A. Optimal approximate posterior derivation

Starting with Eq (3), we start by combining the prior and likelihood for P,

$$\mathcal{L} = \mathbb{E}_{\mathrm{Q}\left(\{\mathbf{W}_\ell\}_{\ell=1}^{L+1}\right)}\left[\log \mathrm{P}\left(\mathbf{Y}, \{\mathbf{W}\}_{\ell=1}^{L+1}|\mathbf{X}\right) - \log \mathrm{Q}\left(\{\mathbf{W}\}_{\ell=1}^{L+1}\right)\right], \tag{25}$$

then split out $\mathbf{W}_{L+1}$ from P and Q,

$$\mathcal{L} = \mathbb{E}_{\mathrm{Q}\left(\{\mathbf{W}\}_{\ell=1}^{L+1}\right)}\Big[\log \mathrm{P}\left(\mathbf{Y}, \{\mathbf{W}\}_{\ell=1}^{L}|\mathbf{X}\right) + \log \mathrm{P}\left(\mathbf{W}_{L+1}|\mathbf{Y}, \mathbf{X}, \{\mathbf{W}_\ell\}_{\ell=1}^{L}\right) \\ - \log \mathrm{Q}\left(\mathbf{W}_{L+1}|\{\mathbf{W}_\ell\}_{\ell=1}^{L}\right) - \log \mathrm{Q}\left(\{\mathbf{W}_\ell\}_{\ell=1}^{L}\right)\Big]. \tag{26}$$

We are interested in the optimal $\mathrm{Q}\left(\mathbf{W}_{L+1}|\{\mathbf{W}_\ell\}_{\ell=1}^{L}\right)$ for any setting of $\{\mathbf{W}_\ell\}_{\ell=1}^{L}$. Therefore, $\mathrm{P}\left(\mathbf{Y}, \{\mathbf{W}\}_{\ell=1}^{L}|\mathbf{X}\right)$ and $\mathrm{Q}\left(\{\mathbf{W}_\ell\}_{\ell=1}^{L}\right)$ do not depend on $\mathbf{W}_{L+1}$ and can be collected into $c(\{\mathbf{W}_\ell\}_{\ell=1}^{L})$. We therefore obtain,

$$\mathcal{L} = \mathbb{E}_{\mathrm{Q}\left(\{\mathbf{W}\}_{\ell=1}^{L+1}\right)}\left[\log \mathrm{P}\left(\mathbf{W}_{L+1}|\mathbf{Y}, \mathbf{X}, \{\mathbf{W}_\ell\}_{\ell=1}^{L}\right) - \log \mathrm{Q}\left(\mathbf{W}_{L+1}|\{\mathbf{W}_\ell\}_{\ell=1}^{L}\right) + c(\{\mathbf{W}_\ell\}_{\ell=1}^{L})\right]. \tag{27}$$

We can nest the expectations,

$$\mathcal{L} = \mathbb{E}_{\mathrm{Q}\left(\{\mathbf{W}\}_{\ell=1}^{L}\right)}\left[\mathbb{E}_{\mathrm{Q}\left(\mathbf{W}_{L+1}|\{\mathbf{W}_\ell\}_{\ell=1}^{L}\right)}\left[\log \mathrm{P}\left(\mathbf{W}_{L+1}|\mathbf{Y}, \mathbf{X}, \{\mathbf{W}_\ell\}_{\ell=1}^{L}\right) - \log \mathrm{Q}\left(\mathbf{W}_{L+1}|\{\mathbf{W}_\ell\}_{\ell=1}^{L}\right)\right] + c(\{\mathbf{W}\}_{\ell=1}^{L})\right]. \tag{28}$$

Then notice that the inner expectation is a KL-divergence,

$$\mathcal{L} = \mathbb{E}_{\mathrm{Q}\left(\{\mathbf{W}\}_{\ell=1}^{L}\right)}\left[-\mathrm{D}_{\mathrm{KL}}\left(\mathrm{Q}\left(\mathbf{W}_{L+1}|\{\mathbf{W}_\ell\}_{\ell=1}^{L}\right)||\,\mathrm{P}\left(\mathbf{W}_{L+1}|\mathbf{Y}, \mathbf{X}, \{\mathbf{W}_\ell\}_{\ell=1}^{L}\right)\right) + c(\{\mathbf{W}\}_{\ell=1}^{L})\right]. \tag{29}$$

B. Reparameterised variational inference

In variational inference, the ELBO objective takes the form,

$$\mathcal{L}(\phi) = \mathbb{E}_{\mathrm{Q}_\phi(\mathbf{w})}\left[\log \mathrm{P}\left(\mathbf{Y}|\mathbf{X}, \mathbf{w}\right) + \log \frac{\mathrm{P}\left(\mathbf{w}\right)}{\mathrm{Q}_\phi\left(\mathbf{w}\right)}\right] \tag{30}$$

where $\mathbf{w}$ is a vector containing all of the elements of the weight matrices in the full network, $\{\mathbf{W}_\ell\}_{\ell=1}^{L+1}$, and $\phi = (\mathbf{U}_0, \{\mathbf{V}_\ell, \mathbf{\Lambda}_\ell\}_{\ell=1}^{L+1})$ are the parameters of the approximate posterior. This objective is difficult to differentiate wrt ϕ, because ϕ parameterises the distribution over which the expectation is taken. Following Kingma & Welling (2013) and Rezende et al. (2014), we sample ϵ from a simple, fixed distribution (e.g. IID Gaussian), and transform them to give samples from $\mathrm{Q}\left(w\right)$,

$$\mathbf{w}(\epsilon; \phi) \sim \mathrm{Q}_\phi\left(\mathbf{w}\right). \tag{31}$$

Thus, the ELBO can be written,

$$\mathcal{L}(\phi) = \mathbb{E}_\epsilon\left[\log \mathrm{P}\left(\mathbf{Y}|\mathbf{X}, \mathbf{w}(\epsilon; \phi)\right) + \log \frac{\mathrm{P}\left(\mathbf{w}(\epsilon; \phi)\right)}{\mathrm{Q}_\phi\left(\mathbf{w}(\epsilon; \phi)\right)}\right] \tag{32}$$

As the distribution over which the expectation is taken is now independent of ϕ, we can form unbiased estimates of the gradient of $\mathcal{L}(\phi)$ by drawing one or a few samples of ϵ.

C. Efficient convolutional linear regression

Working in one dimension for simplicity, the standard form for a convolution in deep learning is

$$Y_{iu,c} = \sum_{c'\delta} X_{i(u+\delta)c'} W_{\delta c',c}. \tag{33}$$

where X is the input image/feature-map, Y is the output feature-map, W is the convolutional weights, i indexes images, c and c' index channels, u indexes the location within the image, and δ indexes the location within the convolutional patch.

Later, we will swap the identity of the "patch location" and the "image location" and to facilitate this, we define them both to be centred on zero,

$$u \in \{-(W-1)/2, \ldots, (W-1)/2\} \qquad \delta \in \{-(S-1)/2, \ldots, (S-1)/2\} \tag{34}$$

where W is the size of an image and S is the size of a patch, such that, for example for a size 3 kernel, $\delta \in \{-1, 0, 1\}$.

To use the expressions for the fully-connected case, we can form a new input, X' by cutting out each image patch,

$$X'_{iu,\delta c} = X_{i(u+\delta)c}, \tag{35}$$

leading to

$$Y_{iu,c} = \sum_{\delta c'} X'_{iu,\delta c'} W_{\delta c',c}. \tag{36}$$

Note that we have used commas to group pairs of indices (iu and $\delta c'$) that may be combined into a single index (e.g. using a reshape operation). Indeed, combining i and u into a single index and combining δ and c' into a single index, this expression can be viewed as standard matrix multiplication,

$$\mathbf{Y} = \mathbf{X}'\mathbf{W}. \tag{37}$$

This means that we can directly apply the approximate posterior we derived for the fully-connected case in Eq. (11) to the convolutional case. To allow for this, we take

$$\mathbf{X}' = \mathbf{\Lambda}_\ell^{1/2} \phi\left(\mathbf{U}_{\ell-1}\right), \qquad \mathbf{Y} = \mathbf{\Lambda}_\ell^{1/2} \mathbf{V}_\ell. \tag{38}$$

While we can explicitly compute $\mathbf{X}'$ by extracting image patches, this imposes a very large memory cost (a factor of 9 for a 3×3 kernel, with stride 1, because there are roughly as many patches as pixels in the image, and a 3×3 patch requires 9 times the storage of a pixel. To implement convolutional linear regression with a more manageable memory cost, we instead compute the matrices required for linear regression directly as convolutions of the input feature-maps, $\mathbf{X}$ with themselves, and as convolutions of the X with the output feature maps, $\mathbf{Y}$, which we describe here.

For linear regression (Eq. 11), we first need to compute,

$$\left(\phi\left(\mathbf{U}_{\ell-1}\right)^T \mathbf{\Lambda}_\ell \mathbf{V}_\ell\right)_{\delta c,c'} = \left(\mathbf{X}'^T \mathbf{Y}\right)_{\delta c,c'} = \sum_{iu} X'_{iu,\delta c} Y_{iu,c'}. \tag{39}$$

Rewriting this in terms of X (i.e. without explicitly cutting out image patches), we obtain,

$$\left(\mathbf{X}'^T \mathbf{Y}\right)_{\delta c,c'} = \sum_{iu} X_{i(u+\delta)c} Y_{iu,c'}. \tag{40}$$

This can be directly viewed as the convolution of X and Y, where we treat Y as the "convolutional weights", u as the location within the now very large (size W) "convolutional patch", and δ as the location in the resulting output. Once we realise that the computation is a spatial convolution, it is possible to fit it into standard convolution functions provided by deep-learning frameworks (albeit with some rearrangement of the tensors).

