# Why bigger is not always better: on finite and infinite neural networks

**Laurence Aitchison** [1]

## Abstract

Recent work has argued that neural networks can be understood theoretically by taking the number of channels to infinity, at which point the outputs become Gaussian process (GP) distributed. However, we note that infinite Bayesian neural networks lack a key facet of the behaviour of real neural networks: the fixed kernel, determined only by network hyperparameters, implies that they cannot do any form of representation learning. The lack of representation or equivalently kernel learning leads to less flexibility and hence worse performance, giving a potential explanation for the inferior performance of infinite networks observed in the literature (e.g. Novak et al. 2019). We give analytic results characterising the prior over representations and representation learning in finite deep linear networks. We show empirically that the representations in SOTA architectures such as ResNets trained with SGD are much closer to those suggested by our deep linear results than by the corresponding infinite network. This motivates the introduction of a new class of network: infinite networks with bottlenecks, which inherit the theoretical tractability of infinite networks while at the same time allowing representation learning.

One approach to understanding and improving neural networks is to perform Bayesian inference in an infinitely wide network (Lee et al., 2018; Matthews et al., 2018; 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) (Li et al., 2019) has suggested that deterministic gradient descent in an infinite network gives slighly lower performance than Bayesian inference in the same network.

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 (Huh et al., 2016) we use a large-scale dataset 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 infinite Bayesian neural networks, because the top-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 from the infinite network literature (Lee et al., 2018; Matthews et al., 2018; Garriga-Alonso et al., 2019; Novak et al., 2019). We begin by giving a toy, two-layer example, contrasting the flexibility of finite networks with the inflexibility of infinite networks, showing that flexible finite networks offer benefits under conditions of model-mismatch. We then introduce infinite networks with bottlenecks, which combine the theoretical tractability of infinite networks with the flexibility of finite networks. To obtain an analytic understanding of kernel/representation flexibility and learning in such networks, we consider linear infinite networks with bottlenecks, which are equivalent to finite deep linear networks. We took two approaches to characterising these networks. 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

[1]University of Bristol, Bristol, UK. Correspondence to: Laurence Aitchison <laurence.aitchison@gmail.com>.

*Proceedings of the $37^{th}$ International Conference on Machine Learning*, Vienna, Austria, PMLR 119, 2020. Copyright 2020 by the author(s).

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 posterior 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 the labels). 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. Bayesian networks behave similarly when the network width and the number of output channels are equal, but as the network width increases, the learned representations become increasingly dominated by the prior, and insensitive to the outputs. Remarkably, we find that in a ResNet trained using SGD on CIFAR-10, the representation differs dramatically from the corresponding infinite network and is instead very close to the output kernel, as suggested by our deep linear results. This confirms the importance of working with a theoretical model, such as infinite networks with bottlenecks, that is capable of capturing representation learning.

## 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 $\mathbf{Y}$,

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

where $\mathbf{\Xi}$ 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,

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

and where the variance of the weights is normalised by the number of inputs to that layer, $X = 4$ for the 4-dimensional input, and $H$ 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, with $H_{\text{gen}} \in \{1, 2, 4\}$ 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 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. In particular, we again generated target outputs by sampling a "true" network from the prior, but we modified the inputs to this network, first by multipling those inputs by 100 (Fig. 1 middle), then by zeroing-out all but the first input unit (Fig. 1 right). Critically, we ensured model-mismatch by putting the original, unmodified inputs into the trained networks. 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 for any given channel, $\mathbf{y}_\nu$, is IID and depends only on the corresponding column of the output weights, $\mathbf{v}_\nu$,

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

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

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

This 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,

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

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

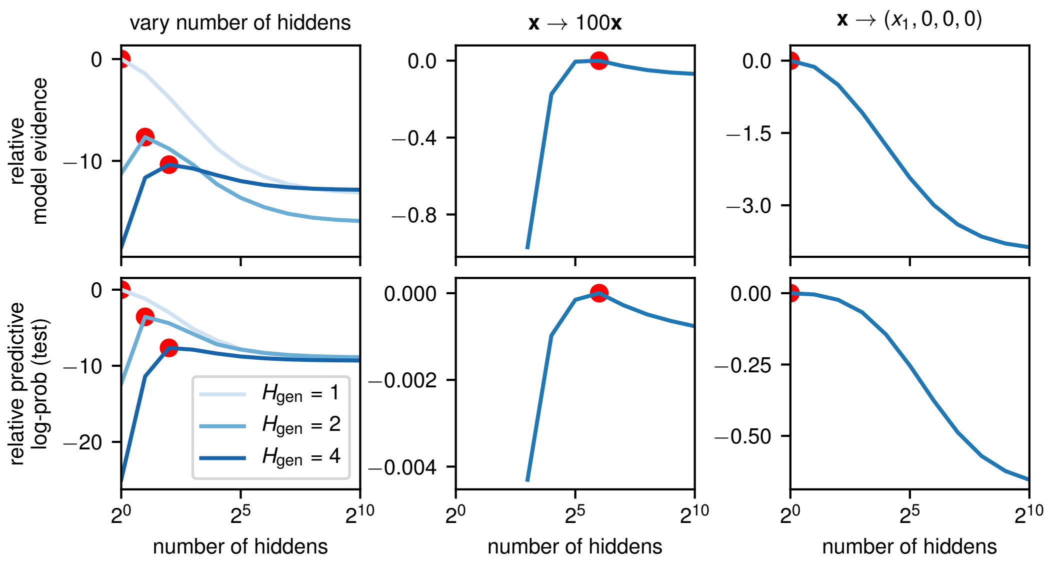

*Figure 1.* A toy fully-connected, two-layer Bayesian linear network showing situations in which smaller networks perform better than larger networks. The red dots indicate the optimal number of hidden units in that simulation. Left: training data generated from the prior with $H_{\text{gen}}$ hidden 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.

integrate over $\mathbf{W}$,

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

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 wider hidden layers networks. In particular,

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

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

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

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. Finally, note that sampling from the prior works well here both because of the concentration result above, and because we use relatively small amount of data, 20 points.

## 2. Infinite networks with finite bottlenecks

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 full infinite network with bottlenecks and show that activity flowing through this network can be understood entirely in terms of kernel and covariance matricies.

Consider a single layer within a fully-connected network, where the potentially infinite 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 a finite number of activations, $\mathbf{A}_\ell$. This activation matrix, $\mathbf{A}_\ell$ is multiplied by another matrix, $\mathbf{M}_\ell$, to give a potentially infinite updated activation matrix, $\mathbf{A}'_\ell$, which is then passed through a non-linearity, $\phi$, to give the potentially infinite activity at this layer, $\mathbf{H}_\ell$. Note that following Matthews et al. (2018), we use "activation" pre-nonlinearity and "activity" post-nonlinearity.

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

$$\begin{tikzcd}
\overbrace{\mathbf{H}_0}^{P\times M_0} = \mathbf{X} \arrow[r] & \overbrace{\mathbf{A}_\ell}^{P\times N_\ell} = \mathbf{H}_{\ell-1}\mathbf{W}_\ell \arrow[r] & \overbrace{\mathbf{A}'_\ell}^{P\times M_\ell} = \mathbf{A}_\ell\mathbf{M}_\ell \arrow[r] & \overbrace{\mathbf{H}_\ell}^{P\times M_\ell} = \phi(\mathbf{A}'_\ell) \arrow[ll, bend right] \\
\underbrace{\mathbf{L}_0 = \tfrac{1}{M_0}\mathbf{X}\mathbf{X}^T}_{\text{input kernel}} & \underbrace{\mathbf{J}_\ell = \mathbb{C}\left[\mathbf{a}_\mu^\ell|\mathbf{H}_{\ell-1}\right]}_{\text{covariance}} & \underbrace{\mathbf{K}_\ell = \tfrac{1}{N_\ell}\mathbf{A}_\ell\mathbf{A}_\ell^T = \tfrac{1}{M_\ell}\mathbf{A}'_\ell\mathbf{A}'^T_\ell}_{\text{activation kernel}} & \underbrace{\mathbf{L}_\ell = \tfrac{1}{M_\ell}\mathbf{H}_\ell\mathbf{H}_\ell^T}_{\text{activity kernel}}
\end{tikzcd}$$

*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 $1/M_\ell$.

where the input data is $\mathbf{H}_0 = \mathbf{X}$, and

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

For an infinite network with bottlenecks, we take the limit as $M_\ell$ goes to infinity, leaving $N_\ell$ finite. As such, the activity before, $\mathbf{A}'_\ell$, and after, $\mathbf{H}_\ell$, the nonlinearity is infinite, with a finite linear bottleneck formed by $\mathbf{A}_\ell$.

