J^{\ell}_{ir,jr}, J^{\ell}_{ku,lu}|\mathbf{L}^0\right] = \tfrac{1}{D^2_{\ell-1}} \sum_{dd'} \mathbb{C}\left[L^{\ell-1}_{i(r+d),j(r+d)}, L^{\ell-1}_{k(u+d'),l(u+d')}|\mathbf{K}^0\right], \tag{58}$$

and this expression can be computed efficiently as a 2D convolution.

A.2.1 Spatially structured and unstructured networks

To understand the very different results for spatially structured and unstructured networks (Fig. 3B–D) despite their having the same infinite limit, we need to consider how Eq. (58) interacts with Eq. (55). For a locally connected (i.e. spatially unstructured) network, the covariance of activations at different locations is always zero, i.e. $J^{\ell}_{ir,js} = 0$ for $r = s$ whereas, for a spatially structured network, the covariance terms have the same scale as the variance terms. The $J_{ir,js}$ terms enter into the variance of the kernel through Eq. (58). Note that there are $D^2_{\ell-1}$ terms in this sum, and the sum is normalized by dividing by $D^2_{\ell-1}$. Thus, in spatially structured networks, there are $D^2_{\ell-1}$ terms all with the same scale, so $\mathbb{C}[J^{\ell}_{ir,jr}, J^{\ell}_{ku,lu}|\mathbf{L}^0]$ is $\mathcal{O}(1)$. In contrast, for spatially unstructured networks, we have only $D_{\ell-1}$ nonzero terms, so $\mathbb{C}[J^{\ell}_{ir,jr}, J^{\ell}_{ku,lu}|\mathbf{L}^0]$ is $\mathcal{O}(1/D_{\ell-1})$. This is particularly important if we consider the last layer. The last layer can be understood as a convolution, where the convolutional patch has the same size as the image (i.e. $D_L = S$), and there is no padding, such the the output has a single spatial location. In this case, the $1/S$ normalizer can be very large, which causes the second term in Eq. (19b) to dominate the first term, leading to the almost constant variance of the kernel as we increase network depth observed in Fig. 3D.

B Kernel flexibility: posterior viewpoint

B.1 Reparameterising finite neural networks

Swapping between a kernel representation and a feature representation is difficult if we work directly with a prior over the weights, $\mathbf{W}_{\ell} \in \mathbb{R}^{N_{\ell-1} \times N_{\ell}}$. Instead, note that as the weights are Gaussian, we can reparameterise the neural network, working instead with $\mathbf{V}_{\ell} \in \mathbb{R}^{P \times N_{\ell}}$ which has independent standard Gaussian entries. In particular, we can write the activities at the next layer using,

$$\mathbf{A}_{\ell} = \mathbf{H}_{\ell-1}\mathbf{W}_{\ell} = \mathbf{L}_{\ell}\mathbf{V}_{\ell}. \tag{59}$$

where $\mathbf{L}_{\ell} \in \mathbb{R}^{P \times P}$ is any matrix that satisfies,

$$\mathbf{J}_{\ell} = \mathbf{L}_{\ell}\mathbf{L}_{\ell}^T. \tag{60}$$

such as the Cholesky decomposition of the covariance, $\mathbf{J}_{\ell}$. We can thus write the kernel as,

$$\mathbf{K}_{\ell} = \tfrac{1}{N_{\ell}}\mathbf{A}_{\ell}\mathbf{A}_{\ell}^T = \tfrac{1}{N_{\ell}}\mathbf{H}_{\ell-1}\mathbf{W}_{\ell}\mathbf{W}_{\ell}^T\mathbf{H}_{\ell-1}^T = \tfrac{1}{N_{\ell}}\mathbf{L}_{\ell}\mathbf{V}_{\ell}\mathbf{V}_{\ell}^T\mathbf{L}_{\ell}^T. \tag{61}$$

Rearranging, we can write $\frac{1}{N_{\ell}}\mathbf{V}_{\ell}\mathbf{V}_{\ell}^T$, or equivalently the mismatch between the covariance, $\mathbf{J}_{\ell}$, and the output kernel, $\mathbf{K}_{\ell}$, in terms of $\mathbf{L}_{\ell}$ and $\mathbf{K}_{\ell}$, and we denote this quantity $\mathbf{R}_{\ell}$ for future use,

$$\mathbf{R}_{\ell} = \mathbf{L}_{\ell}^{-1}\mathbf{K}_{\ell}\mathbf{L}_{\ell}^{-T} = \tfrac{1}{N_{\ell}}\mathbf{V}_{\ell}\mathbf{V}_{\ell}^T \tag{62}$$

where $\mathbf{X}^{-T} = (\mathbf{X}^{-1})^T = (\mathbf{X}^T)^{-1}$.