Next, we need to compute,

$$\left(\phi\left(\mathbf{U}_{\ell-1}\right)^T \mathbf{\Lambda}_\ell \phi\left(\mathbf{U}_{\ell-1}\right)\right) = \left(\mathbf{X}'^T \mathbf{X}'\right)_{\delta c,\delta' c'} = \sum_{iu} X'_{iu,\delta c} X'_{iu,\delta' c'}. \tag{41}$$

Again, rewriting this in terms of X (i.e. without explicitly cutting out image patches), we obtain,

$$\left(\mathbf{X}'^T \mathbf{X}'\right)_{\delta c,\delta' c'} = \sum_{iu} X_{i(u+\delta)c} X_{i(u+\delta')c'}. \tag{42}$$

To treat this as a convolution, we first need exact translational invariance, which can be achieved by using circular boundary conditions. Note that circular boundary conditions are not typically used in neural networks for images, and we therefore

only use circular boundary conditions to define the approximate posterior over weights. The variational framework does not restrict us to also using circular boundary conditions within our feedforward network, and as such, we use standard zero-padding. With exact translational invariance, we can write this expression directly as a convolution,

$$\left(\mathbf{X}'^{T}\mathbf{X}'\right)_{\delta c,\delta' c'} = \sum_{iu} X_{iuc} X_{i(u+\delta'-\delta)c'} \tag{43}$$

where

$$(\delta' - \delta) \in \{-(S-1), \ldots, (S-1)\} \tag{44}$$

i.e. for a size 3 kernel, $(\delta' - \delta) \in \{-2, -1, 0, 1, 2\}$, where we treat X_{iuc} as the "convolutional weights", u as the location within the "convolutional patch", and $\delta' - \delta$ as the location in the resulting output "feature-map".

Finally, note that this form offers considerable benefits in terms of memory consumption. In particular, the output matrices are usually quite small — the number of channels is typically 32 or 64, and the number of locations within a patch is typically 9, giving a very manageable total size that is typically smaller than 1000×1000.

D. Wishart distributions with real-valued degrees of freedom

The classical description of the Wishart distribution,

$$\mathbf{\Sigma} \sim \text{Wishart}\left(\mathbf{I}, \nu\right), \tag{45}$$

where $\mathbf{\Sigma}$ is a $P \times P$ matrix, states that $P \geq \nu$ is an integer and we can generate $\mathbf{\Sigma}$ by taking the product of matrices, $\mathbf{X} \in \mathbb{R}^{\nu \times P}$, generated IID from a standard Gaussian,

$$\mathbf{\Sigma} = \mathbf{X}^T\mathbf{X}, \qquad X_{ij} \sim \mathcal{N}\left(0, 1\right). \tag{46}$$

However, for the purposes of defining learnable approximate posteriors, we need to be able to sample and evaluate the probability density when ν is positive real.

To do this, consider the alternative, much more efficient means of sampling from a Wishart distribution, using the Bartlett decomposition (Bartlett, 1933). The Bartlett decomposition gives the probability density for the Cholesky of a Wishart sample. In particular,

$$\mathbf{T} = \begin{pmatrix} T_{11} & \ldots & T_{1m} \\ \vdots & \ddots & \vdots \\ 0 & \ldots & T_{mm} \end{pmatrix}, \tag{47}$$

$$\mathrm{P}\left(T_{jj}^2\right) = \text{Gamma}\left(T_{jj}^2; \tfrac{\nu-j+1}{2}, \tfrac{1}{2}\right), \tag{48}$$

$$\mathrm{P}\left(T_{j<k}\right) = \mathcal{N}\left(T_{jk}; 0, 1\right). \tag{49}$$

Here, $\mathbf{T}_{jj}$ is usually considered to be sampled from a χ^2 distribution, but we have generalised this slightly using the equivalent Gamma distribution to allow for real-valued ν. Following Chafaï (2015), We need to change variables to T_{jj} rather than T_{jj}^2,

$$\mathrm{P}\left(T_{jj}\right) = \mathrm{P}\left(T_{jj}^2\right)\left|\frac{\partial T_{jj}^2}{\partial T_{jj}}\right|, \tag{50}$$

$$= \text{Gamma}\left(T_{jj}^2; \tfrac{\nu-j+1}{2}, \tfrac{1}{2}\right) 2T_{jj}, \tag{51}$$

$$= \frac{\left(T_{jj}^2\right)^{(\nu-j+1)/2-1} e^{-T_{jj}^2/2}}{2^{(\nu-j+1)/2}\Gamma\left(\frac{\nu-j+1}{2}\right)} 2T_{jj}, \tag{52}$$

$$= \frac{T_{jj}^{\nu-j} e^{-T_{jj}^2/2}}{2^{(\nu-j-1)/2}\Gamma\left(\frac{\nu-j+1}{2}\right)}. \tag{53}$$

Thus, the probability density for $\mathbf{T}$ under the Bartlett sampling operation is

$$\mathrm{P}\left(\mathbf{T}\right) = \underbrace{\prod_j \frac{T_{jj}^{\nu-j} e^{-T_{jj}^2/2}}{2^{\frac{\nu-j-1}{2}} \Gamma\left(\frac{\nu-j+1}{2}\right)}}_{\text{on-diagonals}} \underbrace{\prod_{k\in\{j+1,\ldots,m\}} \frac{1}{\sqrt{2\pi}} e^{-T_{jk}^2/2}}_{\text{off-diagonals}}. \tag{54}$$

To convert this to a distribution on $\mathbf{\Sigma}$, we need the volume element for the transformation from $\mathbf{T}$ to $\mathbf{\Sigma}$,

$$\mathrm{d}\mathbf{\Sigma} = 2^m \prod_{j=1}^m T_{jj}^{m-j+1} \mathrm{d}\mathbf{T}, \tag{55}$$

which can be obtained directly by computing the log-determinant of the Jacobian for the transformation from $\mathbf{T}$ to $\mathbf{\Sigma}$, or by taking the ratio of Eq. (54) and the usual Wishart probability density (with integral ν). Thus,

$$\mathrm{P}\left(\mathbf{\Sigma}\right) = \mathrm{P}\left(\mathbf{T}\right) \left(2^m \prod_{j=1}^m T_{jj}^{m-j+1}\right)^{-1} \tag{56}$$

$$= \prod_j \frac{T_{jj}^{\nu-m-1} e^{-T_{jj}^2/2}}{2^{\frac{\nu-j+1}{2}} \Gamma\left(\frac{\nu-j+1}{2}\right)} \cdot \prod_{k\in\{j+1,\ldots,m\}} \frac{1}{\sqrt{2\pi}} e^{-T_{jk}^2/2}. \tag{57}$$

Breaking this down into separate components and performing straightforward algebraic manipulations,

$$\prod_j T_{jj}^{\nu-m-1} = |\mathbf{T}|^{\nu-m-1} = |\mathbf{\Sigma}|^{(\nu-m-1)/2}, \tag{58}$$

$$\mathrm{P}\left(\mathbf{\Sigma}\right) = \prod_j e^{-T_{jj}^2/2} \prod_{k\in\{j+1,\ldots,m\}} e^{-T_{jk}^2/2} \tag{59}$$

$$= e^{-\sum_{jk} T_{jk}^2/2} = e^{-\operatorname{Tr}(\mathbf{\Sigma})/2}, \tag{60}$$

$$\prod_j \frac{1}{2^{(\nu-j-1)/2}} \prod_{k\in\{j+1,\ldots,m\}} \frac{1}{\sqrt{2}} = \left(\prod_j \frac{1}{2^{(\nu-j-1)/2}}\right) \left(\prod_j \frac{1}{2^{(m-j-1)/2}}\right) \tag{61}$$

$$= \left(\prod_j \frac{1}{2^{(\nu-j+1)/2}}\right) \left(\prod_j \frac{1}{2^{(j-1)/2}}\right) \tag{62}$$

$$= \prod_j \frac{1}{2^{(\nu-m)/2}} = 2^{-m\nu/2}, \tag{63}$$

where the second-to-last line was obtained by noting that $j-1$ covers the same range of integers as $m-j-1$ under the product. Finally, using the definition of the multivariate Gamma function,

$$\prod_j \Gamma\left(\tfrac{\nu-j+1}{2}\right) \prod_{k\in\{j+1,\ldots,m\}} \sqrt{\pi} = \pi^{m(m-1)/4} \prod_j \Gamma\left(\tfrac{\nu-j+1}{2}\right) = \Gamma_m\left(\tfrac{\nu}{2}\right). \tag{64}$$

We thereby re-obtain the probability density for the standard Wishart distribution,

$$\mathrm{P}\left(\mathbf{\Sigma}\right) = \frac{|\mathbf{\Sigma}|^{(\nu-m-1)/2} e^{-\operatorname{Tr}(\mathbf{\Sigma})/2}}{2^{m\nu/2} \Gamma_m\left(\frac{\nu}{2}\right)}. \tag{65}$$