For a fully-connected network, the columns of $\mathbf{W}_\ell$ and $\mathbf{M}_\ell$, denoted $\mathbf{w}_\lambda^\ell$ and $\mathbf{m}_\lambda^\ell$ are generated IID from a Gaussian distribution,

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

$$\mathrm{P}\left(\mathbf{M}_\ell\right) = \prod_{\lambda=1}^{M_\ell} \mathrm{P}\left(\mathbf{m}_\lambda^\ell\right) = \prod_{\lambda=1}^{M_\lambda^\ell} \mathcal{N}\left(\mathbf{m}_\lambda^\ell; \mathbf{0}, \tfrac{1}{N_\ell}\mathbf{I}\right). \tag{12}$$

where the normalization constants, $1/M_{\ell-1}$ and $1/N_{\ell-1}$, ensure that activations remain normalized as they flow through the network.

Following the infinite network literature, we would like to characterise activity flowing through the network in terms of the activation kernel, $\mathbf{K}_\ell$ and activity kernel, $\mathbf{L}_\ell$,

$$\begin{aligned}
&\mathbf{K}_\ell \equiv \tfrac{1}{N_\ell}\mathbf{A}_\ell\mathbf{A}_\ell^T \\
&\mathbf{L}_\ell \equiv \tfrac{1}{M_\ell}\mathbf{H}_\ell\mathbf{H}_\ell^T \qquad \mathbf{L}_0 = \tfrac{1}{M_0}\mathbf{X}\mathbf{X}^T
\end{aligned} \tag{13}$$

We begin by characterising the relationship between $\mathbf{A}'_\ell$ and $\mathbf{A}_\ell$. As each channel (column) of $\mathbf{A}'_\ell$ is a linear function of the corresponding channel of the weights, $\mathbf{a}'^\ell_\lambda = \mathbf{A}_\ell\mathbf{m}_\lambda^\ell$, these activations are Gaussian and IID conditioned on $\mathbf{A}_\ell$,

$$\begin{aligned}
\mathrm{P}\left(\mathbf{A}'_\ell|\mathbf{A}_\ell\right) &= \prod_{\lambda=1}^{M_\ell} \mathrm{P}\left(\mathbf{a}'^\ell_\lambda|\mathbf{A}_\ell\right) \\
&= \prod_{\lambda=1}^{M_\ell} \mathcal{N}\left(\mathbf{a}'^\ell_\lambda; \mathbf{0}, \mathbf{K}_\ell\right) = \mathrm{P}\left(\mathbf{A}'_\ell|\mathbf{K}_\ell\right)
\end{aligned} \tag{14}$$

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

$$\begin{aligned}
\lim_{M_\ell\to\infty} \tfrac{1}{M_\ell}\mathbf{M}_\ell\mathbf{M}_\ell^T &= \tfrac{1}{N_\ell}\mathbf{I} \\
\lim_{M_\ell\to\infty} \tfrac{1}{M_\ell}\mathbf{A}'_\ell\left(\mathbf{A}'_\ell\right)^T &= \lim_{M_\ell\to\infty} \tfrac{1}{M_\ell}\mathbf{A}_\ell\mathbf{M}_\ell\mathbf{M}_\ell^T\mathbf{A}_\ell^T \\
&= \tfrac{1}{N_\ell}\mathbf{A}_\ell\mathbf{A}_\ell^T = \mathbf{K}_\ell,
\end{aligned} \tag{15}$$

Thus, the kernel for $\mathbf{A}_\ell$ is equivalent to the kernel for $\mathbf{A}'_\ell$ in infinite networks with finite bottlenecks (Fig. 2).

Next, consider computing $\mathbf{K}_\ell$ from $\mathbf{L}_{\ell-1}$. As each channel (column) of the activations is a linear function of the corresponding channel of the weights, $\mathbf{a}_\lambda^\ell = \mathbf{H}_{\ell-1}\mathbf{w}_\lambda^\ell$, the activations are Gaussian and IID conditioned on the activity at the previous layer,

$$\begin{aligned}
\mathrm{P}\left(\mathbf{A}_\ell|\mathbf{H}_{\ell-1}\right) &= \prod_{\mu=1}^{N_\ell} \mathrm{P}\left(\mathbf{a}_\lambda^\ell|\mathbf{H}_{\ell-1}\right) \\
&= \prod_{\mu=1}^{N_\ell} \mathcal{N}\left(\mathbf{a}_\lambda^\ell; \mathbf{0}, \mathbf{J}_\ell\right) = \mathrm{P}\left(\mathbf{A}_\ell|\mathbf{J}_\ell\right),
\end{aligned} \tag{16}$$

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

$$\mathbf{J}_\ell = \mathbf{L}_{\ell-1} = \tfrac{1}{M_{\ell-1}}\mathbf{H}_{\ell-1}\mathbf{H}_{\ell-1}^T, \tag{17}$$

but the relationship is more complex in convolutional architectures (Garriga-Alonso et al., 2019; Novak et al., 2019) (Appendix A.2). As $\mathbf{A}_\ell$ is always finite and random, $\mathbf{K}_\ell$ is also a random variable, and inspecting the above expressions, its distribution can be written as a Wishart, centered on $\mathbf{L}_{\ell-1}$.

Finally consider computing $\mathbf{L}_\ell$ from $\mathbf{K}_\ell$. Note that as both $\mathbf{A}'_\ell$ and $\mathbf{H}_\ell$ are infinite, we can directly use standard results from infinite neural networks, i.e. those from Cho & Saul (2009), as in Lee et al. (2018); Matthews et al. (2018); Garriga-Alonso et al. (2019); Novak et al. (2019).

Linear infinite networks with finite bottlenecks can be obtained by setting $\mathbf{H}_\ell = \phi(\mathbf{A}'_\ell) = \mathbf{A}'_\ell$, implying that

$\mathbf{L}_\ell = \mathbf{K}_\ell$. Critically, this is equivalent to a deep linear network obtained by in adddition setting $\mathbf{M}_\ell = \mathbf{I}$ so that $\mathbf{A}'_\ell = \mathbf{A}_\ell$ and $M_\ell = N_\ell$, as these choices imply that the $\mathbf{A}_\ell = \mathbf{A}'_\ell = \mathbf{H}_\ell$ so that again, $\mathbf{L}_\ell = \mathbf{K}_\ell$.

### 2.1. DNNs are deep GPs

Given this setup, we can see that even a finite nonlinear network (i.e. with $\mathbf{M}_\ell = \mathbf{I}$) is a deep Gaussian process. In particular, in a deep Gaussian process, the activations at layer $\ell$, denoted $\mathbf{A}_\ell$, consist of $N_\ell$ IID channels that are Gaussian-process distributed (Eq. 16), with a kernel/covariance determined by the activations at the previous layer. For a fully connected network,

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

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.

## 3. The prior view on kernel flexibility

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 $\mathbf{L}_\ell$. If the prior gives a stochastic kernel with higher variance, then it will be easier to shape that kernel by conditioning on data. In the appendix, we derive recursive updates for deep, linear, convolutional networks, but here, for simplicity we give the fully-connected updates,

$$\begin{aligned}
\mathbb{C}\left[J^\ell_{ij}, J^\ell_{kl} | \mathbf{L}^0\right] &= \mathbb{C}\left[L^{\ell-1}_{ij}, L^{\ell-1}_{kl} | \mathbf{K}^0\right] && (19a)\\
\mathbb{C}\left[K^\ell_{ij}, K^\ell_{kl} | \mathbf{L}^0\right] &\approx \mathbb{C}\left[J^\ell_{ij}, J^\ell_{kl} | \mathbf{L}^0\right] && (19b)\\
&\quad + \tfrac{1}{N_\ell}\left(\left\langle J^\ell_{ik}\right\rangle \left\langle J^\ell_{jl}\right\rangle + \left\langle J^\ell_{il}\right\rangle \left\langle J^\ell_{jk}\right\rangle\right) &&\\
\mathbb{C}\left[L^\ell_{ij}, L^\ell_{kl} | \mathbf{L}^0\right] &= \mathbb{C}\left[K^\ell_{ij}, K^\ell_{kl} | \mathbf{L}^0\right] && (19c)
\end{aligned}$$

where,

$$\left\langle J^\ell_{ij}\right\rangle = \mathbb{E}\left[J^\ell_{ij} | \mathbf{L}^0\right] = L^0_{ij} \qquad (19d)$$

where $i$, $j$, $k$ and $l$ index datapoints.