B.2 MAP inference

Here, we consider MAP inference over $\mathbf{V}_{\ell}$. As the entries of $\mathbf{V}_{\ell}$ have a standard Gaussian prior, we have,

$$\log \mathrm{P}\left(\mathbf{V}_{\ell}\right) = -\tfrac{1}{2}\,\mathrm{Tr}\left(\mathbf{V}_{\ell}\mathbf{V}_{\ell}^T\right) + \mathrm{const} \tag{63}$$

$$= -\tfrac{N_{\ell}}{2}\,\mathrm{Tr}\left(\mathbf{L}_{\ell}^{-1}\mathbf{K}_{\ell}\mathbf{L}_{\ell}^{-T}\right) + \mathrm{const} \tag{64}$$

$$= -\tfrac{N_{\ell}}{2}\,\mathrm{Tr}\left(\mathbf{K}_{\ell}\mathbf{L}_{\ell}^{-T}\mathbf{L}_{\ell}^{-1}\right) + \mathrm{const} \tag{65}$$

$$= -\tfrac{N_{\ell}}{2}\,\mathrm{Tr}\left(\mathbf{K}_{\ell}\left(\mathbf{L}_{\ell}\mathbf{L}_{\ell}^T\right)^{-1}\right) + \mathrm{const} \tag{66}$$

$$= -\tfrac{N_{\ell}}{2}\,\mathrm{Tr}\left(\mathbf{K}_{\ell}\mathbf{J}_{\ell}^{-1}\right) + \mathrm{const} \tag{67}$$

We can write the likelihood in the same form,

$$\log \mathrm{P}\left(\mathbf{Y}|\mathbf{J}_{L+1}\right) = -\tfrac{1}{2}\operatorname{Tr}\left(\mathbf{Y}^T\mathbf{J}_{L+1}^{-1}\mathbf{Y}\right) + \text{const} \tag{68}$$

$$\log \mathrm{P}\left(\mathbf{Y}|\mathbf{J}_{L+1}\right) = -\tfrac{1}{2}\operatorname{Tr}\left(\mathbf{Y}\mathbf{Y}^T\mathbf{J}_{L+1}^{-1}\right) + \text{const} \tag{69}$$

$$\log \mathrm{P}\left(\mathbf{Y}|\mathbf{J}_{L+1}\right) = -\tfrac{N_{L+1}}{2}\operatorname{Tr}\left(\mathbf{K}_{L+1}\mathbf{J}_{L+1}^{-1}\right) + \text{const} \tag{70}$$

where,

$$\mathbf{K}_{L+1} = \tfrac{1}{N_{L+1}}\mathbf{Y}\mathbf{Y}^T. \tag{71}$$

Thus, the joint probability can be written as,

$$\log \mathrm{P}\left(\mathbf{V}_1, \ldots, \mathbf{V}_L, \mathbf{Y}|\mathbf{X}\right) = -\tfrac{1}{2}\sum_{\ell=1}^{L+1} N_\ell \operatorname{Tr}\left(\mathbf{K}_\ell\mathbf{J}_\ell^{-1}\right) + \text{const}\,. \tag{72}$$

Now we find, the MAP values of $\mathbf{V}_1, \ldots \mathbf{V}_L$

$$\mathbf{V}_1^*, \ldots, \mathbf{V}_L^* = \underset{\mathbf{V}_1,\ldots,\mathbf{V}_L}{\arg\max} \log \mathrm{P}\left(\mathbf{V}_1, \ldots, \mathbf{V}_L, \mathbf{Y}|\mathbf{X}\right), \tag{73}$$

by taking gradients of $\mathrm{P}\left(\mathbf{V}_1, \ldots, \mathbf{V}_L, \mathbf{Y}|\mathbf{X}\right)$ wrt $\mathbf{K}_1, \ldots, \mathbf{K}_L$. In particular, we consider a linear, fully connected network where $\mathbf{J}_\ell = \mathbf{K}_{\ell-1}$,

$$\mathbf{0} = \frac{\partial}{\partial \mathbf{K}_\ell}\log \mathrm{P}\left(\mathbf{V}_1, \ldots, \mathbf{V}_L, \mathbf{Y}|\mathbf{X}\right) = -\tfrac{N_\ell}{2}\mathbf{K}_{\ell-1}^{-1} + \tfrac{N_{\ell+1}}{2}\mathbf{K}_\ell^{-1}\mathbf{K}_{\ell+1}\mathbf{K}_\ell^{-1} \tag{74}$$

where we have used,

$$\frac{\partial \operatorname{Tr}\left(\mathbf{K}_\ell^{-1}\mathbf{K}_{\ell+1}\right)}{\partial \mathbf{K}_\ell} = -\mathbf{K}_\ell^{-1}\mathbf{K}_{\ell+1}\mathbf{K}_\ell^{-1} \tag{75}$$

$$\frac{\partial \operatorname{Tr}\left(\mathbf{K}_{\ell-1}^{-1}\mathbf{K}_\ell\right)}{\partial \mathbf{K}_\ell} = \mathbf{K}_{\ell-1}^{-1} \tag{76}$$

Thus, the MAP kernel changes as a fixed ratio,

$$\mathbf{S} = N_{\ell+1}\mathbf{K}_{\ell+1}\mathbf{K}_\ell^{-1} = N_\ell\mathbf{K}_\ell\mathbf{K}_{\ell-1}^{-1}, \tag{77}$$