E. Full description of our method for deep Gaussian process

Here we give the full derivation for our doubly-stochastic variational deep GPs, following Salimbeni & Deisenroth (2017). A deep Gaussian process (DGP; (Damianou & Lawrence, 2013; Salimbeni & Deisenroth, 2017)) defines a prior over function

values, $\mathbf{F}_\ell \in \mathbb{R}^{P\times N_\ell}$, where ℓ is the layer, P is the number of input points, and N_ℓ is the "width" of this layer, by stacking $L+1$ layers of standard Gaussian processes (we use $L+1$ layers instead of the typical L layers to retain consistency with how we define BNNs):

$$\mathrm{P}\left(\mathbf{Y}, \{\mathbf{F}_\ell\}_{\ell=1}^{L+1} | \mathbf{X}\right) = \mathrm{P}\left(\mathbf{Y}|\mathbf{F}_L\right) \prod_{\ell=1}^{L+1} \mathrm{P}\left(\mathbf{F}_\ell|\mathbf{F}_{\ell-1}\right). \tag{66}$$

Here, the input is $\mathbf{F}_0 = \mathbf{X}$, and the output is $\mathbf{Y}$ (which could be continuous values for regression, or class labels for classification), and the distribution over $\mathbf{F}_\ell \in \mathbb{R}^{P\times N_\ell}$ factorises into independent multivariate Gaussian distributions over each function,

$$\mathrm{P}\left(\mathbf{F}_\ell|\mathbf{F}_{\ell-1}\right) = \prod_{\lambda=1}^{N_\ell} \mathrm{P}\left(\mathbf{f}_\lambda^\ell|\mathbf{F}_{\ell-1}\right) = \prod_{\lambda=1}^{N_\ell} \mathcal{N}\left(\mathbf{f}_\lambda^\ell|\mathbf{0}, \mathbf{K}\left(\mathbf{F}_{\ell-1}\right)\right), \tag{67}$$

where $\mathbf{f}_\lambda^\ell$ is the λth column of $\mathbf{F}_\ell$, giving the activation of all datapoints for the λth feature, and $\mathbf{K}\left(\cdot\right)$ is a function that computes the kernel-matrix from the features in the previous layer.

To define a variational approximate posterior, we augment $\{\mathbf{F}_\ell\}_{\ell=1}^{L+1}$ with inducing points consisting of the function values $\{\mathbf{U}_\ell\}_{\ell=1}^{L+1}$,

$$\mathrm{P}\left(\mathbf{Y}, \{\mathbf{F}_\ell, \mathbf{U}_\ell\}_{\ell=1}^{L+1} | \mathbf{X}, \mathbf{U}_0\right) = \mathrm{P}\left(\mathbf{Y}|\mathbf{F}_{L+1}\right) \prod_{\ell=1}^{L+1} \mathrm{P}\left(\mathbf{F}_\ell, \mathbf{U}_\ell|\mathbf{F}_{\ell-1}, \mathbf{U}_{\ell-1}\right) \tag{68}$$

and because $\mathbf{F}_\ell$ and $\mathbf{U}_\ell$ are the function outputs corresponding to different inputs ($\mathbf{F}_{\ell-1}$ and $\mathbf{U}_{\ell-1}$), they form a joint multivariate Gaussian distribution, analogous to Eq. (67),

$$\mathrm{P}\left(\mathbf{F}_\ell, \mathbf{U}_\ell|\mathbf{F}_{\ell-1}, \mathbf{U}_{\ell-1}\right) = \prod_{\lambda=1}^{N_l} \mathcal{N}\left(\begin{pmatrix}\mathbf{f}_\lambda^\ell \\ \mathbf{u}_\lambda^\ell\end{pmatrix} \middle| \mathbf{0}, \mathbf{K}\left(\begin{pmatrix}\mathbf{F}_{\ell-1} \\ \mathbf{U}_{\ell-1}\end{pmatrix}\right)\right). \tag{69}$$

Following Salimbeni & Deisenroth (2017) we form an approximate posterior by conditioning the function values, $\mathbf{F}_\ell$ on the inducing outputs, $\mathbf{U}_\ell$,

$$\begin{aligned}\mathrm{Q}\left(\{\mathbf{F}_\ell, \mathbf{U}_\ell\}_{\ell=1}^{L+1} | \mathbf{X}, \mathbf{U}_0\right) &= \prod_{\ell=1}^{L+1} \mathrm{Q}\left(\mathbf{F}_\ell, \mathbf{U}_\ell|\mathbf{F}_{\ell-1}, \mathbf{U}_{\ell-1}\right) \\ &= \prod_{\ell=1}^{L+1} \mathrm{P}\left(\mathbf{F}_\ell|\mathbf{U}_\ell, \mathbf{U}_{\ell-1}, \mathbf{F}_{\ell-1}\right) \mathrm{Q}\left(\mathbf{U}_\ell|\mathbf{U}_{\ell-1}\right),\end{aligned} \tag{70}$$

where $\mathrm{Q}\left(\mathbf{U}_\ell|\mathbf{U}_{\ell-1}\right)$ is given by Eq. (22), i.e.

$$\mathrm{Q}\left(\mathbf{U}_\ell|\mathbf{U}_{\ell-1}\right) = \prod_{\lambda=1}^{N_\ell} \mathcal{N}\left(\mathbf{u}_\lambda^\ell \middle| \boldsymbol{\Sigma}_\ell^{\mathbf{u}} \boldsymbol{\Lambda}_\ell \mathbf{v}_\lambda^\ell, \boldsymbol{\Sigma}_\ell^{\mathbf{u}}\right), \qquad \boldsymbol{\Sigma}_\ell^{\mathbf{u}} = \left(\mathbf{K}^{-1}\left(\mathbf{U}_{\ell-1}\right) + \boldsymbol{\Lambda}_\ell\right)^{-1}, \tag{71}$$

and $\mathrm{P}\left(\mathbf{F}_\ell|\mathbf{U}_\ell, \mathbf{U}_{\ell-1}, \mathbf{F}_{\ell-1}\right)$ is given by standard manipulations of Eq. (69). Importantly, the prior can be factorised in an analogous fashion,

$$\mathrm{P}\left(\mathbf{F}_\ell, \mathbf{U}_\ell|\mathbf{F}_{\ell-1}, \mathbf{U}_{\ell-1}\right) = \mathrm{P}\left(\mathbf{F}_\ell|\mathbf{U}_\ell, \mathbf{U}_{\ell-1}, \mathbf{F}_{\ell-1}\right) \mathrm{P}\left(\mathbf{U}_\ell|\mathbf{U}_{\ell-1}\right). \tag{72}$$

The full ELBO can be written as

$$\mathcal{L} = \mathbb{E}_{\mathrm{Q}\left(\{\mathbf{F}_\ell, \mathbf{U}_\ell\}_{\ell=1}^{L+1} | \mathbf{X}, \mathbf{U}_0\right)} \left[\log \frac{\mathrm{P}\left(\mathbf{Y}, \{\mathbf{F}_\ell, \mathbf{U}_\ell\}_{\ell=1}^{L+1} | \mathbf{X}, \mathbf{U}_0\right)}{\mathrm{Q}\left(\{\mathbf{F}_\ell, \mathbf{U}_\ell\}_{\ell=1}^{L+1} | \mathbf{X}, \mathbf{U}_0\right)}\right]. \tag{73}$$

Substituting Eqs. (68), (70) and (72) into Eq. (73), the $\mathrm{P}\left(\mathbf{F}_\ell|\mathbf{U}_\ell, \mathbf{U}_{\ell-1}, \mathbf{F}_{\ell-1}\right)$ terms cancel and we obtain,

$$\mathcal{L} = \mathbb{E}_{\mathrm{Q}\left(\{\mathbf{F}_\ell, \mathbf{U}_\ell\}_{\ell=1}^{L+1} | \mathbf{X}, \mathbf{U}_0\right)} \left[\log \mathrm{P}\left(\mathbf{Y}|\mathbf{F}_{L+1}\right) + \sum_{\ell=1}^{L+1} \log \frac{\mathrm{P}\left(\mathbf{U}_\ell|\mathbf{U}_{\ell-1}\right)}{\mathrm{Q}\left(\mathbf{U}_\ell|\mathbf{U}_{\ell-1}\right)}\right]. \tag{74}$$

Analogous to the BNN case, we evaluate the ELBO by alternatively sampling the inducing function values $\mathbf{U}_\ell$ given $\mathbf{U}_{\ell-1}$ and propagating the data by sampling from $\mathrm{P}\left(\mathbf{F}_\ell|\mathbf{U}_\ell, \mathbf{U}_{\ell-1}, \mathbf{F}_{\ell-1}\right)$ (see Alg. 2). As in Salimbeni & Deisenroth (2017), since all likelihoods we use factorise across datapoints, we only need to sample from the marginals of the latter distribution to sample the propagated function values. As in the BNN case, the parameters of the approximate posterior are the global inducing inputs, $\mathbf{U}_0$, and the pseudo-outputs and precisions at all layers, $\{\mathbf{V}_\ell, \boldsymbol{\Lambda}_\ell\}_{\ell=1}^{L}$. We can similarly use standard reparameterised variational inference (Kingma & Welling, 2013; Rezende et al., 2014) to optimise these variational parameters, as $\mathrm{Q}\left(\mathbf{U}_\ell|\mathbf{U}_{\ell-1}\right)$ is Gaussian. We use Adam (Kingma & Ba, 2014) to optimise the parameters.