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, $N$,

$$\mathbb{C}\left[K^{L+1}_{ij}, K^{L+1}_{kl} | \mathbf{L}^0\right] \approx \tfrac{L+1}{N}\left(L^0_{ik} L^0_{jl} + L^0_{il} L^0_{jk}\right). \qquad (20)$$

This expression is so simple because, for a fully-connected linear network, the expected covariance at each layer is the same. For nonlinear and convolutional or locally-connected networks the covariance is still proportional to $1/N$, but the depth-dependence becomes more complex, as the covariance changes as it propagates through layers.

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. In essence, for a locally connected network, we sample the weights for each spatial region independently, so we in effect average over more IID random variables, reducing the variance of the kernel at the next layer, and hence reducing the possibility for data to shape that representation. In contrast, for a convolutional network we share weights across locations, increasing the variance in the kernel, and hence increasing the possibility for data to shape the representation. 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).

## 4. The posterior view on kernel flexibility

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,

$$\mathbf{K}_0 = \mathbf{L}_0 = \tfrac{1}{M_0}\mathbf{X}\mathbf{X}^T, \qquad (21)$$

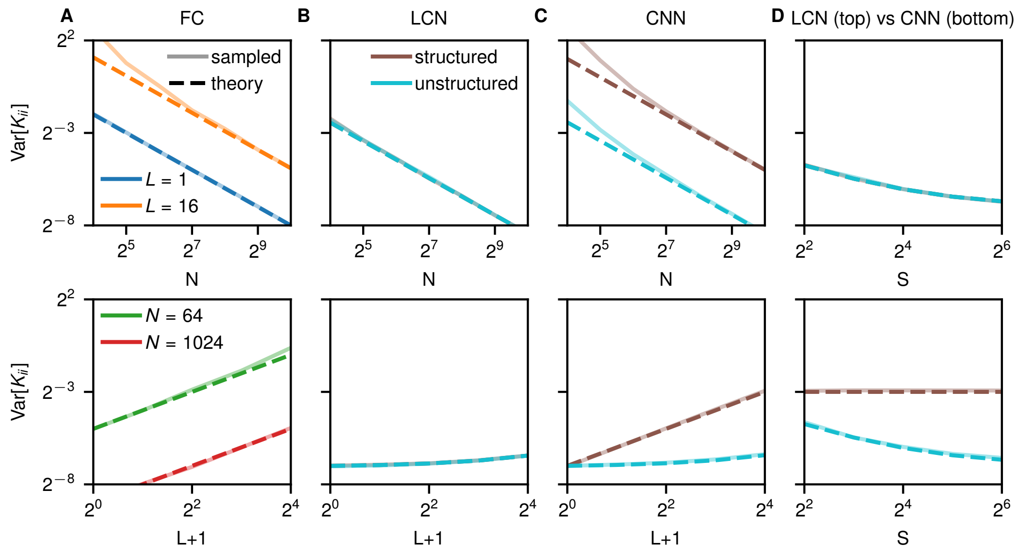

*Figure 3.* The variance of the stochastic kernel induced by randomly sampling weights in finite, linear, fully connected and convolutional networks, with spatially structured and unstructured inputs. We use normalized inputs and circular convolutions to ensure that the kernel's expected value remains equal to 1 at all locations as it propagates through the network. 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.

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 $\ell$. 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 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 widths, from $N_1$ to $N_L$ to $N$, then we get a similar expression,

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

where the kernels slowly transition from $\mathbf{K}_0$ to $\mathbf{K}_L$. The key difference is that the similarity between the top-layer representation, $\mathbf{K}_L$, and the output kernel, $\mathbf{K}_{L+1}$, 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,

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

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,

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

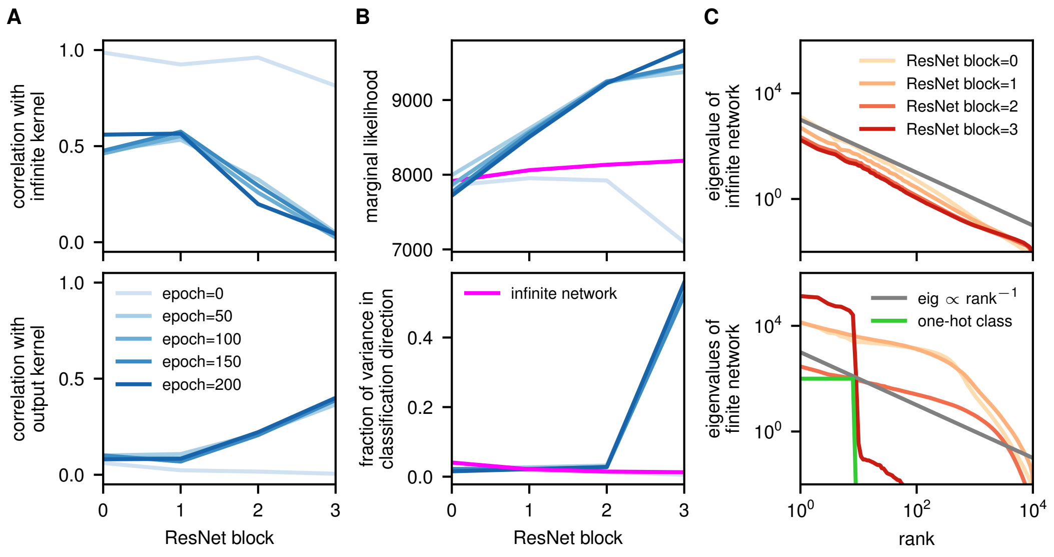

*Figure 4.* Comparison of kernels for finite and infinite neural networks at different layers. All kernels are computed on test data. **A** (top) Correlation (coefficient) between the kernel defined by the infinite network, and kernel defined by a finite network after different numbers of training epochs. **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 10 functions given by the one-hot class labels, evaluated using the kernel output by different ResNet blocks. **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 ResNet blocks.

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

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 ResNet blocks (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 time for random sampling to build up discrepancies. 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 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. Combined, these results indicate that the top-layer representation is much closer to the output kernel, as suggested by the deep linear results, than it is to the corresponding infinite network. 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, under a GP, with a kernel given by a scaled sum of the kernel at that ResNet block, and the identity (see Appendix D; Fig. 4B top). For the infinite network, the marginal likelihood increased somewhat as we moved through network layers, and the untrained finite network performed similarly, except that there was a decrease 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 grew rapidly as we move through ResNet blocks.

To gain an insight into how training shaped the neural network kernels, we computed the variance in the subspace defined by the one-hot outputs (i.e. the classification directions; Fig. 4B bottom). We might have expected to see a steady increase in the variance in this subspace as we move through layers, but in fact the level was very small, only rising appreciably at the final block, and only for trained networks. To try to understand these results, we computed the eigenvalue spectrum of the kernels. 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 $10 - 1$ large and roughly equally-sized eigenvalues, followed by much smaller eigenvalues, which mirrors the spectrum of the output kernel. This again confirms that the top-layer representation in trained networks is much closer to the output kernel than it is to corresponding infinite network.

## 5. Related work

Agrawal et al. (2020) independently introduced infinite networks with finite bottlenecks, but then made a very different contribution in that context. In particular, they highlighted that if we take the limit as some layers of a neural network go to infinity, convergence to the infinite networks with bottlenecks considered here is not immediate, but requires the neural network components to exhibit sufficient uniformity with respect to their inputs. In contrast, we show that finite bottlenecks can introduce flexibility and thereby improve performance even in two-layer linear networks, give analytic results in the case of linear networks, and show that these considerations are likely to be important in realistic large-scale networks, by showing that the kernel for a trained ResNet differs dramatically from that for the corresponding infinite network.

Technically, our work bears similarity to classical work on the dynamics of gradient descent in unregularised deep linear networks (Saxe et al., 2013). Importantly, the lack of regularisation in this work implies that infinitely many optimal solutions are available (e.g. all the lower-layer weights being fixed to the identity). In contrast, we focused on Bayesian inference, but also considered the optimal solution for regularised networks, which are much more constrained.

## 6. 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. Thus, we introduced infinite networks with bottlenecks, and argue that they may be as incorporate flexibility and are able to perform representation learning, they may be a better model of real neural networks. We then assessed the flexibility of deep linear networks 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 neural networks increase in width, 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.

## Acknowledgements

I would like to thank Adri Garriga-Alonso, Sebastian Ober and Vidhi Lalchand for useful discussions.

## References

Agrawal, D., Papamarkou, T., and Hinkle, J. Wide neural networks with bottlenecks are deep gaussian processes. *arXiv preprint arXiv:2001.00921*, 2020.

Arora, S., Du, S. S., Hu, W., Li, Z., Salakhutdinov, R., and Wang, R. On exact computation with an infinitely wide neural net. *arXiv preprint arXiv:1904.11955*, 2019.

Bui, T., Hernández-Lobato, D., Hernandez-Lobato, J., Li, Y., and Turner, R. Deep gaussian processes for regression using approximate expectation propagation. In *International Conference on Machine Learning*, pp. 1472–1481, 2016.

Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. In *Advances in neural information processing systems*, pp. 6571–6583, 2018.

Cho, Y. and Saul, L. K. Kernel methods for deep learning. *NeurIPS*, 2009.

Garriga-Alonso, A., Rasmussen, C. E., and Aitchison, L. Deep convolutional networks as shallow Gaussian processes. *ICLR*, 2019.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016.

Huh, M., Agrawal, P., and Efros, A. A. What makes imagenet good for transfer learning? *arXiv preprint arXiv:1608.08614*, 2016.

Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J., and Sohl-Dickstein, J. Deep neural networks as Gaussian processes. *ICLR*, 2018.

Li, Z., Wang, R., Yu, D., Du, S. S., Hu, W., Salakhutdinov, R., and Arora, S. Enhanced convolutional neural tangent kernels. *arXiv preprint arXiv:1911.00809*, 2019.

Matthews, A., Rowland, M., Hron, J., Turner, R., and Ghahramani, Z. Gaussian process behaviour in wide deep neural networks. *ICLR*, 2018.

Novak, R., Xiao, L., Bahri, Y., Lee, J., Yang, G., Hron, J., Abolafia, D. A., Pennington, J., and Sohl-Dickstein, J. Bayesian deep convolutional networks with many channels are Gaussian processes. *ICLR*, 2019.

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

Rasmussen, C. E. and Williams, C. K. *Gaussian processes for machine learning*. MIT press, 2006.

Saxe, A. M., McClelland, J. L., and Ganguli, S. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

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

Zagoruyko, S. and Komodakis, N. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

## A. Kernel flexibility: prior viewpoint

To compute the covariance, which we denote $\mathbb{C}\left[\cdot\right]$ 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.

Eq. (28b) 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. 13) into the definition of the covariance,

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

As the $a$'s are jointly Gaussian, 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, the covariance of the kernel becomes,

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

Substituting this into the second term in Eq. (28b)

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

and writing the expected product in terms of the product of expectations and covariance,

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

where,

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

Substituting Eq. (29) and Eq. (34) into Eq. (28b), we obtain

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

### A.1. Fully connected network

Now we evaluate Eq. (28a) first for a fully connected network,

$$a^\ell_{\lambda,i} = \sum_\mu h^{\ell-1}_{i,\mu} W^\ell_{\mu,\lambda} \tag{37}$$

where the weights are drawn from an independent zero-mean Gaussian, such that

$$E\left[W^\ell_{\mu,\lambda} W^\ell_{\nu,\lambda}\right] = \tfrac{1}{N_{\ell-1}} \delta_{\mu,\nu} \tag{38}$$

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

$$\mathrm{P}\left(\mathbf{a}^\ell_\lambda\right) = \mathcal{N}\left(\mathbf{a}^\ell_\lambda; \mathbf{0}, \mathbf{J}^\ell\right), \tag{39}$$

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

$$\begin{align}
J^\ell_{ij} &= \mathbb{C}\left[a^\ell_{i,\lambda}, a^\ell_{j,\lambda}\right] = \mathbb{E}\left[a^\ell_{i,\lambda} a^\ell_{j,\lambda}\right] \tag{40}\\
&= \mathbb{E}\left[\left(\sum_\mu h^{\ell-1}_{i,\mu} W^\ell_{\mu,\lambda}\right)\left(\sum_\nu h^{\ell-1}_{j,\nu} W^\ell_{\nu,\lambda}\right)\right] \tag{41}\\
&= \sum_{\mu\nu} h^{\ell-1}_{i,\mu} h^{\ell-1}_{j,\nu}\, \mathbb{E}\left[W^\ell_{\mu,\lambda} W^\ell_{\nu,\lambda}\right] \tag{42}
\end{align}$$

substituting for the expectation (Eq. 38), and identifying the activity kernel (Eq. 13),

$$= \tfrac{1}{N_{\ell-1}} \sum_\mu h^{\ell-1}_{i,\mu} h^{\ell-1}_{j,\mu} = L^{\ell-1}_{ij} \tag{43}$$

Thus,

$$\mathbb{C}\left[J^\ell_{ij}, J^\ell_{kl} | \mathbf{K}^0\right] = \mathbb{C}\left[L^{\ell-1}_{ij}, L^{\ell-1}_{kl} | \mathbf{K}^0\right] \tag{44}$$

Combining this expression with Eq. (36) gives a complete form for the updates (Eq. 28),

$$\begin{align}
\mathbb{C}\left[J^\ell_{ij}, J^\ell_{kl} | \mathbf{L}^0\right] &= \mathbb{C}\left[L^{\ell-1}_{ij}, L^{\ell-1}_{kl} | \mathbf{K}^0\right] \tag{45a}\\
\mathbb{C}\left[K^\ell_{is,jr}, K^\ell_{kl} | \mathbf{L}^0\right] &= \mathbb{C}\left[J^\ell_{ij}, J^\ell_{kl} | \mathbf{L}^0\right] + \tfrac{1}{N_\ell}\left(\left\langle J^\ell_{ik}\right\rangle\left\langle J^\ell_{jl}\right\rangle + \left\langle J^\ell_{il}\right\rangle\left\langle J^\ell_{jk}\right\rangle\right) \nonumber\\
&\quad + \tfrac{1}{N_\ell}\left(\mathbb{C}\left[J^\ell_{ik}, J^\ell_{jl} | \mathbf{L}_0\right] + \mathbb{C}\left[J^\ell_{il}, J^\ell_{jk} | \mathbf{L}_0\right]\right) \tag{45b}\\
\mathbb{C}\left[L^\ell_{ij}, L^\ell_{kl} | \mathbf{L}^0\right] &= \mathbb{C}\left[K^\ell_{ij}, K^\ell_{kl} | \mathbf{L}^0\right] \tag{45c}
\end{align}$$

However, this form is difficult to analyse due to the complexity of Eq. (45b). Instead we can form an approximation to Eq. (45b) by noting that one of the recursive terms is negligable. Taking all network widths to be equal, $N_\ell = N$, (or at least of the same order), if

$$\mathbb{C}\left[L^{\ell-1}_{ij}, L^{\ell-1}_{kl} | \mathbf{L}^0\right] = \mathcal{O}(1/N), \tag{46}$$

then

$$\mathbb{C}\left[J^\ell_{ij}, J^\ell_{kl} | \mathbf{L}^0\right] = \mathcal{O}(1/N), \tag{47}$$

as the network is chosen such that the activities, and hence the covariances, $J^\ell_{ij}$ remain $\mathcal{O}(1)$

$$\mathbb{C}\left[K^\ell_{is,jr}, K^\ell_{kl} | \mathbf{L}^0\right] = \mathbb{C}\left[J^\ell_{ij}, J^\ell_{kl} | \mathbf{L}^0\right] + \tfrac{1}{N_\ell}\left(\left\langle J^\ell_{ik}\right\rangle\left\langle J^\ell_{jl}\right\rangle + \left\langle J^\ell_{il}\right\rangle\left\langle J^\ell_{jk}\right\rangle\right) + \mathcal{O}(1/N^2) = \mathcal{O}(1/N), \tag{48}$$

so,

$$\mathbb{C}\left[L^\ell_{ij}, L^\ell_{kl} | \mathbf{L}^0\right] = \mathcal{O}(1/N). \tag{49}$$

To begin the recursion, the data is fixed, so

$$\mathbb{C}\left[J^1_{ij}, J^1_{kl}|\mathbf{L}^0\right] = \mathbb{C}\left[L^0_{ij}, L^0_{kl}|\mathbf{L}^0\right] = 0, \tag{50}$$

and,

$$\mathbb{C}\left[K^1_{ij}, K^1_{kl}|\mathbf{L}^0\right] = \tfrac{1}{N}\left(\left\langle L^0_{ik}\right\rangle\left\langle L^0_{jl}\right\rangle + \left\langle J^0_{il}\right\rangle\left\langle J^0_{jk}\right\rangle\right) = \mathcal{O}(1/N). \tag{51}$$

Thus, the covariance of the kernels and covariances is in indeed $\mathcal{O}(1/N)$, so $\mathbb{C}\, K^\ell_{ij}, K^\ell_{kl}$ can be approximated by Eq. (48). Combining this approximation with Eq. (45) gives the expressions in the main text (Eq. 19). Finally, note that the approximation in Eq. (48) remains true only as long as the number of layers is small, $L \ll N$.