As the input kernel, $\mathbf{K}_0$, and the output kernel, $\mathbf{K}_{L+1}$, are fixed we can solve for $\mathbf{S}$,

$$\mathbf{S}^{L+1} = \prod_{\ell=1}^{L+1} N_\ell\mathbf{K}_\ell\mathbf{K}_{\ell-1}^{-1} = \mathbf{K}_{L+1}\mathbf{K}_0^{-1}\prod_{\ell=1}^{L+1} N_\ell \tag{78}$$

so,

$$\mathbf{S} = \left(\mathbf{K}_{L+1}\mathbf{K}_0^{-1}\right)^{1/L+1}\left(\prod_{\ell=1}^{L+1} N_\ell\right)^{1/L+1} \tag{79}$$

where the final term is the geometric average of the width at each layer. As such, the kernel at any given layer is,

$$\mathbf{K}_\ell = \left(\prod_{\ell'=1}^{\ell} \mathbf{K}_\ell\mathbf{K}_{\ell-1}^{-1}\right)\mathbf{K}_0 \tag{80}$$

$$\mathbf{K}_\ell = \left(\prod_{\ell'=1}^{\ell} \tfrac{1}{N_{\ell'}}\mathbf{S}\right)\mathbf{K}_0 \tag{81}$$

$$\mathbf{K}_\ell = \frac{\left(\prod_{\ell'=1}^{L+1} N_{\ell'}\right)^{\ell/(L+1)}}{\prod_{\ell'=1}^{\ell} N_{\ell'}}\left(\mathbf{K}_{L+1}\mathbf{K}_0^{-1}\right)^{\ell/(L+1)}\mathbf{K}_0 \tag{82}$$

defining the geometric average of the number of units at each layer prior to (and including) ℓ, and after ℓ,

$$N_{\leq \ell} = \left(\prod_{\ell'=1}^{\ell} N_{\ell'} \right)^{1/\ell} \tag{83}$$

$$N_{\ell <} = \left(\prod_{\ell'=\ell+1}^{L+1} N_{\ell'} \right)^{1/(L+1-\ell)} \tag{84}$$

we can write,

$$\frac{\left(\prod_{\ell'=1}^{L+1} N_\ell \right)^{\ell/(L+1)}}{\prod_{\ell'=1}^{\ell} N_\ell} = \frac{\left(\left(N_{\leq \ell}\right)^{\ell} \left(N_{\ell <}\right)^{L+1-\ell} \right)^{\ell/(L+1)}}{\left(N_{\leq \ell}\right)^{\ell}} \tag{85}$$

$$= \left(\left(N_{\leq \ell}\right)^{-(L+1-\ell)} \left(N_{\ell <}\right)^{L+1-\ell} \right)^{\ell/(L+1)} \tag{86}$$

$$= \left(\frac{N_{\ell <}}{N_{\leq \ell}} \right)^{\frac{\ell(L+1-\ell)}{L+1}} \tag{87}$$

This factor is the ratio of the geometric average of the widths for the previous and subsequent layers, to a power which depends on the distance to the end points (for $\ell = 0$ or $\ell = L+1$ this factor is 1),

$$\mathbf{K}_\ell = \left(\frac{N_{\ell <}}{N_{\leq \ell}} \right)^{\frac{\ell(L+1-\ell)}{L+1}} \left(\mathbf{K}_{L+1} \mathbf{K}_0^{-1} \right)^{\ell/(L+1)} \mathbf{K}_0 \tag{88}$$

Thus, MAP does something sensible: no matter what the network widths (and including as the network widths go to infinity), the representation interpolates smoothly between the input and output kernels. However, the scale of these representations can shift in a strange, and potentially pathological fashion. Remember that we normalized the weights, taking into account the width of each layer such that the representations maintained the same scale, irrespective of layer width. However, under MAP inference, the network width controls the scale of the kernel, with larger kernels at layer ℓ given by widening layers from 1 to ℓ, and narrowing layers from $\ell + 1$ to $L + 1$.

C Deriving a cost-function such that gradient descent is equivalent to sampling

The pathologies in the above derivations indicate that MAP, using full-batch gradient descent may give a very approximation of the kernel induced by *stochastic* gradient descent. As such, we consider Langevin sampling which not only gives Bayesian inference, but also gives a good starting point for thinking about the noise introduced by stochastic gradient descent. In particular, we perform Langevin sampling over $\mathbf{V}_\ell$ (Eq. 59)

$$d\mathbf{V}_\ell = \tfrac{1}{2} dt \frac{\partial \mathcal{L}}{\partial \mathbf{V}_\ell} + d\mathbf{\Xi}_\ell, \tag{89}$$

where $d\mathbf{\Xi}_\ell$ is a matrix-valued Weiner process. Remembering that the objective is completely specified by $\mathbf{R}_\ell = \frac{1}{N_\ell} \mathbf{V}_\ell \mathbf{V}_\ell^T$, for a linear or finite-infinite network, we consider the effect of this sampling on $\mathbf{R}_\ell$,

$$\mathbb{E}\left[d\mathbf{R}_\ell | \mathbf{R}_\ell \right] = \tfrac{1}{N_\ell} d\left(\mathbf{V}_\ell \mathbf{V}_\ell^T \right) = \tfrac{1}{2N_\ell} dt \left(\frac{\partial \mathcal{L}}{\partial \mathbf{V}_\ell} \mathbf{V}_\ell^T + \mathbf{V}_\ell \frac{\partial \mathcal{L}}{\partial \mathbf{V}_\ell}^T \right) + \tfrac{1}{N_\ell} \mathbb{E}\left[d\mathbf{\Xi}_\ell d\mathbf{\Xi}_\ell^T \right]. \tag{90}$$