Algorithm 2 Global inducing points for deep Gaussian processes

Parameters: $\mathbf{U}_0$, $\{\mathbf{V}_\ell, \mathbf{\Lambda}_\ell\}_{\ell=1}^L$.
Neural network inputs: $\mathbf{F}_0$
Neural network outputs: $\mathbf{F}_{L+1}$
$\mathcal{L} \leftarrow 0$
for ℓ **in** $\{1, \ldots, L+1\}$ **do**
  Compute the mean and covariance over the inducing outputs at this layer
  $\mathbf{\Sigma}_\ell^{\mathbf{u}} = \left(\mathbf{K}^{-1}\left(\mathbf{U}_{\ell-1}\right) + \mathbf{\Lambda}_\ell\right)^{-1}$
  $\mathbf{M}_\ell = \mathbf{\Sigma}_\ell^{\mathbf{u}} \mathbf{\Lambda}_\ell \mathbf{V}_\ell$
  Sample the inducing outputs and compute the ELBO
  $\mathbf{U}_\ell \sim \mathcal{N}\left(\mathbf{M}_\ell, \mathbf{\Sigma}_\ell^{\mathbf{u}}\right) = \mathrm{Q}\left(\mathbf{U}_\ell | \mathbf{U}_{\ell-1}\right)$
  $\mathcal{L} \leftarrow \mathcal{L} + \log \mathrm{P}\left(\mathbf{U}_\ell | \mathbf{U}_{\ell-1}\right) - \log \mathcal{N}\left(\mathbf{U}_\ell | \mathbf{M}_\ell, \mathbf{\Sigma}_\ell^{\mathbf{u}}\right)$
  Propagate the inputs using the sampled inducing outputs,
  $\mathbf{F}_\ell \sim \mathrm{P}\left(\mathbf{F}_\ell | \mathbf{U}_\ell, \mathbf{U}_{\ell-1}, \mathbf{F}_{\ell-1}\right)$
end for
$\mathcal{L} \leftarrow \mathcal{L} + \log \mathrm{P}\left(\mathbf{Y} | \mathbf{F}_{L+1}\right)$

F. Motivating the approximate posterior for deep GPs

Our original motivation for the approximate posterior was for the BNN case, which we then extended to deep GPs. Here, we show how the same approximate posterior can be motivated from a deep GP perspective. As with the BNN case, we first derive the form of the optimal approximate posterior for the last layer, in the regression case. Without inducing points, the ELBO becomes

$$\mathcal{L} = \mathbb{E}_{\mathrm{Q}\left(\{\mathbf{F}_\ell\}_{\ell=1}^{L+1}\right)}\left[\log \frac{\mathrm{P}\left(\mathbf{Y}, \{\mathbf{F}_\ell\}_{\ell=1}^{L+1}\right)}{\mathrm{Q}\left(\{\mathbf{F}_\ell\}_{\ell=1}^{L+1}\right)}\right], \tag{75}$$

where we have defined a generic variational posterior $\mathrm{Q}\left(\{\mathbf{F}_\ell\}_{\ell=1}^{L+1}\right)$. Since we are interested in the form of $\mathrm{Q}\left(\mathbf{F}_{L+1} \middle| \{\mathbf{F}_\ell\}_{\ell=1}^{L}\right)$, we rearrange the ELBO so that all terms that do not depend on $\mathbf{F}_{L+1}$ are absorbed into a constant, c:

$$\mathcal{L} = \mathbb{E}_{\mathrm{Q}\left(\{\mathbf{F}_\ell\}_{\ell=1}^{L+1}\right)}\left[\log \mathrm{P}\left(\mathbf{Y}, \mathbf{F}_{L+1} \middle| \{F_\ell\}_{\ell=1}^{L}\right) - \log \mathrm{Q}\left(\mathbf{F}_{L+1}\right) + c\right]. \tag{76}$$

Some straightforward rearrangements lead to a similar form to before,

$$\mathcal{L} = \mathbb{E}_{\mathrm{Q}\left(\{\mathbf{F}_\ell\}_{\ell=1}^{L}\right)}\left[\log \mathrm{P}\left(\mathbf{Y} \middle| \{\mathbf{F}_\ell\}_{\ell=1}^{L}\right) - D_{KL}\left(\mathrm{Q}\left(\mathbf{F}_{L+1} | \{\mathbf{F}_\ell\}_{\ell=1}^{L}\right) || \, \mathrm{P}\left(\mathbf{F}_{L+1} \middle| \mathbf{Y}, \{\mathbf{F}_\ell\}_{\ell=1}^{L}\right)\right) + c\right], \tag{77}$$

from which we see that the optimal conditional posterior is given by $\mathrm{Q}\left(\mathbf{F}_{L+1} | \{\mathbf{F}_\ell\}_{\ell=1}^{L}\right) = \mathrm{Q}\left(\mathbf{F}_{L+1} | \mathbf{F}_L\right) = \mathrm{P}\left(\mathbf{F}_{L+1} \middle| \mathbf{Y}, \mathbf{F}_L\right)$, which has a closed form for regression: it is simply the standard GP posterior given by training data $\mathbf{Y}$ at inputs $\mathbf{F}_L$. In particular, for likelihood

$$\mathrm{P}\left(\mathbf{Y} | \mathbf{F}_{L+1}, \mathbf{\Lambda}_{L+1}\right) = \prod_{\lambda=1}^{N_{L+1}} \mathcal{N}\left(\mathbf{y}_\lambda^{L+1} \middle| \mathbf{f}_\lambda^{L+1}, \mathbf{\Lambda}_{L+1}^{-1}\right), \tag{78}$$

where $\mathbf{\Lambda}_{L+1}$ is the precision,

$$\mathrm{Q}\left(\mathbf{F}_{L+1} | \mathbf{F}_L\right) = \prod_{\lambda=1}^{N_{L+1}} \mathcal{N}\left(\mathbf{f}_\lambda^{L+1} \middle| \mathbf{\Sigma}_{L+1}^{\mathbf{f}} \mathbf{\Lambda}_{L+1} \mathbf{y}_\lambda^{L+1}, \mathbf{\Sigma}_{L+1}^{\mathbf{f}}\right), \tag{79}$$

$$\mathbf{\Sigma}_{L+1}^{\mathbf{f}} = \left(\mathbf{K}^{-1}\left(\mathbf{F}_L\right) + \mathbf{\Lambda}_{L+1}\right)^{-1}. \tag{80}$$

This can be understood as kernelized Bayesian linear regression conditioned on the features from the previous layers. Finally, as is usual in GPs (Rasmussen & Williams, 2006), the predictive distribution for test points can be obtained by conditioning using this approximate posterior.

G. Comparing our deep GP approximate posterior to previous work

The standard approach to inference in deep GPs (e.g. Salimbeni & Deisenroth, 2017) involves local inducing points and an approximate posterior over $\{\mathbf{U}_\ell\}_{\ell=1}^{L+1}$ that is factorised across layers,

$$\mathrm{Q}\left(\{\mathbf{U}_\ell\}_{\ell=1}^{L+1}\right) = \prod_{l=1}^{L+1} \prod_{\lambda=1}^{N_\ell} \mathcal{N}\left(\mathbf{u}_\lambda^\ell; \mathbf{m}_\lambda^\ell, \mathbf{\Sigma}_\lambda^\ell\right). \tag{81}$$

This approximate posterior induces an approximate posterior over the underlying infinite-dimensional functions $\mathcal{F}_\ell$ at each layer, which are implicitly used to propagate the data through the network via $\mathbf{F}_\ell = \mathcal{F}_\ell(\mathbf{F}_{\ell-1})$. We show a graphical model summarising the standard approach in Fig. A1A. While, as Salimbeni & Deisenroth (2017) point out, the function values $\{\mathbf{F}_\ell\}_{\ell=1}^{L+1}$ are correlated, the functions $\{\mathcal{F}_\ell\}_{\ell=1}^{L+1}$ themselves are independent across layers. We note that for BNNs, this is equivalent to having a posterior over weights that factorises across layers.

One approach to introduce dependencies across layers for the functions would be to introduce the notion of global inducing points, propagating the initial $\mathbf{U}_0$ through the model. In fact, as we note in the Related Work section, Ustyuzhaninov et al. (2020) independently proposed this approach to introducing dependencies, using a toy problem to motivate the approach. However, they kept the form of the approximate posterior the same as the standard approach (Eq. 21); we show the corresponding graphical model in Fig. A1B. The graphical model shows that as adjacent functions $\mathcal{F}_\ell$ and $\mathcal{F}_{\ell+1}$ share the parent node $\mathbf{U}_\ell$, they are in fact dependent. However, non-adjacent functions do not share any parent nodes, and so are independent; this can be seen by considering the d-separation criterion (Pearl, 1988) for $\mathcal{F}_{\ell-1}$ and $\mathcal{F}_{\ell+1}$, which have parents $(\mathbf{U}_{\ell-2}, \mathbf{U}_{\ell-1})$ and $(\mathbf{U}_\ell, \mathbf{U}_{\ell+1})$ respectively.

Our approach, by contrast, determines the form of the approximate posterior over $\mathbf{U}_\ell$ by performing Bayesian regression using $\mathbf{U}_{\ell-1}$ as input to that layer's GP, where the output data is $\mathbf{V}_\ell$. This results in a posterior that depends on the previous layer, $\mathrm{Q}\left(\mathbf{U}_\ell|\mathbf{U}_{\ell-1}\right)$. We show the corresponding graphical model in Fig. A1C. From this graphical model it is straightforward to see that our approach results in a posterior over functions that are correlated across all layers.