## 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 $\ell$, spatial location $r$ and channel $\lambda$ are given by,

$$a^\ell_{i,r\lambda} = \sum_{r'\mu} h^{\ell-1}_{i,r'\mu} W^\ell_{r'\mu,r\lambda}. \tag{52}$$

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, $\mathbf{h}^{\ell-1} \in \mathbb{R}^{P\times SM_{\ell-1}}$, and the weights, $\mathbf{W}_\ell \in \mathbb{R} \in^{SM_{\ell-1}\times SN_\ell}$, 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,

$$E\left[W^\ell_{r'\mu,r\lambda} W^\ell_{s'\nu,s\lambda}\right] = \tfrac{1}{M_{\ell-1}D_{\ell-1}} \delta_{\mu,\nu} \sum_{d\in\mathcal{D}_{\ell-1}} \delta_{r',(r+d)}\delta_{s',(s+d)}. \tag{53}$$

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,

$$E\left[W^\ell_{r'\mu,r\lambda} W^\ell_{s'\nu,s\lambda}\right] = \tfrac{1}{M_{\ell-1}D_{\ell-1}} \delta_{\mu,\nu}\delta_{r,s} \sum_{d\in\mathcal{D}_{\ell-1}} \delta_{r',(r+d)}\delta_{s',(s+d)}. \tag{54}$$

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

$$J^\ell_{ir,js} = \mathbb{E}\left[a^\ell_{i,r\lambda} a^\ell_{j,s\lambda}|\mathbf{L}_{\ell-1}\right] \tag{55}$$

$$= \mathbb{E}\left[\left(\sum_{r'\mu} h^{\ell-1}_{i,r'\mu} W^\ell_{r'\mu,r\lambda}\right)\left(\sum_{s'\nu} h^{\ell-1}_{j,s'\nu} W^\ell_{s'\nu,r\lambda}\right)\middle|\mathbf{L}_{\ell-1}\right] \tag{56}$$

$$= \sum_{\mu\nu r's'} h^{\ell-1}_{i,r'\mu} h^{\ell-1}_{j,s'\nu}\, \mathbb{E}\left[W^\ell_{r'\mu,r\lambda} W^\ell_{s'\nu,r\lambda}\right] \tag{57}$$

substituting the covariance of the weights (Eq. 53), and noting that the product of $h$’s forms the definition of the activity kernel (Eq. 13),

$$J^\ell_{ir,js} = \tfrac{1}{D_{\ell-1}} \sum_{d\in\mathcal{D}_{\ell-1}} L^{\ell-1}_{i(r+d),j(s+d)} \tag{58}$$

For locally connected intermediate layers, we instead substitute Eq. (54), which gives the same result, except that the output locations must be the same for there to be any covariance in the weights,

$$J^\ell_{ir,js} = \tfrac{1}{D_{\ell-1}} \delta_{r,s} \sum_{d\in\mathcal{D}_{\ell-1}} L^{\ell-1}_{i(r+d),j(s+d)} \tag{59}$$

Substituting this into Eq. (28a),

$$\mathbb{C}\left[J^{\ell}_{ir,js}, J^{\ell}_{ku,lv}|\mathbf{L}^0\right] = \mathbb{C}\left[\tfrac{1}{D_{\ell-1}}\sum_{d\in\mathcal{D}_{\ell-1}} L^{\ell-1}_{i(r+d),j(s+d)}, \tfrac{1}{D_{\ell-1}}\sum_{d\in\mathcal{D}_{\ell-1}} L^{\ell-1}_{k(u+d),l(v+d)}\right]. \tag{60}$$

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. (28), as indexing both a datapoint and a spatial location (i.e. $i \to i, s$),

$$\mathbb{C}\left[J^{\ell}_{ir,js}, J^{\ell}_{ku,lv}|\mathbf{L}^0\right] = \tfrac{1}{D^2_{\ell-1}}\sum_{dd'}\mathbb{C}\left[L^{\ell-1}_{i(r+d),j(s+d)}, L^{\ell-1}_{k(u+d'),l(v+d')}|\mathbf{L}^0\right] \tag{61a}$$

$$\mathbb{C}\left[K^{\ell}_{ir,js}, K^{\ell}_{ku,lv}|\mathbf{L}^0\right] \approx \mathbb{C}\left[J^{\ell}_{ir,js}, J^{\ell}_{ku,lv}|\mathbf{L}^0\right] + \tfrac{1}{N_\ell}\left(\left\langle J^{\ell}_{ir,ku}\right\rangle\left\langle J^{\ell}_{js,lv}\right\rangle + \left\langle J^{\ell}_{ir,lv}\right\rangle\left\langle J^{\ell}_{js,ku}\right\rangle\right) \tag{61b}$$

$$\mathbb{C}\left[L^{\ell}_{ir,js}, L^{\ell}_{ku,lv}|\mathbf{L}^0\right] = \mathbb{C}\left[K^{\ell}_{ir,js}, K^{\ell}_{ku,lv}|\mathbf{L}^0\right] \tag{61c}$$

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

$$\mathbb{C}\left[J^{\ell}_{ir,jr}, J^{\ell}_{ku,lu}|\mathbf{L}^0\right] = \tfrac{1}{D^2_{\ell-1}}\sum_{dd'}\mathbb{C}\left[L^{\ell-1}_{i(r+d),j(r+d)}, L^{\ell-1}_{k(u+d'),l(u+d')}|\mathbf{L}^0\right], \tag{62}$$

which reduces computational complexity, and the resulting expression can even be evaluated efficiently as a 2D convolution.

### A.2.1. Convolutional and locally connected networks

To understand the very different results for convolutional and locally structured networks (Fig. 3B–D) despite their having the same infinite limit, we need to consider how Eq. (62) interacts with Eq. (59). For a locally connected network, the covariance of activations at different locations is always zero, i.e. $J^{\ell}_{ir,js} = 0$ for $r \neq s$ whereas, for a spatially structured network, the $J_{ir,js}$ terms for $r \neq s$ have the same scale as those for $r = s$. The $J_{ir,js}$ terms enter into the variance of the kernel through Eq. (62). 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 convolutional networks, there are $D^2_{\ell-1}$ terms all with the same scale, whereas in spatially unstructured networks, we have only $D_{\ell-1}$ nonzero terms, introducing an effective $1/D_{\ell-1}$ normalizer. 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 $1/S = 1/D_L$ normalizer can be very large.

## 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, where $P$ is the number of datapoints. In particular, we can write the activities at the next layer using,

$$\mathbf{A}_\ell = \mathbf{H}_{\ell-1}\mathbf{W}_\ell = \mathbf{U}^T_\ell\mathbf{V}_\ell. \tag{63}$$

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

$$\mathbf{J}_\ell = \mathbf{U}^T_\ell\mathbf{U}_\ell. \tag{64}$$

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

$$\mathbf{K}_\ell = \tfrac{1}{N_\ell}\mathbf{A}_\ell\mathbf{A}^T_\ell = \tfrac{1}{N_\ell}\mathbf{H}_{\ell-1}\mathbf{W}_\ell\mathbf{W}^T_\ell\mathbf{H}^T_{\ell-1} = \tfrac{1}{N_\ell}\mathbf{U}^T_\ell\mathbf{V}_\ell\mathbf{V}^T_\ell\mathbf{U}_\ell. \tag{65}$$

Rearranging, we can write $\tfrac{1}{N_\ell}\mathbf{V}_\ell\mathbf{V}^T_\ell$, 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,

$$\mathbf{R}_\ell = \mathbf{U}^{-T}_\ell\mathbf{K}_\ell\mathbf{U}^{-1}_\ell = \tfrac{1}{N_\ell}\mathbf{V}_\ell\mathbf{V}^T_\ell \tag{66}$$

where $\mathbf{X}^{-T} = (\mathbf{X}^{-1})^T = (\mathbf{X}^T)^{-1}$, and we have assumed that $\mathbf{J}_\ell$ is invertible, which if nothing else, requires that the number of features $M_\ell$ and $N_\ell$ are larger than the number of datapoints.