As the only stochasticity comes from the last term, and this term has known expectation,

$$\tfrac{1}{N_\ell} \mathbb{E}\left[d\mathbf{\Xi}_\ell d\mathbf{\Xi}_\ell^T \right] = dt\, \mathbf{I}, \tag{91}$$

We can compute the expected update, which becomes the exact update as we take $N_\ell \to \infty$,

$$\lim_{N_\ell \to \infty} \frac{d\mathbf{R}_\ell}{dt} = \mathbb{E}\left[\frac{d\mathbf{R}_\ell}{dt} \middle| \mathbf{R}_\ell \right] = \tfrac{1}{2N_\ell} \left(\left(\frac{\partial \mathcal{L}}{\partial \mathbf{V}_\ell} \right) \mathbf{V}_\ell^T + \mathbf{V}_\ell \left(\frac{\partial \mathcal{L}}{\partial \mathbf{V}_\ell} \right)^T \right) + dt \mathbf{I}. \tag{92}$$

To check that these dynamics are sensible, we consider performing Langevin sampling using the above dynamics under the zero-mean, unit-variance prior on elements of $\mathbf{V}_\ell$,

$$\mathcal{L} = -\tfrac{1}{2}\operatorname{Tr}\left(\mathbf{V}_\ell\mathbf{V}_\ell^T\right), \tag{93}$$

so the gradient is,

$$\frac{\partial\mathcal{L}}{\partial\mathbf{V}_\ell} = \frac{\partial}{\partial\mathbf{V}_\ell}\left[-\tfrac{1}{2}\operatorname{Tr}\left(\mathbf{V}_\ell\mathbf{V}_\ell^T\right)\right] = -\mathbf{V}_\ell. \tag{94}$$

Thus,

$$\mathbb{E}\left[\frac{d\mathbf{R}_\ell}{dt}\middle|\mathbf{R}_\ell\right] = \tfrac{1}{N_\ell}\mathbf{V}_\ell\mathbf{V}_\ell^T + \mathbf{I} = -\mathbf{R}_\ell + \mathbf{I}. \tag{95}$$

Now, we set the expected change in $\mathbf{R}_\ell$ equal to zero,

$$\mathbf{0} = \mathbb{E}\left[\frac{d\mathbf{R}_\ell}{dt}\right] = -\mathbb{E}\left[\mathbf{R}_\ell\right] + \mathbf{I}. \tag{96}$$

and solving for the expected value of $\mathbf{R}_\ell$,

$$\mathbb{E}\left[\mathbf{R}_\ell\right] = \mathbb{E}\left[\tfrac{1}{N_\ell}\mathbf{V}_\ell\mathbf{V}_\ell^T\right] = \mathbf{I}, \tag{97}$$

which is equal to the expected value of $\frac{1}{N_\ell}\mathbf{V}_\ell\mathbf{V}_\ell^T$ under the prior, as is necessary given that these dynamics perform exact Langevin sampling in the limit.

C.1 Langevin dynamics as the modes of an objective

We can work directly with the Langevin dynamics derived above, but this gives updates to $\mathbf{R}_\ell$ that depend on all the $\mathbf{R}$'s. Instead, we get a more flexible approach by approximating $\mathbf{R}_\ell$ and its changes as deterministic, and understanding the resulting dynamics as optimizing a loss function. We begin by rewriting gradients of the loss function wrt $\mathbf{V}_\ell$ in terms of gradient wrt $\mathbf{R}_\ell$ (omitting the layer index, ℓ for brevity),

$$\frac{\partial\mathcal{L}}{\partial V_{\alpha\beta}} = \sum_{ij}\frac{\partial\mathcal{L}}{\partial R_{ij}}\frac{\partial R_{ij}}{\partial V_{\alpha\beta}} \tag{98}$$

substituting the value of R_{ij} and computing the derivative,

$$\frac{\partial\mathcal{L}}{\partial V_{\alpha\beta}} = \tfrac{1}{N_\ell}\sum_{ij}\frac{\partial\mathcal{L}}{\partial R_{ij}}\frac{\partial}{\partial V_{\alpha\beta}}\sum_k V_{ik}V_{jk} \tag{99}$$

$$\frac{\partial\mathcal{L}}{\partial V_{\alpha\beta}} = \tfrac{1}{N_\ell}\sum_{ij}\frac{\partial\mathcal{L}}{\partial R_{ij}}\left(\delta_{\alpha i}\delta_{\beta k}V_{jk} + \delta_{\alpha j}\delta_{\beta k}V_{ik}\right) \tag{100}$$

$$\frac{\partial\mathcal{L}}{\partial V_{\alpha\beta}} = \left(\tfrac{1}{N_\ell}\sum_j\frac{\partial\mathcal{L}}{\partial R_{\alpha j}}V_{j\beta}\right) + \left(\tfrac{1}{N_\ell}\sum_i\frac{\partial\mathcal{L}}{\partial R_{i\alpha}}V_{i\beta}\right) \tag{101}$$