H. Parameter scaling for ADAM

The standard optimiser for variational BNNs is ADAM (Kingma & Ba, 2014), which we also use. Considering similar RMSprop updates for simplicity (Tieleman & Hinton, 2012),

$$\Delta w = \eta \frac{g}{\sqrt{\mathbb{E}\left[g^2\right]}} \tag{82}$$

where the expectation over g^2 is approximated using a moving-average of past gradients. Thus, absolute parameter changes are going to be of order η. This is fine if all the parameters have roughly the same order of magnitude, but becomes a serious problem if some of the parameters are very large and others are very small. For instance, if a parameter is around 10^{-4} and $\eta = 10^{-4}$, then a single ADAM step can easily double the parameter estimate, or change it from positive to negative. In contrast, if a parameter is around 1, then ADAM, with η^{-4} can make proportionally much smaller changes to this parameter, (around 0.01%). Thus, we need to ensure that all of our parameters have the same scale, especially as we mix methods, such as combining factorised and global inducing points. We thus design all our new approximate posteriors (i.e. the inducing inputs and outputs) such that the parameters have a scale of around 1. The key issue is that the mean weights in factorised methods tend to be quite small — they have scale around $1/\sqrt{\text{fan-in}}$. To resolve this issue, we store scaled weights, and we divide these stored, scaled mean parameters by the fan-in as part of the forward pass,

$$\text{weights} = \frac{\text{scaled weights}}{\sqrt{\text{fan-in}}}. \tag{83}$$

This scaling forces us to use larger learning rates than are typically used.

I. Exploring the effect of the number of inducing points

In this section, we briefly consider the effect of changing the number of inducing points, M, used in global inducing. We reconsider the toy problem from Sec. 2.3, and plot predictive posteriors obtained with global inducing as the number of inducing points increases from 2 to 40 (noting that in Fig. 1 we used 100 inducing points).

We plot the results of our experiment in Fig. A2. While two inducing points are clearly not sufficient, we observe that there is remarkably very little difference between the predictive posteriors for 10 or more inducing points. This observation is reflected in the ELBOs per datapoint (listed above each plot), which show that adding more points beyond 10 gains very little in terms of closeness to the true posterior.

However, we note that this is a very simple dataset: it consists of only two clusters of close points with a very clear trend. Therefore, we would expect that for more complex datasets more inducing points would be necessary. We leave a full

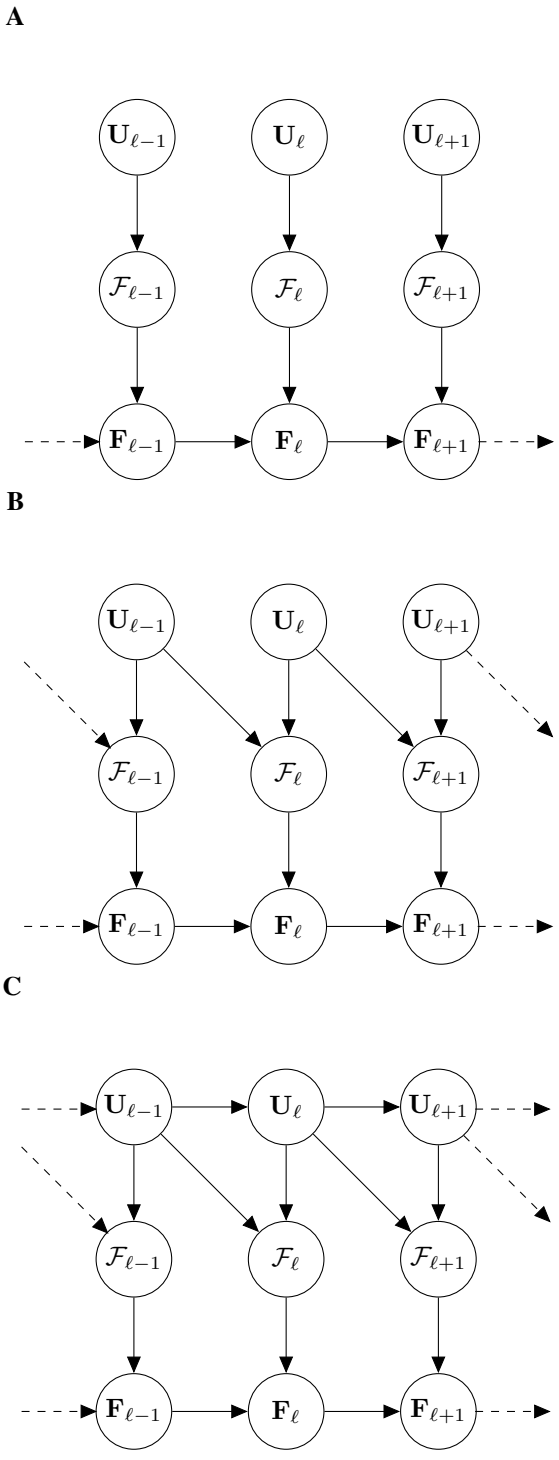

Figure A1. Comparison of the graphical models for three approaches to inference in deep GPs: Salimbeni & Deisenroth (2017), Ustyuzhaninov et al. (2020), and ours. **A** Graphical model illustrating the approach of Salimbeni & Deisenroth (2017). The inducing inputs, $\{\mathbf{U}_{\ell-1}\}_{\ell=1}^{L+1}$ are treated as learned parameters and are therefore omitted from the model. **B** Graphical model illustrating the approach of Ustyuzhaninov et al. (2020). The inducing inputs are given by the inducing outputs at the previous layer. **C** Graphical model illustrating our approach.

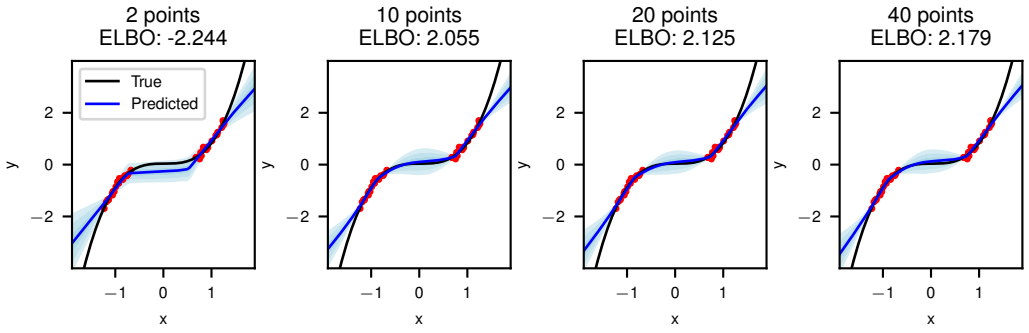

Figure A2. Predictive distributions on the toy dataset as the number of inducing points changes.

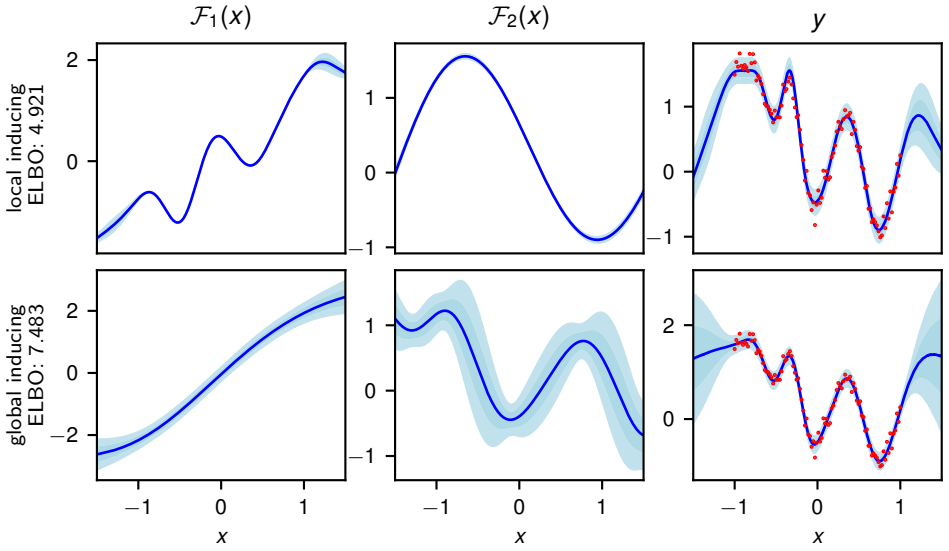

Figure A3. Posterior distributions for 2-layer DGPs with local inducing and global inducing. The first two columns show the predictive distributions for each layer taken individually, while the last column shows the predictive distribution of the output y.

investigation of how many inducing points are required to obtain a suitable approximate posterior, such as that found in Burt et al. (2020) for sparse GP regression, to future work.

J. Understanding compositional uncertainty

In this section, we take inspiration from the experiments of Ustyuzhaninov et al. (2020) which investigate the compositional uncertainty obtained by different approximate posteriors for DGPs. They noted that methods which factorise over layers have a tendency to cause the posterior distribution for each layer to collapse to a (nearly) deterministic function, resulting in worse uncertainty quantification within layers and worse ELBOs. In contrast, they found that allowing the approximate posterior to have correlations between layers allows those layers to capture more uncertainty, resulting in better ELBOs and therefore a closer approximation to the true posterior. They argue that this then allows the model to better discover compositional structure in the data.

We first consider a toy problem consisting of 100 datapoints generated by sampling from a two-layer DGP of width one, with squared-exponential kernels in each layer. We then fit two two-layer DGPs to this data - one using local inducing, the other

Table 2. ELBOs and variances of the intermediate functions for a BNN fit to the toy data of Fig. 1.

	ELBO	$\mathbb{V}[\mathcal{F}_1]$	$\mathbb{V}[\mathcal{F}_2]$	$\mathbb{V}[\mathcal{F}_3]$
factorised	-4.585	0.0728	0.4765	0.1926
local inducing	-5.469	0.0763	0.4473	0.0643
global inducing	2.236	0.4820	0.4877	1.0820

using global inducing. The results of this experiment can be seen in Fig. A3, which show the final fit, along with the learned posteriors over intermediate functions $\mathcal{F}_1$ and $\mathcal{F}_2$. These results mirror those observed by Ustyuzhaninov et al. (2020) on a similar experiment: local inducing, which factorises over layers, collapses to a nearly deterministic posterior over the intermediate functions, whereas global inducing provides a much broader distribution over functions for the two layers. Therefore, global inducing leads to a wider range of plausible functions that could explain the data via composition, which can be important in understanding the data. We observe that this behaviour directly leads to better uncertainty quantification for the out-of-distribution region, as well as better ELBOs.