### 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,

$$\begin{aligned}
\log \mathrm{P}\left(\mathbf{V}_\ell\right) &= -\tfrac{1}{2} \operatorname{Tr}\left(\mathbf{V}_\ell \mathbf{V}_\ell^T\right) + \text{const} && (67)\\
&= -\tfrac{N_\ell}{2} \operatorname{Tr}\left(\mathbf{U}_\ell^{-T} \mathbf{K}_\ell \mathbf{U}_\ell^{-1}\right) + \text{const} && (68)\\
&= -\tfrac{N_\ell}{2} \operatorname{Tr}\left(\mathbf{K}_\ell \mathbf{U}_\ell^{-1} \mathbf{U}_\ell^{-T}\right) + \text{const} && (69)\\
&= -\tfrac{N_\ell}{2} \operatorname{Tr}\left(\mathbf{K}_\ell \left(\mathbf{U}_\ell^T \mathbf{U}_\ell\right)^{-1}\right) + \text{const} && (70)\\
&= -\tfrac{N_\ell}{2} \operatorname{Tr}\left(\mathbf{K}_\ell \mathbf{J}_\ell^{-1}\right) + \text{const} && (71)
\end{aligned}$$

We can write the likelihood in the same form,

$$\begin{aligned}
\log \mathrm{P}\left(\mathbf{Y}|\mathbf{J}_{L+1}\right) &= -\tfrac{1}{2} \operatorname{Tr}\left(\mathbf{Y}^T \mathbf{J}_{L+1}^{-1} \mathbf{Y}\right) + \text{const} && (72)\\
\log \mathrm{P}\left(\mathbf{Y}|\mathbf{J}_{L+1}\right) &= -\tfrac{1}{2} \operatorname{Tr}\left(\mathbf{Y}\mathbf{Y}^T \mathbf{J}_{L+1}^{-1}\right) + \text{const} && (73)\\
\log \mathrm{P}\left(\mathbf{Y}|\mathbf{J}_{L+1}\right) &= -\tfrac{N_{L+1}}{2} \operatorname{Tr}\left(\mathbf{K}_{L+1} \mathbf{J}_{L+1}^{-1}\right) + \text{const} && (74)
\end{aligned}$$

where,

$$\mathbf{K}_{L+1} = \tfrac{1}{N_{L+1}} \mathbf{Y}\mathbf{Y}^T. \tag{75}$$

Note that we would usually incorporate IID noise in the outputs, and we are not doing so here in order to give exact, interpretable solutions. We do not expect this to change the overall pattern of the results, except to marginally weaken the connection between the the output-kernel, $\mathbf{K}_{L+1}$, and top-layer kernel, $\mathbf{K}_L$.

Thus, the joint probability can be written as,

$$\log \mathrm{P}\left(\mathbf{V}_1, \dots, \mathbf{V}_L, \mathbf{Y}|\mathbf{X}\right) = -\tfrac{1}{2} \sum_{\ell=1}^{L+1} N_\ell \operatorname{Tr}\left(\mathbf{K}_\ell \mathbf{J}_\ell^{-1}\right) + \text{const}\,. \tag{76}$$

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

$$\mathbf{V}_1^*, \dots, \mathbf{V}_L^* = \underset{\mathbf{V}_1, \dots, \mathbf{V}_L}{\arg\max} \log \mathrm{P}\left(\mathbf{V}_1, \dots, \mathbf{V}_L, \mathbf{Y}|\mathbf{X}\right), \tag{77}$$

by taking gradients of $\mathrm{P}\left(\mathbf{V}_1, \dots, \mathbf{V}_L, \mathbf{Y}|\mathbf{X}\right)$ wrt $\mathbf{K}_1, \dots, \mathbf{K}_L$. Note that we can find the mode of this distribution by differeniating with respect to many different quantities, and we choose $\mathbf{K}_\ell$ because of algebraic convenience, and because it includes all relevant information from $\mathbf{V}_\ell$ (Eq. 65). Further, note that as we are still working with the probability density of $\mathbf{V}_1, \dots, \mathbf{V}_L$ we should not include a Jacobian term. Now we consider a linear, fully connected network where $\mathbf{J}_\ell = \mathbf{K}_{\ell-1}$,

$$\mathbf{0} = \frac{\partial}{\partial \mathbf{K}_\ell} \log \mathrm{P}\left(\mathbf{V}_1, \dots, \mathbf{V}_L, \mathbf{Y}|\mathbf{X}\right) = -\tfrac{N_\ell}{2} \mathbf{K}_{\ell-1}^{-1} + \tfrac{N_{\ell+1}}{2} \mathbf{K}_\ell^{-1} \mathbf{K}_{\ell+1} \mathbf{K}_\ell^{-1} \tag{78}$$

where we have used,

$$\begin{aligned}
\frac{\partial \operatorname{Tr}\left(\mathbf{K}_\ell^{-1} \mathbf{K}_{\ell+1}\right)}{\partial \mathbf{K}_\ell} &= -\mathbf{K}_\ell^{-1} \mathbf{K}_{\ell+1} \mathbf{K}_\ell^{-1} && (79)\\
\frac{\partial \operatorname{Tr}\left(\mathbf{K}_{\ell-1}^{-1} \mathbf{K}_\ell\right)}{\partial \mathbf{K}_\ell} &= \mathbf{K}_{\ell-1}^{-1} && (80)
\end{aligned}$$

Thus, the MAP kernel changes as a fixed ratio,

$$\mathbf{S} = N_{\ell+1} \mathbf{K}_{\ell+1} \mathbf{K}_\ell^{-1} = N_\ell \mathbf{K}_\ell \mathbf{K}_{\ell-1}^{-1}, \tag{81}$$

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

$$\mathbf{S}^{L+1} = \prod_{\ell=1}^{L+1} N_\ell \mathbf{K}_\ell \mathbf{K}_{\ell-1}^{-1} = \mathbf{K}_{L+1} \mathbf{K}_0^{-1} \prod_{\ell=1}^{L+1} N_\ell \tag{82}$$

so,

$$\mathbf{S} = \left(\mathbf{K}_{L+1}\mathbf{K}_0^{-1}\right)^{1/L+1} \left(\prod_{\ell=1}^{L+1} N_\ell\right)^{1/L+1} \tag{83}$$

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

$$\begin{aligned}\mathbf{K}_\ell &= \left(\prod_{\ell'=1}^{\ell} \mathbf{K}_\ell \mathbf{K}_{\ell-1}^{-1}\right)\mathbf{K}_0 &(84)\\ &= \left(\prod_{\ell'=1}^{\ell} \tfrac{1}{N_{\ell'}}\mathbf{S}\right)\mathbf{K}_0 &(85)\\ &= \frac{\left(\prod_{\ell'=1}^{L+1} N_{\ell'}\right)^{\ell/(L+1)}}{\prod_{\ell'=1}^{\ell} N_{\ell'}} \left(\mathbf{K}_{L+1}\mathbf{K}_0^{-1}\right)^{\ell/(L+1)}\mathbf{K}_0 &(86)\end{aligned}$$

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

$$\begin{aligned}N_{\leq\ell} &= \left(\prod_{\ell'=1}^{\ell} N_{\ell'}\right)^{1/\ell} &(87)\\ N_{\ell<} &= \left(\prod_{\ell'=\ell+1}^{L+1} N_{\ell'}\right)^{1/(L+1-\ell)} &(88)\end{aligned}$$

we can write,

$$\begin{aligned}\frac{\left(\prod_{\ell'=1}^{L+1} N_\ell\right)^{\ell/(L+1)}}{\prod_{\ell'=1}^{\ell} N_\ell} &= \frac{\left(\left(N_{\leq\ell}\right)^{\ell}\left(N_{\ell<}\right)^{L+1-\ell}\right)^{\ell/(L+1)}}{\left(N_{\leq\ell}\right)^{\ell}} &(89)\\ &= \left(\left(N_{\leq\ell}\right)^{-(L+1-\ell)}\left(N_{\ell<}\right)^{L+1-\ell}\right)^{\ell/(L+1)} &(90)\\ &= \left(\frac{N_{\ell<}}{N_{\leq\ell}}\right)^{\frac{\ell(L+1-\ell)}{L+1}} &(91)\end{aligned}$$

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 power is 0),

$$\mathbf{K}_\ell = \left(\frac{N_{\ell<}}{N_{\leq\ell}}\right)^{\frac{\ell(L+1-\ell)}{L+1}} \left(\mathbf{K}_{L+1}\mathbf{K}_0^{-1}\right)^{\ell/(L+1)}\mathbf{K}_0 \tag{92}$$

Thus, MAP does something sensible: no matter 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 $\ell$ given by widening layers from 1 to $\ell$, 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 poor 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. 63)

$$d\mathbf{V}_\ell = \tfrac{1}{2} dt \frac{\partial \mathcal{L}}{\partial \mathbf{V}_\ell} + d\mathbf{\Xi}_\ell, \tag{93}$$

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 $\mathbf{R}_\ell$. In particular, we consider the expected change in $\mathbf{R}_\ell$ under Langevin sampling,

$$\mathbb{E}\left[d\mathbf{R}_\ell | \mathbf{R}_\ell\right] = \mathbb{E}\left[\tfrac{1}{N_\ell} d\left(\mathbf{V}_\ell\mathbf{V}_\ell^T\right)\right] = \tfrac{1}{2N_\ell} dt \left(\frac{\partial \mathcal{L}}{\partial \mathbf{V}_\ell}\mathbf{V}_\ell^T + \mathbf{V}_\ell \frac{\partial \mathcal{L}}{\partial \mathbf{V}_\ell}^T\right) + \tfrac{1}{N_\ell}\mathbb{E}\left[d\mathbf{\Xi}_\ell d\mathbf{\Xi}_\ell^T\right]. \tag{94}$$