putting this expression back in matrix form,

$$\frac{\partial\mathcal{L}}{\partial\mathbf{V}_\ell} = \tfrac{1}{N_\ell}\left(\frac{\partial\mathcal{L}}{\partial\mathbf{R}_\ell}\mathbf{V}_\ell + \left(\frac{\partial\mathcal{L}}{\partial\mathbf{R}_\ell}\right)^T\mathbf{V}_\ell\right) \tag{102}$$

and remembering that $\mathbf{R}$, and derivatives wrt $\mathbf{R}$ are symmetric,

$$\frac{\partial\mathcal{L}}{\partial\mathbf{V}_\ell} = \frac{2}{N_\ell}\frac{\partial\mathcal{L}}{\partial\mathbf{R}_\ell}\mathbf{V}_\ell. \tag{103}$$

Substituting this into Eq. (92),

$$\mathbb{E}\left[\frac{d\mathbf{R}_\ell}{dt}\middle|\mathbf{R}_\ell\right] = \frac{1}{N_\ell^2}\left(\frac{\partial\mathcal{L}}{\partial\mathbf{R}_\ell}\mathbf{V}_\ell\mathbf{V}_\ell^T + \mathbf{V}_\ell\mathbf{V}_\ell^T\left(\frac{\partial\mathcal{L}}{\partial\mathbf{R}_\ell}\right)^T\right) + \mathbf{I} \tag{104}$$

$$= \frac{1}{N_\ell}\left(\frac{\partial\mathcal{L}}{\partial\mathbf{R}_\ell}\mathbf{R}_\ell + \mathbf{R}_\ell\frac{\partial\mathcal{L}}{\partial\mathbf{R}_\ell}\right) + \mathbf{I} \tag{105}$$

$$\tag{106}$$

Note $\mathcal{L}$ and its gradients scale linearly with N_ℓ, so the first term here does not vanish as N_ℓ grows.

We can write this expresssion as the preconditioned gradient of a modified objective, $\mathcal{L}'$,

$$\mathcal{L}' = \mathcal{L} + \tfrac{N_\ell}{2} \log |\mathbf{R}_\ell| \,. \tag{107}$$

Remembering that that $\left(\mathbf{B}^T \otimes \mathbf{A}\right) \mathrm{vec}\left(\mathbf{X}\right) = \mathrm{vec}\left(\mathbf{AXB}\right)$, where $\otimes$ is the Kronecker product, and as $(\mathbf{R}_\ell \otimes \mathbf{I} + \mathbf{I} \otimes \mathbf{R}_\ell)$ is positive definite, we can consider preconditioned gradient descent on $\mathcal{L}'$,

$$\tfrac{1}{N_\ell} \left(\mathbf{R}_\ell \otimes \mathbf{I} + \mathbf{I} \otimes \mathbf{R}_\ell\right) \mathrm{vec}\left(\frac{\partial \mathcal{L}'}{\partial \mathbf{R}}\right) = \mathrm{vec}\left(\tfrac{1}{N_\ell}\left(\frac{\partial \mathcal{L}'}{\partial \mathbf{R}_\ell}\mathbf{R}_\ell + \mathbf{R}_\ell \frac{\partial \mathcal{L}'}{\partial \mathbf{R}_\ell}\right)\right) \tag{108}$$

$$= \mathrm{vec}\left(\tfrac{1}{N_\ell}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{R}_\ell}\mathbf{R}_\ell + \mathbf{R}_\ell \frac{\partial \mathcal{L}}{\partial \mathbf{R}_\ell}\right) + \mathbf{I}\right) \tag{109}$$

$$= \mathrm{vec}\left(\mathbb{E}\left[\frac{d\mathbf{R}_\ell}{dt}\middle|\, \mathbf{R}_\ell\right]\right) \tag{110}$$

As such, the expected updates to $\mathbf{R}_\ell$ induced by Langevin sampling are equivalent to preconditioned gradient descent on a modified objective, $\mathcal{L}'$.

C.2 The sampling objective as modified maximum-likelihood under a Wishart prior

To further check that the Langevin sampling result is sensible, we note that it is very similar to doing MAP inference under a Wishart prior, but that sampling fixes pathologies in this proceedure due to the skew inherent in the Wishart distribution.

In particular, the Wishart probability density is given by,

$$\log \mathrm{P}\left(\mathbf{K}_\ell|\mathbf{J}_\ell\right) = \log \mathrm{Wishart}\left(\mathbf{K}_\ell; \tfrac{1}{N_\ell}\mathbf{J}_\ell, N_\ell\right) \tag{111}$$

$$= \tfrac{N_\ell - P - 1}{2} \log |\mathbf{K}_\ell| - \tfrac{N_\ell}{2} \log |\mathbf{J}_\ell| - \tfrac{N_\ell}{2} \mathrm{Tr}\left(\mathbf{J}_{\ell-1}^{-1}\mathbf{K}_\ell\right). \tag{112}$$