To illustrate a similar phenomenon in BNNs, we reconsider the toy problem of Sec. 2.3. As it is not meaningful to consider neural networks with only one hidden unit per layer, instead of plotting intermediate functions we instead look at the mean variance of the functions at random input points, following roughly the experiment Dutordoir et al. (2019) in Table 1. For each layer, we consider the quantity

$$\mathbb{E}_x \left[\frac{1}{N_\ell} \sum_{\lambda=1}^{N_\ell} \mathbb{V}[f_\lambda^l(x)] \right], \tag{84}$$

where the expectation is over random input points, which we sample from a standard normal. We expect that for methods which introduce correlations across layers, this quantity will be higher, as there will be a wider range of intermediate functions that could plausibly explain the data. We confirm this in Table 2, which indicates that global inducing leads to many more compositions of functions being considered as plausible explanations of the data. This is additionally reflected in the ELBO, which is far better for global inducing than the other, factorised methods. However, we note that the variances are far closer than we might otherwise expect for the second layer. We hypothesise that this is due to the pruning effects described in Trippe & Turner (2018), where a layer has many weights that are close to the prior that are then pruned out by the following layer by having the outgoing weights collapse to zero. In fact, we note that the variances in the last layer are small for the factorised methods, which supports this hypothesis. By contrast, global inducing leads to high variances across all layers.

We believe that understanding the role of compositional uncertainty in variational inference for deep Bayesian models can lead to important conclusions about both the models being used and the compositional structure underlying the data being modelled, and is therefore an important direction for future work to consider.

K. UCI results with Bayesian neural networks

For this Appendix, we consider all of the UCI datasets from (Hernández-Lobato & Adams, 2015), along with four approximation families: factorised (i.e. mean-field), local inducing, global inducing, and fac→gi, which may offer some computational advantages to global inducing. We also considered three priors: the standard $\mathcal{N}(0, 1)$ prior, NealPrior, and ScalePrior. The test LLs and ELBOs for BNNs applied to UCI datasets are given in Fig. A4. Note that the ELBOs for the global inducing methods (both global inducing and fac→gi) are almost always better than those for baseline methods, often by a very large margin. However, as noted earlier, this does not necessarily correspond to better test log likelihoods due to model misspecification: there is not a straightforward relationship between the ELBO and the predictive performance, and so it is possible to obtain better test log likelihoods with worse inference. We present all the results, including for the test RMSEs, in tabulated form in Appendix P.

K.1. Experimental details

The architecture we considered for all BNN UCI experiments were fully-connected ReLU networks with 2 hidden layers of 50 hidden units each. We performed a grid search to select the learning rate and minibatch size. For the fully factorised approximation, we selected the learning rate from $\{3\text{e-}4, 1\text{e-}3, 3\text{e-}3, 1\text{e-}2\}$ and the minibatch size from $\{32, 100, 500\}$,

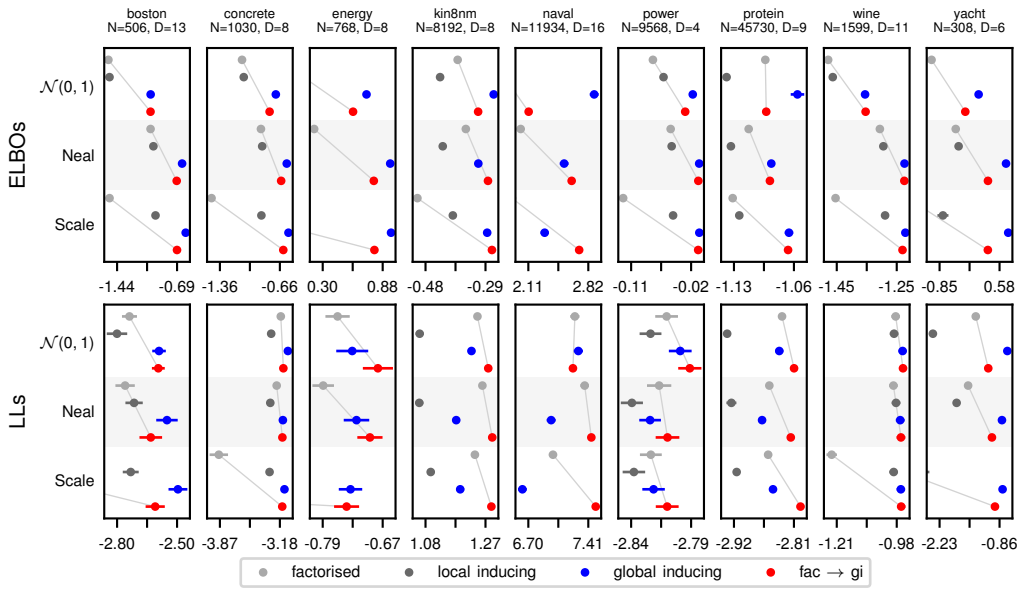

Figure A4. ELBOs per datapoint and average test log likelihoods for BNNs on UCI datasets.

optimising for 25000 gradient steps; for the other methods we selected the learning rate from $\{3\text{e-}3, 1\text{e-}2\}$ and fixed the minibatch size to 10000 (as in Salimbeni & Deisenroth, 2017), optimising for 10000 gradient steps. For all methods we selected the hyperparameters that gave the best ELBO. We trained the models using 10 samples from the approximate posterior, while using 100 for evaluation. For the inducing point methods, we used the selected batch size for the number of inducing points per layer. For all methods, we initialised the log noise variance at -3, but use the scaling trick in Appendix H to accelerate convergence, scaling by a factor of 10. Note that for the fully factorised method we used the local reparameterisation trick (Kingma et al., 2015); however, for fac $\rightarrow$ gi we cannot do so because the inducing point methods require that covariances be propagated through the network correctly. For the inducing point methods, we additionally use output channel-specific precisions, $\mathbf{\Lambda}_{l,\lambda}$, which effectively allows the network to prune unnecessary neurons if that benefits the ELBO. However, we only parameterise the diagonal of these precision matrices to save on computational and memory cost.

L. Uncertainty calibration & out-of-distribution detection for CIFAR-10

To assess how well our methods capture uncertainty, we consider calibration, as well as the predictive entropy for out-of-distribution data. Calibration is assessed by comparing the model's probabilistic assessment of its confidence with its accuracy — the proportion of the time that it is actually correct. For instance, gathering model predictions with some confidence (e.g. softmax probabilities in the range 0.9 to 0.95), and looking at the accuracy of these predictions, we would expect the model to be correct with probability 0.925; a higher or lower value would represent miscalibration.

We begin by plotting calibration curves (for the small 'ResNet' model) in Fig. A5, obtained by binning the predictions in 20 equal bins and assessing the mean accuracy of the binned predictions. For well-calibrated models, we expect the line to lie on the diagonal. A line above the diagonal indicates the model is underconfident (the model is performing better than it expects), whereas a line below the diagonal indicates it is overconfident (it is performing worse than it expects). While it is difficult to draw strong conclusions from these plots, it appears generally that factorised is poorly calibrated for both priors, that SpatialIWPrior generally improves calibration over ScalePrior, and that local inducing with SpatialIWPrior performs very well.

To come to more quantitative conclusions, we use expected calibration error (ECE; (Naeini et al., 2015; Guo et al., 2017)), which measures the expected absolute difference between the model's confidence and accuracy. Confirming the results from

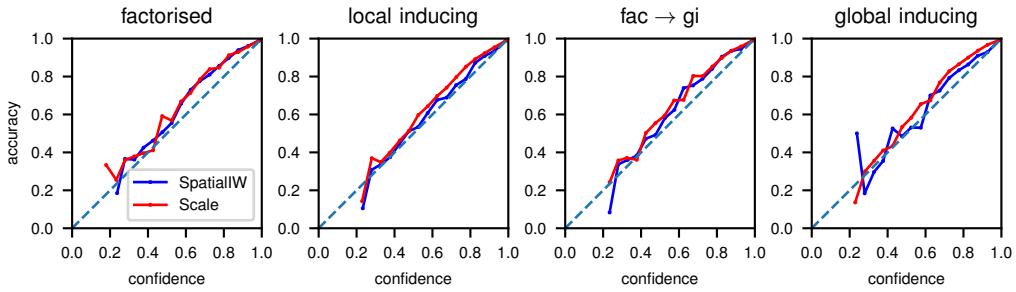

Figure A5. Calibration curves for CIFAR-10

Table 3. Expected calibration error for CIFAR-10

	factorised	local inducing	fac → gi	global inducing
ScalePrior	0.053	0.040	0.049	**0.038**
SpatialIWPrior	0.045	**0.018**	0.036	0.021

the plots (Fig. A5), we find that using the more sophisticated SpatialIWPrior gave considerable improvements in calibration. While, as expected, we find that our most accurate prior, SpatialIWPrior, in combination with global inducing points did very well (ECE of 0.021), the model with the best ECE is actually local inducing with SpatialIWPrior, albeit by a very small margin. We leave investigation of exactly why this is to future work. Finally, note our final ECE value of 0.021 is a considerable improvement over those for uncalibrated models in Guo et al. (2017) (Table 1), which are in the region of 0.03-0.045 (although considerably better calibration can be achieved by post-hoc scaling of the model's confidence).