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

$$\tfrac{1}{N_\ell}\mathbb{E}\left[d\mathbf{\Xi}_\ell d\mathbf{\Xi}_\ell^T\right] = dt\, \mathbf{I}, \tag{95}$$

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

$$\lim_{N_\ell \to \infty} \frac{d\mathbf{R}_\ell}{dt} = \mathbb{E}\left[\frac{d\mathbf{R}_\ell}{dt}\middle| \mathbf{R}_\ell\right] = \tfrac{1}{2N_\ell} dt \left(\left(\frac{\partial \mathcal{L}}{\partial \mathbf{V}_\ell}\right)\mathbf{V}_\ell^T + \mathbf{V}_\ell \left(\frac{\partial \mathcal{L}}{\partial \mathbf{V}_\ell}\right)^T\right) + dt\mathbf{I}. \tag{96}$$

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 $\mathbf{V}_\ell$,

$$\mathcal{L} = -\tfrac{1}{2}\operatorname{Tr}\left(\mathbf{V}_\ell\mathbf{V}_\ell^T\right), \tag{97}$$

so the gradient is,

$$\frac{\partial \mathcal{L}}{\partial \mathbf{V}_\ell} = \frac{\partial}{\partial \mathbf{V}_\ell}\left[-\tfrac{1}{2}\operatorname{Tr}\left(\mathbf{V}_\ell\mathbf{V}_\ell^T\right)\right] = -\mathbf{V}_\ell. \tag{98}$$

Thus,

$$\mathbb{E}\left[\frac{d\mathbf{R}_\ell}{dt}\middle| \mathbf{R}_\ell\right] = \tfrac{1}{N_\ell}\mathbf{V}_\ell\mathbf{V}_\ell^T + \mathbf{I} = -\mathbf{R}_\ell + \mathbf{I}. \tag{99}$$

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

$$\mathbf{0} = \mathbb{E}\left[\frac{d\mathbf{R}_\ell}{dt}\right] = -\mathbb{E}\left[\mathbf{R}_\ell\right] + \mathbf{I}. \tag{100}$$

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

$$\mathbb{E}\left[\mathbf{R}_\ell\right] = \mathbb{E}\left[\tfrac{1}{N_\ell}\mathbf{V}_\ell\mathbf{V}_\ell^T\right] = \mathbf{I}, \tag{101}$$

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 write the expected dynamics of $\mathbf{R}_\ell$ under Langevin sampling as the gradient of a surrogate objective, $\mathcal{L}'$,

$$\mathcal{L}' = \mathcal{L} + \tfrac{N_\ell}{2}\log|\mathbf{R}| = \mathcal{L} + \tfrac{N_\ell}{2}\log\left|\mathbf{V}_\ell\mathbf{V}_\ell^T\right|. \tag{102}$$

The gradient of the determinant is given by the pseudo-inverse,

$$\frac{\partial}{\partial \mathbf{V}_\ell}\log\left|\tfrac{1}{N_\ell}\mathbf{V}_\ell\mathbf{V}_\ell^T\right| = 2\left(\mathbf{V}_\ell\mathbf{V}_\ell^T\right)^{-1}\mathbf{V}_\ell. \tag{103}$$

Thus, continuous gradient descent on the full objective, with a learning rate of $\frac{1}{2}$, gives,

$$d\mathbf{V}_\ell = \tfrac{1}{2}dt \left[\frac{\partial \mathcal{L}'}{\partial \mathbf{V}_\ell}\right] = \tfrac{1}{2}dt \left[\frac{\partial \mathcal{L}}{\partial \mathbf{V}_\ell} + N_\ell \left(\mathbf{V}_\ell \mathbf{V}_\ell^T\right)^{-1} \mathbf{V}_\ell\right] \tag{104}$$

The implied change in $\mathbf{R}$ is,

$$d\mathbf{R}_\ell = \tfrac{1}{N_\ell}\left(d\mathbf{V}_\ell \mathbf{V}_\ell^T + \mathbf{V}_\ell d\mathbf{V}_\ell^T\right) = \tfrac{dt}{2N_\ell}\left(\left(\frac{\partial \mathcal{L}}{\partial \mathbf{V}_\ell}\right)\mathbf{V}_\ell^T + \mathbf{V}_\ell\left(\frac{\partial \mathcal{L}}{\partial \mathbf{V}_\ell}\right)^T\right) + dt\mathbf{I} \tag{105}$$

And this is exactly equal to the change in $\mathbf{R}$ induced by Langevin sampling Eq. (96).

## 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,

$$\begin{aligned}\log \mathrm{P}\left(\mathbf{K}_\ell|\mathbf{J}_\ell\right) &= \log \mathrm{Wishart}\left(\mathbf{K}_\ell; \tfrac{1}{N_\ell}\mathbf{J}_\ell, N_\ell\right) &(106)\\ &= \tfrac{N_\ell - P - 1}{2}\log|\mathbf{K}_\ell| - \tfrac{N_\ell}{2}\log|\mathbf{J}_\ell| - \tfrac{N_\ell}{2}\operatorname{Tr}\left(\mathbf{J}_{\ell-1}^{-1}\mathbf{K}_\ell\right). &(107)\end{aligned}$$

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

$$\begin{aligned}\mathbb{E}\left[\mathbf{K}_\ell|\mathbf{J}_\ell\right] &= \mathbf{J}_\ell &(108a)\\ \operatorname*{arg\,max}_{\mathbf{K}_\ell}\left[\log \mathrm{P}\left(\mathbf{K}_\ell|\mathbf{J}_\ell\right)\right] &= \left(N_\ell - P - 1\right)\mathbf{J}_\ell, &(108b)\end{aligned}$$

where the matricies $\mathbf{K}_\ell$ and $\mathbf{J}_\ell$ are $P \times P$, and $\mathbf{K}_\ell$ is the inner product of $N_\ell$ 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,

$$\operatorname*{arg\,max}_{\mathbf{K}_\ell}\left[\log \mathrm{P}\left(\mathbf{K}_\ell|\mathbf{J}_\ell\right) + \tfrac{P+1}{2}\log|\mathbf{K}_\ell|\right] = \mathbf{J}_\ell. \tag{109}$$

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, $\mathcal{L}'$,

$$\begin{aligned}\log \mathrm{P}\left(\mathbf{K}_\ell|\mathbf{J}_\ell\right) + \tfrac{P+1}{2}\log|\mathbf{K}_\ell| &= \tfrac{N_\ell}{2}\log\left|\mathbf{J}_\ell^{-1}\mathbf{K}_\ell\right| - \tfrac{N_\ell}{2}\operatorname{Tr}\left(\mathbf{J}_\ell^{-1}\mathbf{K}_\ell\right) &(110)\\ &= \tfrac{N_\ell}{2}\log\left|\mathbf{U}_\ell^{-T}\mathbf{K}_\ell\mathbf{U}_\ell^{-1}\right| - \tfrac{N_\ell}{2}\operatorname{Tr}\left(\mathbf{U}_\ell^{-T}\mathbf{K}_\ell\mathbf{U}_\ell^{-1}\right) &(111)\\ &= \tfrac{N_\ell}{2}\log|\mathbf{R}_\ell| - \tfrac{N_\ell}{2}\operatorname{Tr}\left(\mathbf{R}_\ell\right). &(112)\end{aligned}$$

if we consider a simple one-layer setup, with $\mathcal{L}$ given by Eq. (97).