The pathologies arise if we compare the expectation and the mode of this distribution,

$$\mathbb{E}\left[\mathbf{K}_\ell|\mathbf{J}_\ell\right] = \mathbf{J}_\ell \tag{113}$$

$$\arg\max_{\mathbf{K}_\ell} \left[\log \mathrm{P}\left(\mathbf{K}_\ell|\mathbf{J}_\ell\right)\right] = \left(N_\ell - P - 1\right)\mathbf{J}_\ell, \tag{114}$$

where the matricies $\mathbf{K}_\ell$ and $\mathbf{J}_\ell$ are $P \times P$, and $\mathbf{K}_\ell$ is the inner product of N_ℓ vectors with covariance $\frac{1}{N_\ell}\mathbf{J}_\ell$. Thus, the mode gives a very poor characterisation of the expectation of the distribution, to the extent if $N_\ell = P + 1$, the mode is zero while the expectation can take on any value. Thankfully, it is possible to find a closely related optimization problem that gives a good characterisation of the mean. In particular, we need to incorporate a new term in the objective that counteracts the "shrinkage" induced by the skew in the Wishart, such that the mode of the new objective equals the expectation,

$$\arg\max_{\mathbf{K}_\ell} \left[\log \mathrm{P}\left(\mathbf{K}_\ell|\mathbf{J}_\ell\right) + \tfrac{P+1}{2} \log |\mathbf{K}_\ell|\right] = \mathbf{J}_\ell. \tag{115}$$

Critically, this term, $\frac{P+1}{2} \log |\mathbf{K}_\ell|$ is almost entirely independent of the parameters (it depends only on the size, P), and the combined objective is equivalent to the objective for Langevin sampling,

$$\log \mathrm{P}\left(\mathbf{K}_\ell|\mathbf{J}_\ell\right) + \tfrac{P+1}{2} \log |\mathbf{K}_\ell| = \tfrac{N_\ell}{2} \log \left|\mathbf{J}_\ell^{-1}\mathbf{K}_\ell\right| - \tfrac{N_\ell}{2} \mathrm{Tr}\left(\mathbf{J}_\ell^{-1}\mathbf{K}_\ell\right) \tag{116}$$

$$= \tfrac{N_\ell}{2} \log \left|\mathbf{L}_\ell^{-1}\mathbf{K}_\ell\mathbf{L}_\ell^{-T}\right| - \tfrac{N_\ell}{2} \mathrm{Tr}\left(\mathbf{L}_\ell^{-1}\mathbf{K}_\ell\mathbf{L}_\ell^{-T}\right) \tag{117}$$

$$= \tfrac{N_\ell}{2} \log |\mathbf{R}_\ell| - \tfrac{N_\ell}{2} \mathrm{Tr}\left(\mathbf{R}_\ell\right). \tag{118}$$

C.3 Representation learning in deep networks

The log-probability of the data at the final layer can be written in the same form as the objective for Langevin sampling (Eq. 107), and the modified objective for Wishart inference (Eq. 116). In particular,

$$\log \mathrm{P}\left(\mathbf{y}_\mu|\mathbf{K}_L\right) = -\tfrac{1}{2}\mathbf{y}_\mu^T\mathbf{L}_L^{-1}\mathbf{y}_\mu - \tfrac{1}{2}\left|\mathbf{K}_L\right| + \mathrm{const}\,. \tag{119}$$

defining the constant kernel, $\mathbf{K}_{L+1} = \frac{1}{Y}\mathbf{Y}\mathbf{Y}^T$, we can write the log-probability of $\mathbf{Y}$ in a manner that is consistent with the previous kernels,

$$\log \mathrm{P}\left(\mathbf{Y}|\mathbf{K}_L\right) = \tfrac{Y}{2}\left(\log\left|\mathbf{L}_L^{-1}\mathbf{K}_{L+1}\right| - \mathrm{Tr}\left(\mathbf{L}_L^{-1}\mathbf{K}_{L+1}\right)\right) + \mathrm{const} \tag{120}$$

where the log determinant of $\mathbf{K}_{L+1}$ is constant, so can be included without changing the objective.

As such, the full objective can be written as,

$$\mathcal{L} = \sum_{\ell=1}^{L+1} \tfrac{N_\ell}{2}\left(\log\left|\mathbf{L}_{\ell-1}^{-1}\mathbf{K}_\ell\right| - \mathrm{Tr}\left(\mathbf{L}_{\ell-1}^{-1}\mathbf{K}_\ell\right)\right). \tag{121}$$

When we differentiate, only the terms that vary with $\mathbf{K}_\ell$ are relevant,

$$\mathcal{L} = \tfrac{N_\ell}{2}\left(\log\left|\mathbf{L}_{\ell-1}^{-1}\mathbf{K}_\ell\right| - \mathrm{Tr}\left(\mathbf{L}_{\ell-1}^{-1}\mathbf{K}_\ell\right)\right) + \tfrac{N_{\ell+1}}{2}\left(\log\left|\mathbf{L}_\ell^{-1}\mathbf{K}_{\ell+1}\right| - \mathrm{Tr}\left(\mathbf{L}_\ell^{-1}\mathbf{K}_{\ell+1}\right)\right). \tag{122}$$

While the derivations up to this point have been the same, the gradients are different for fully connected, locally connected, and convolutional networks diverge.