In addition to calibration, we consider out-of-distribution performance. Given out-of-distribution data, we would hope that the network would give high output entropies (i.e. low confidence in the predictions), whereas for in-distribution data, we would hope for low entropy predictions (i.e. high confidence in the predictions). To evaluate this, we consider the mean predictive entropies for both the CIFAR-10 test set and the SVHN (Netzer et al., 2011) test set, when the network has been trained on CIFAR-10. We compare the ratio (CIFAR-10 entropy/ SVHN entropy) of these mean predictive entropies for each model in Table 4; a lower ratio indicates that the model is doing a better job of differentiating the datasets. We see that for both priors we considered, global inducing performs the best.

M. UCI results with deep Gaussian processes

In this appendix, we again consider all of the UCI datasets from Hernández-Lobato & Adams (2015) for DGPs with depths ranging from two to five layers. We compare DSVI (Salimbeni & Deisenroth, 2017), local inducing, and global inducing. While local inducing uses the same inducing-point architecture as Salimbeni & Deisenroth (2017), the actual implementation and parameterisation is very different. As such, we do expect to see differences between local inducing and Salimbeni & Deisenroth (2017).

We show our results in Fig. A6. Here, the results are not as clear-cut as in the BNN case. For the smaller datasets (i.e. boston, concrete, energy, wine, but with the notable exception of yacht), global inducing generally outperforms both local inducing and DSVI, as noted in the main text, especially when considering the ELBOs. We do however observe that for power, protein, yacht, and one model for kin8nm, the local approaches sometimes outperform global inducing, even for the ELBOs. We believe this is due to the fact that relatively few inducing points were used (100), in combination with the fact that global inducing has far fewer variational parameters than the local approaches. This may make optimisation harder in the global inducing case, especially for larger datasets where the model uncertainty does not matter as much, as the posterior concentration will be stronger. Importantly, however, our results on CIFAR-10 indicate that these issues do not arise in very large-scale, high-dimensional datasets, which are of most interest for future work. Surprisingly, local inducing generally significantly outperforms DSVI, even though they are different parameterisations of the same approximate posterior. We leave consideration of why this is for future work.

We provide tabulated results, including for RMSEs, in Appendix P.

Table 4. Predictive entropy ratios for CIFAR-10 & SVHN

	factorised	local inducing	fac → gi	global inducing
ScalePrior	0.506	0.534	0.462	**0.352**
SpatialIWPrior	0.461	0.480	0.412	**0.342**

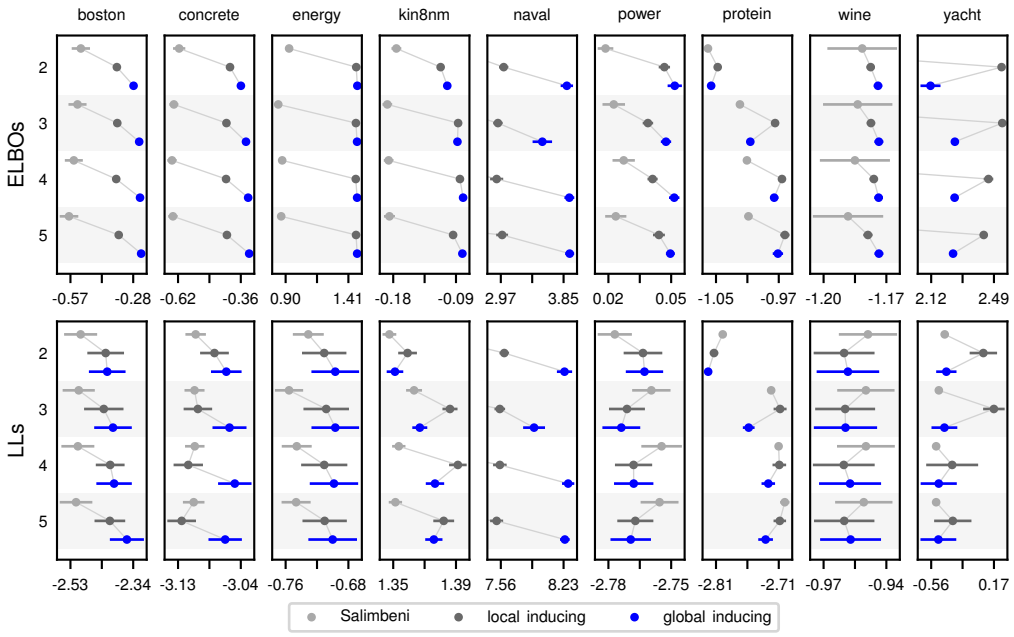

Figure A6. ELBOs per datapoint and average test log likelihoods for DGPs on UCI datasets. The numbers indicate the depths of the models.

M.1. Experimental details

Here, we matched the experimental setup in Salimbeni & Deisenroth (2017) as closely as possible. In particular, we used 100 inducing points, and full-covariance observation noise. However, our parameterisation is still somewhat different from theirs, in part because our approximate posterior is defined in terms of noisy function-values, while their approximate posterior was defined in terms of the function-values themselves.

As the original results in Salimbeni & Deisenroth (2017) used different UCI splits, and did not provide the ELBO, we reran their code `https://github.com/ICL-SML/Doubly-Stochastic-DGP` (changing the number of epochs and noise variance to reflect the values in the paper), which gave very similar log likelihoods to those in the paper.

N. MNIST 500

For MNIST, we considered a LeNet-inspired model consisting of two conv2d-relu-maxpool blocks, followed by conv2d-relu-linear, where the convolutions all have 3×3 kernels with 64 channels. We trained all models using a learning rate of 10^{-3}.

When training on very small datasets, such as the first 500 training examples in MNIST, we can see a variety of pathologies emerge with standard methods. To help build intuition for these pathologies, we introduce a sanity check for the ELBO. In particular, we could imagine a model that sets the distribution over all lower-layer parameters equal to the prior, and sets the top-layer parameters so as to ensure that the predictions are uniform. With 10 classes, this results in an average test log likelihood of $-2.30 \approx \log(1/10)$, and an ELBO (per datapoint) of approximately -2.30. We found that many combinations of the approximate posterior/prior converged to ELBOs near this baseline. Indeed, the only approximate posterior to escape

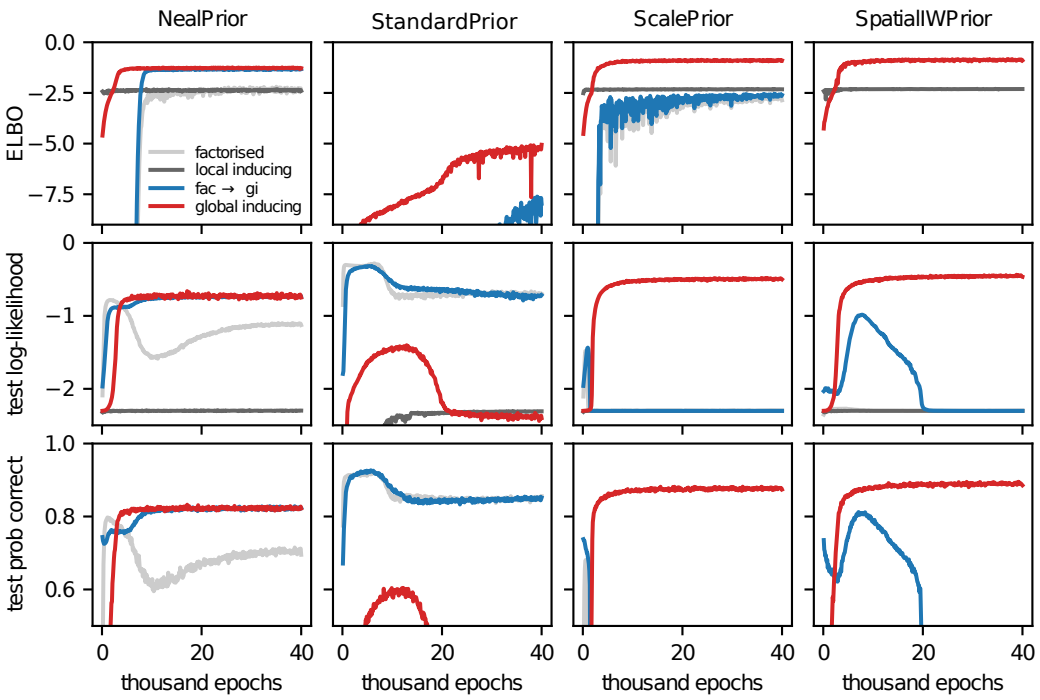

Figure A7. The ELBO, test log likelihoods and classification accuracy with different priors and approximate posteriors on a reduced MNIST dataset consisting of only the first 500 training examples.

this baseline for ScalePrior and SpatialIWPrior is global inducing points. This is because ScalePrior and SpatialIWPrior both offer the flexibility to shrink the prior variance, and hence shrink the weights towards zero, giving uniform predictions, and potentially zero KL divergence. In contrast, NealPrior and StandardPrior do not offer this flexibility: you always have to pay something in KL divergence in order to give uniform predictions. We believe that this is the reason that factorised performs better than expected with NealPrior, despite having an ELBO that is close to the baseline. Furthermore, it is unclear why local inducing gives very test log likelihood and performance, despite having an ELBO that is similar to factorised. For StandardPrior, all the ELBOs are far lower than the baseline, and far lower than for any other priors. Despite this, factorised and fac $\rightarrow$ gi in combination with StandardPrior appear to transiently perform better in terms of predictive accuracy than any other method. These results should sound a note of caution whenever we try to use factorised approximate posteriors with fixed prior covariances (e.g. Blundell et al., 2015; Farquhar et al., 2020). We leave a full investigation of these effects for future work.