## 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. 102), and the modified objective for Wishart inference (Eq. 110). In particular,

$$\log \mathrm{P}\left(\mathbf{y}_\mu|\mathbf{K}_L\right) = -\tfrac{1}{2}\mathbf{y}_\mu^T\mathbf{L}_L^{-1}\mathbf{y}_\mu - \tfrac{1}{2}\left|\mathbf{K}_L\right| + \text{const}. \tag{113}$$

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,

$$\log \mathrm{P}\left(\mathbf{Y}|\mathbf{K}_L\right) = \tfrac{Y}{2}\left(\log\left|\mathbf{L}_L^{-1}\mathbf{K}_{L+1}\right| - \operatorname{Tr}\left(\mathbf{L}_L^{-1}\mathbf{K}_{L+1}\right)\right) + \text{const} \tag{114}$$

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,

$$\mathcal{L} = \sum_{\ell=1}^{L+1} \tfrac{N_\ell}{2} \left(\log\left|\mathbf{L}_{\ell-1}^{-1}\mathbf{K}_\ell\right| - \operatorname{Tr}\left(\mathbf{L}_{\ell-1}^{-1}\mathbf{K}_\ell\right)\right). \tag{115}$$

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

$$\mathcal{L} = \tfrac{N_\ell}{2} \left(\log\left|\mathbf{L}_{\ell-1}^{-1}\mathbf{K}_\ell\right| - \operatorname{Tr}\left(\mathbf{L}_{\ell-1}^{-1}\mathbf{K}_\ell\right)\right) + \tfrac{N_{\ell+1}}{2} \left(\log\left|\mathbf{L}_{\ell}^{-1}\mathbf{K}_{\ell+1}\right| - \operatorname{Tr}\left(\mathbf{L}_{\ell}^{-1}\mathbf{K}_{\ell+1}\right)\right). \tag{116}$$

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,

$$\mathbf{L}_\ell = \mathbf{K}_\ell. \tag{117}$$

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

$$\mathcal{L} = \tfrac{N_\ell}{2} \left(\log\left|\mathbf{K}_{\ell-1}^{-1}\mathbf{K}_\ell\right| - \operatorname{Tr}\left(\mathbf{K}_{\ell-1}^{-1}\mathbf{K}_\ell\right)\right) + \tfrac{N_{\ell+1}}{2} \left(\log\left|\mathbf{K}_{\ell}^{-1}\mathbf{K}_{\ell+1}\right| - \operatorname{Tr}\left(\mathbf{K}_{\ell}^{-1}\mathbf{K}_{\ell+1}\right)\right). \tag{118}$$

Differentiating the relevant terms,

$$\frac{\partial \operatorname{Tr} \mathbf{K}_\ell^{-1}\mathbf{K}_{\ell+1}}{\partial \mathbf{K}_\ell} = -\mathbf{K}_\ell^{-1}\mathbf{K}_{\ell+1}\mathbf{K}_\ell^{-1} \tag{119a}$$

$$\frac{\partial \operatorname{Tr} \mathbf{K}_{\ell-1}^{-1}\mathbf{K}_{\ell}}{\partial \mathbf{K}_\ell} = \mathbf{K}_{\ell-1}^{-1} \tag{119b}$$

$$\frac{\partial \log\left|\mathbf{K}_{\ell-1}^{-1}\mathbf{K}_{\ell}\right|}{\partial \mathbf{K}_\ell} = \frac{\partial \log\left|\mathbf{K}_\ell\right|}{\partial \mathbf{K}_\ell} = \mathbf{K}_\ell^{-1} \tag{119c}$$

$$\frac{\partial \log\left|\mathbf{K}_{\ell}^{-1}\mathbf{K}_{\ell+1}\right|}{\partial \mathbf{K}_\ell} = -\frac{\partial \log\left|\mathbf{K}_\ell\right|}{\partial \mathbf{K}_\ell} = -\mathbf{K}_\ell^{-1} \tag{119d}$$

We then set the gradients to zero,

$$\mathbf{0} = \frac{\partial \mathcal{L}}{\partial \mathbf{K}_\ell} = -\left(N_{\ell+1} - N_\ell\right)\mathbf{K}_\ell^{-1} + N_{\ell+1}\,\mathbf{K}_\ell^{-1}\mathbf{K}_{\ell+1}\mathbf{K}_\ell^{-1} - N_\ell\,\mathbf{K}_{\ell-1}^{-1} \tag{120}$$

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

$$\mathbf{0} = -\left(N_{\ell+1} - N_\ell\right)\mathbf{I} + N_{\ell+1}\,\mathbf{K}_{\ell+1}\mathbf{K}_\ell^{-1} - N_\ell\,\mathbf{K}_\ell\mathbf{K}_{\ell-1}^{-1}, \tag{121}$$

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

$$\mathbf{0} = -\left(N_{\ell+1} - N_\ell\right)\mathbf{I} + N_{\ell+1}\,\mathbf{T}_{\ell+1} - N_\ell\,\mathbf{T}_\ell. \tag{122}$$

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

$$\mathbf{T}_{\ell+1} = \mathbf{I} + \tfrac{N_\ell}{N_{\ell+1}}\left(\mathbf{T}_\ell - \mathbf{I}\right) \tag{123}$$

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

$$\mathbf{T}_\ell = \begin{cases} \mathbf{T} & \text{for } \ell \in \{1, \dots, L\} \\ \mathbf{I} + \tfrac{N}{Y}\left(\mathbf{T} - \mathbf{I}\right) & \text{for } \ell = L+1 \end{cases} \tag{124}$$

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

$$\mathbf{K}_{L+1}\mathbf{K}_0^{-1} = \mathbf{T}_{L+1}\mathbf{T}^L, \tag{125}$$

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

$$\mathbf{K}_{L+1}\mathbf{K}_0^{-1} = \left(\mathbf{I} + \tfrac{N}{Y}\left(\mathbf{T} - \mathbf{I}\right)\right)\mathbf{T}^L \tag{126}$$

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

$$\lim_{N/Y\to 0} \mathbf{T} = \left(\mathbf{K}_{L+1}\mathbf{K}_0^{-1}\right)^{1/L} \tag{127}$$

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 $\mathbf{0} = \mathbf{T} - \mathbf{I}$ because otherwise the $\frac{N}{Y}\left(\mathbf{T} - \mathbf{I}\right)$ term will explode,

$$\lim_{N/Y\to\infty} \mathbf{T} = \mathbf{I}, \tag{128}$$

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 $Y = N$,

$$\mathbf{T} = \left(\mathbf{K}_{L+1}\mathbf{K}_0^{-1}\right)^{1/(L+1)} \tag{129}$$

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,

$$\mathbf{K}_L = \mathbf{T}^L\mathbf{K}_0 = \left(\mathbf{K}_{L+1}\mathbf{K}_0^{-1}\right)^{L/(L+1)}\mathbf{K}_0 \tag{130}$$

## 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 $\lambda_i$,

$$\mathbf{K} = \sum_i \lambda_i \mathbf{K}_i. \tag{131}$$

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

$$\log \mathrm{P}\left(\mathbf{Y}\right) = -\tfrac{1}{2}\operatorname{Tr}\left(\mathbf{K}^{-1}\mathbf{Y}\mathbf{Y}^T\right) - \tfrac{N}{2}\log\left|\mathbf{K}\right| + \text{const}\,. \tag{132}$$

And the gradient is,

$$\frac{\partial \log \mathrm{P}\left(\mathbf{Y}\right)}{\partial \lambda_\alpha} = \tfrac{1}{2}\operatorname{Tr}\mathbf{L}_\alpha\mathbf{L}_\mathrm{y} - \tfrac{N}{2}\operatorname{Tr}\mathbf{L}_\alpha. \tag{133}$$

where,

$$\mathbf{L}_\alpha = \mathbf{K}^{-1}\mathbf{K}_\alpha \tag{134}$$

$$\mathbf{L}_\mathrm{y} = \mathbf{K}^{-1}\mathbf{Y}\mathbf{Y}^T \tag{135}$$

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

$$\begin{aligned}
\mathbb{E}\left[\frac{\partial}{\partial\lambda_\beta}\left[\tfrac{1}{2}\operatorname{Tr}\mathbf{L}_\alpha\mathbf{L}_\mathrm{y}\right]\right] &= \mathbb{E}\left[-\tfrac{1}{2}\left(\operatorname{Tr}\mathbf{L}_\beta\mathbf{L}_\alpha\mathbf{L}_\mathrm{y} + \operatorname{Tr}\mathbf{L}_\alpha\mathbf{L}_\beta\mathbf{L}_\mathrm{y}\right)\right] && (136)\\
&= -\tfrac{1}{2}\left(\operatorname{Tr}\mathbf{L}_\beta\mathbf{L}_\alpha\,\mathbb{E}\left[\mathbf{L}_\mathrm{y}\right] + \operatorname{Tr}\mathbf{L}_\alpha\mathbf{L}_\beta\,\mathbb{E}\left[\mathbf{L}_\mathrm{y}\right]\right) && (137)\\
&= -\tfrac{N}{2}\left(\operatorname{Tr}\mathbf{L}_\beta\mathbf{L}_\alpha + \operatorname{Tr}\mathbf{L}_\alpha\mathbf{L}_\beta\right) && (138)\\
&= -N\operatorname{Tr}\mathbf{L}_\alpha\mathbf{L}_\beta && (139)
\end{aligned}$$

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,

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

Thus,

$$\mathbb{E}\left[\frac{\partial^2}{\partial \lambda_\alpha \lambda_\beta}\log \mathrm{P}\left(\mathbf{Y}\right)\right] = -\tfrac{N}{2}\operatorname{Tr}\mathbf{L}_\alpha\mathbf{L}_\beta \tag{140}$$