C.4 Fully connected networks

For fully connected networks,

$$\mathbf{L}_\ell = \mathbf{K}_\ell. \tag{123}$$

so the terms in the objective that depend on $\mathbf{K}_\ell$ are,

$$\mathcal{L} = \tfrac{N_\ell}{2}\left(\log\left|\mathbf{K}_{\ell-1}^{-1}\mathbf{K}_\ell\right| - \mathrm{Tr}\left(\mathbf{K}_{\ell-1}^{-1}\mathbf{K}_\ell\right)\right) + \tfrac{N_{\ell+1}}{2}\left(\log\left|\mathbf{K}_\ell^{-1}\mathbf{K}_{\ell+1}\right| - \mathrm{Tr}\left(\mathbf{K}_\ell^{-1}\mathbf{K}_{\ell+1}\right)\right). \tag{124}$$

Differentiating the relevant terms,

$$\frac{\partial \,\mathrm{Tr}\,\mathbf{K}_\ell^{-1}\mathbf{K}_{\ell+1}}{\partial \mathbf{K}_\ell} = -\mathbf{K}_\ell^{-1}\mathbf{K}_{\ell+1}\mathbf{K}_\ell^{-1} \tag{125a}$$

$$\frac{\partial \,\mathrm{Tr}\,\mathbf{K}_{\ell-1}^{-1}\mathbf{K}_\ell}{\partial \mathbf{K}_\ell} = \mathbf{K}_{\ell-1}^{-1} \tag{125b}$$

$$\frac{\partial \log\left|\mathbf{K}_{\ell-1}^{-1}\mathbf{K}_\ell\right|}{\partial \mathbf{K}_\ell} = \frac{\partial \log|\mathbf{K}_\ell|}{\partial \mathbf{K}_\ell} = \mathbf{K}_\ell^{-1} \tag{125c}$$

$$\frac{\partial \log\left|\mathbf{K}_\ell^{-1}\mathbf{K}_{\ell+1}\right|}{\partial \mathbf{K}_\ell} = -\frac{\partial \log|\mathbf{K}_\ell|}{\partial \mathbf{K}_\ell} = -\mathbf{K}_\ell^{-1} \tag{125d}$$

We then set the gradients to zero,

$$\mathbf{0} = \frac{\partial \mathcal{L}}{\partial \mathbf{K}_\ell} = -\left(N_{\ell+1} - N_\ell\right)\mathbf{K}_\ell^{-1} + N_{\ell+1}\,\mathbf{K}_\ell^{-1}\mathbf{K}_{\ell+1}\mathbf{K}_\ell^{-1} - N_\ell\,\mathbf{K}_{\ell-1}^{-1} \tag{126}$$

We pre multiply by $\mathbf{K}_\ell$,

$$\mathbf{0} = -\left(N_{\ell+1} - N_\ell\right)\mathbf{I} + N_{\ell+1}\,\mathbf{K}_{\ell+1}\mathbf{K}_\ell^{-1} - N_\ell\,\mathbf{K}_\ell\mathbf{K}_{\ell-1}^{-1}, \tag{127}$$

And note that the resulting expression can be written in terms of a ratio, $\mathbf{R}_{\ell+1} = \mathbf{K}_{\ell+1}\mathbf{K}_\ell^{-1}$

$$\mathbf{0} = -\left(N_{\ell+1} - N_\ell\right)\mathbf{I} + N_{\ell+1}\,\mathbf{R}_{\ell+1} - N_\ell\,\mathbf{R}_\ell. \tag{128}$$

Solving for $\mathbf{R}_{\ell+1}$,

$$\mathbf{R}_{\ell+1} = \mathbf{I} + \tfrac{N_\ell}{N_{\ell+1}}\left(\mathbf{R}_\ell - \mathbf{I}\right) \tag{129}$$

We use $N_\ell = N$, for $\ell \in \{1, \ldots, N\}$,and $N_{L+1} = Y$,

$$\mathbf{R}_\ell = \begin{cases} \mathbf{R} & \text{for } \ell \in \{1, \ldots, L\} \\ \mathbf{I} + \frac{N}{Y}\left(\mathbf{R} - \mathbf{I}\right) & \text{for } \ell = L+1 \end{cases} \tag{130}$$

to compute $\mathbf{R}$, we use,

$$\mathbf{K}_{L+1}\mathbf{K}_0^{-1} = \mathbf{R}_{L+1}\mathbf{R}^L, \tag{131}$$

substituting for $\mathbf{R}_{L+1}$,

$$\mathbf{K}_{L+1}\mathbf{K}_0^{-1} = \left(\mathbf{I} + \tfrac{N}{Y}\left(\mathbf{R} - \mathbf{I}\right)\right)\mathbf{R}^L \tag{132}$$

As this cannot be solved analytically for $\mathbf{R}$, we consider three special cases. First, if there are many outputs in comparison to the number of hidden units (i.e. $N/Y \to 0$),

$$\lim_{N/Y\to 0}\mathbf{R} = \left(\mathbf{K}_{L+1}\mathbf{K}_0^{-1}\right)^{1/L} \tag{133}$$

and thus, the top-level kernel is equal to the output kernel, i.e. $\mathbf{K}_L = \mathbf{K}_{L+1}$. Second, we consider the other extreme where there are many more hidden units than output channels (i.e. $N/Y \to \infty$). In this limit, we must have $\mathbf{0} = \mathbf{R} - \mathbf{I}$ because otherwise the $\frac{N}{Y}\left(\mathbf{R} - \mathbf{I}\right)$ term will explode,

$$\lim_{N/Y\to\infty}\mathbf{R} = \mathbf{I}, \tag{134}$$

thus, the representation does not change as it flows throught the network. Finally, we consider a more reasonable case where the number of hidden units is of the order of the number of output channels — in particular, we consider $Y = N$,

$$\mathbf{R} = \left(\mathbf{K}_{L+1}\mathbf{K}_0^{-1}\right)^{1/(L+1)} \tag{135}$$

as such, the top-layer kernel is almost — but not quite — equal to the output kernel, but it does get closer as the network gets deeper,

$$\mathbf{K}_L = \mathbf{R}^L\mathbf{K}_0 = \left(\mathbf{K}_{L+1}\mathbf{K}_0^{-1}\right)^{L/(L+1)}\mathbf{K}_0 = \mathbf{K}_{L+1}^{L/(L+1)}\mathbf{K}_0^{1/L} \tag{136}$$

D NATURAL GRADIENTS FOR A GAUSSIAN-PROCESS SUM KERNEL

We begin by defining the covariance (kernel) as the sum over a set of kernels, $\mathbf{K}_i$, weighted by λ_i,

$$\mathbf{K} = \sum_i \lambda_i \mathbf{K}_i. \tag{137}$$

Our goal is to find the maximum-likelihood λ_i parameters using a natural-gradient method. The likelihood is,

$$\log \mathrm{P}\left(\mathbf{Y}\right) = -\tfrac{1}{2}\operatorname{Tr}\left(\mathbf{K}^{-1}\mathbf{Y}\mathbf{Y}^T\right) - \tfrac{N}{2}\log|\mathbf{K}| + \text{const}. \tag{138}$$

$$\tag{139}$$

And the gradient is,

$$\frac{\partial \log \mathrm{P}\left(\mathbf{Y}\right)}{\partial \lambda_\alpha} = \tfrac{1}{2}\operatorname{Tr}\mathbf{L}_\alpha\mathbf{L}_\mathrm{y} - \tfrac{N}{2}\operatorname{Tr}\mathbf{L}_\alpha. \tag{140}$$

where,

$$\mathbf{L}_\alpha = \mathbf{K}^{-1}\mathbf{K}_\alpha \tag{141}$$

$$\mathbf{L}_\mathrm{y} = \mathbf{K}^{-1}\mathbf{Y}\mathbf{Y}^T \tag{142}$$

For a natural-gradient method, need about the expected-second-derivatives. For the first term, these are,

$$\mathbb{E}\left[\frac{\partial^2}{\partial\lambda_\alpha\lambda_\beta}\left[\tfrac{1}{2}\operatorname{Tr}\mathbf{L}_\alpha\mathbf{L}_\mathrm{y}\right]\right] = \mathbb{E}\left[-\tfrac{1}{2}\left(\operatorname{Tr}\mathbf{L}_\beta\mathbf{L}_\alpha\mathbf{L}_\mathrm{y} + \operatorname{Tr}\mathbf{L}_\alpha\mathbf{L}_\beta\mathbf{L}_\mathrm{y}\right)\right] \tag{143}$$

$$= -\tfrac{1}{2}\left(\operatorname{Tr}\mathbf{L}_\beta\mathbf{L}_\alpha\,\mathbb{E}\left[\mathbf{L}_\mathrm{y}\right] + \operatorname{Tr}\mathbf{L}_\alpha\mathbf{L}_\beta\,\mathbb{E}\left[\mathbf{L}_\mathrm{y}\right]\right) \tag{144}$$

$$= -\tfrac{N}{2}\left(\operatorname{Tr}\mathbf{L}_\beta\mathbf{L}_\alpha + \operatorname{Tr}\mathbf{L}_\alpha\mathbf{L}_\beta\right) \tag{145}$$

$$= -N\operatorname{Tr}\mathbf{L}_\alpha\mathbf{L}_\beta \tag{146}$$

using basic matrix identities, and the fact that, under the model, $\mathbb{E}\left[\mathbf{L}_\mathrm{y}\right] = N\mathbf{I}$. The second term is independent of $\mathbf{Y}$, so we can just compute the second derivative,

$$\frac{\partial^2}{\partial\lambda_\alpha\lambda_\beta}\left[-\tfrac{N}{2}\operatorname{Tr}\mathbf{L}_\alpha\right] = \tfrac{N}{2}\operatorname{Tr}\mathbf{L}_\beta\mathbf{L}_\alpha.$$

Thus,

$$\mathbb{E}\left[\frac{\partial^2}{\partial\lambda_\alpha\lambda_\beta}\log \mathrm{P}\left(\mathbf{Y}\right)\right] = -\tfrac{N}{2}\operatorname{Tr}\mathbf{L}_\alpha\mathbf{L}_\beta \tag{147}$$