O. Additional experimental details

All the methods were implemented in PyTorch. We ran the toy experiments on CPU, with the UCI experiments being run on a mixture of CPU and GPU. The remaining experiments – linear, CIFAR-10, and MNIST 500 – were run on various GPUs. For CIFAR-10, the most intensive of our experiments, we trained the models on one NVIDIA Tesla P100-PCIE-16GB. We optimised using ADAM (Kingma & Ba, 2014) throughout.

Factorised We initialise the posterior weight means to follow the Neal scaling (i.e. drawn from NealPrior); however, we use the scaling described in Appendix H to accelerate training. We initialise the weight variances to 1e-3$/\sqrt{N_{\ell-1}}$ for each layer.

Inducing point methods For global inducing, we initialise the inducing inputs, $\mathbf{U}_0$, and pseudo-outputs for the last layer, V_{L+1}, using the first batch of data, except for the toy experiment, where we initialise using samples from $\mathcal{N}(0, 1)$ (since we

used more inducing points than datapoints). For the remaining layers, we initialise the pseudo-outputs by sampling from $\mathcal{N}(0, 1)$. We initialise the log precision to -4, except for the last layer, where we initialise it to 0. We additionally use a scaling factor of 3 as described in Appendix H. For local inducing, the initialisation is largely the same, except we initialise the pseudo-outputs for every layer by sampling from $\mathcal{N}(0, 1)$. We additionally sample the inducing inputs for every layer from $\mathcal{N}(0, 1)$.

Toy experiment For each variational method, we optimise the ELBO over 5000 epochs, using full batches for the gradient descent. We use a learning rate of 1e-2. We fix the noise variance at its true value, to help assess the differences between each method more clearly. We use 10 samples from the variational posterior for training, using 100 for testing. For HMC, we use 10000 samples to burn in, and 10000 samples for evaluation, which we subsequently thin by a factor of 10. We initialise the samples from a standard normal distribution, and use 20 leapfrog steps for each sample. We hand-tune the leapfrog step sizes to be 0.0007 and 0.003 for the burn-in and sampling phases, respectively.

Deep linear network We use 10 inducing points for the inducing point methods. We use 1 sample from the approximate posterior for training and 10 for testing, training for 40 periods of 1000 gradient steps, using full batches for each step, with a learning rate of 1e-2.

UCI experiments The splits that we used for the UCI datasets can be found at `https://github.com/yaringal/DropoutUncertaintyExps`.

CIFAR-10 The CIFAR-10 dataset (`https://www.cs.toronto.edu/~kriz/cifar.html`; (Krizhevsky et al., 2009)) is a 10-class dataset comprising RGB, 32×32 images. It is divided in two sets: a training set of 50,000 examples, and a validation set of 10,000 examples. For the purposes of this paper, we use the validation set as our test set and refer to it as such, as is commonly done in the literature. We use a batch size of 500, with one sample from the approximate posterior for training and 10 for testing. For pre-processing, we normalise the data using the training dataset's mean and standard deviation. Finally, we train for 1000 epochs with a learning rate of 1e-2 (see App. H for an explanation of why our learning rate is higher than might be expected), and we use a tempering scheme for the first 100 epochs, slowly increasing the influence of the KL divergence to the prior by multiplying it by a factor that increases from 0 to 1. In our scheme, we increase the factor in a step-wise manner, meaning that for the first ten epochs it is 0, then 0.1 for the next ten, 0.2 for the following ten, and so on. Importantly, we still have 900 epochs of training where the standard, untempered ELBO is used, meaning that our results reflect that ELBO. Finally, we note that we share the precisions $\mathbf{\Lambda}_\ell$ within layers instead of using a separate precision for each output channel as was done in the UCI case. This saves memory and computational cost although possibly at the expense of predictive performance. For the full ResNet-18 experiments, we use the same training procedure. We do not use batch normalization (Ioffe & Szegedy, 2015) for the fully factorised case, since batch normalization is difficult to interpret in a Bayesian manner as it treats training and testing points differently. However, for the inducing point methods, we use a modified version of batch normalization that computes batch statistics from the inducing data, which do not change between training and testing. For the random cropping, we use zero padding of four pixels on each edge (resulting in 40×40 images), and randomly crop out a 32×32 image.

MNIST 500 The MNIST dataset (`http://yann.lecun.com/exdb/mnist/`) is a dataset of grayscale handwritten digits, each 28×28 pixels, with 10 classes. It comprises 60,000 training images and 10,000 test images. For the MNIST 500 experiments, we trained using the first 500 images from the training dataset and discarded the rest. We normalised the images using the full training dataset's statistics.

Deep GPs As mentioned, we largely follow the approach of Salimbeni & Deisenroth (2017) for hyperparameters. For global inducing, we initialise the inducing inputs to the first batch of training inputs, and we initialise the pseudo-outputs for the last layer to the respective training outputs. For the remaining layers, we initialise the pseudo-outputs by sampling from a standard normal distribution. We initialise the precision matrix to be diagonal with log precision zero for the output layer, and log precision -4 for the remaining layers. For local inducing, we initialise inducing inputs and pseudo-outputs by sampling from a standard normal for every layer, and initialise the precision matrices to be diagonal with log precision zero.

P. Tables of UCI Results

Table 5. Average test log likelihoods in nats for BNNs on UCI datasets (errors are ± 1 standard error)

	factorised	local inducing	global inducing	factorised → global
boston - $\mathcal{N}(0,1)$	-2.74 ± 0.03	-2.80 ± 0.04	$\mathbf{-2.59 \pm 0.03}$	-2.60 ± 0.02
NealPrior	-2.76 ± 0.04	-2.71 ± 0.04	$\mathbf{-2.55 \pm 0.05}$	-2.63 ± 0.05
ScalePrior	-3.63 ± 0.03	-2.73 ± 0.03	$\mathbf{-2.50 \pm 0.04}$	-2.61 ± 0.04
concrete - $\mathcal{N}(0,1)$	-3.17 ± 0.02	-3.28 ± 0.01	$\mathbf{-3.08 \pm 0.01}$	-3.14 ± 0.01
NealPrior	-3.21 ± 0.01	-3.29 ± 0.01	$\mathbf{-3.14 \pm 0.01}$	-3.15 ± 0.02
ScalePrior	-3.89 ± 0.09	-3.30 ± 0.01	$\mathbf{-3.12 \pm 0.01}$	-3.15 ± 0.01
energy - $\mathcal{N}(0,1)$	-0.76 ± 0.02	-1.75 ± 0.01	-0.73 ± 0.03	$\mathbf{-0.68 \pm 0.03}$
NealPrior	-0.79 ± 0.02	-2.06 ± 0.09	-0.72 ± 0.02	$\mathbf{-0.70 \pm 0.02}$
ScalePrior	-2.55 ± 0.01	-2.42 ± 0.02	$\mathbf{-0.73 \pm 0.02}$	-0.74 ± 0.02
kin8nm - $\mathcal{N}(0,1)$	1.24 ± 0.01	1.06 ± 0.01	1.22 ± 0.01	$\mathbf{1.28 \pm 0.01}$
NealPrior	1.26 ± 0.01	1.06 ± 0.01	1.18 ± 0.01	$\mathbf{1.29 \pm 0.01}$
ScalePrior	1.23 ± 0.01	1.10 ± 0.01	1.19 ± 0.01	$\mathbf{1.29 \pm 0.00}$
naval - $\mathcal{N}(0,1)$	7.25 ± 0.04	6.06 ± 0.10	$\mathbf{7.29 \pm 0.04}$	7.23 ± 0.02
NealPrior	7.37 ± 0.03	4.28 ± 0.37	6.97 ± 0.04	$\mathbf{7.45 \pm 0.02}$
ScalePrior	6.99 ± 0.03	2.80 ± 0.00	6.63 ± 0.04	$\mathbf{7.50 \pm 0.02}$
power - $\mathcal{N}(0,1)$	-2.81 ± 0.01	-2.82 ± 0.01	-2.80 ± 0.01	$\mathbf{-2.79 \pm 0.01}$
NealPrior	$\mathbf{-2.81 \pm 0.01}$	-2.84 ± 0.01	-2.82 ± 0.01	$\mathbf{-2.81 \pm 0.01}$
ScalePrior	-2.82 ± 0.01	-2.84 ± 0.01	-2.82 ± 0.01	$\mathbf{-2.81 \pm 0.01}$
protein - $\mathcal{N}(0,1)$	-2.83 ± 0.00	-2.93 ± 0.00	-2.84 ± 0.00	$\mathbf{-2.81 \pm 0.00}$
NealPrior	-2.86 ± 0.00	-2.92 ± 0.01	-2.87 ± 0.00	$\mathbf{-2.82 \pm 0.00}$
ScalePrior	-2.86 ± 0.00	-2.91 ± 0.00	-2.85 ± 0.00	$\mathbf{-2.80 \pm 0.00}$
wine - $\mathcal{N}(0,1)$	-0.98 ± 0.01	-0.99 ± 0.01	$\mathbf{-0.96 \pm 0.01}$	$\mathbf{-0.96 \pm 0.01}$
NealPrior	-0.99 ± 0.01	-0.98 ± 0.01	-0.97 ± 0.01	$\mathbf{-0.96 \pm 0.01}$
ScalePrior	-1.22 ± 0.01	-0.99 ± 0.01	$\mathbf{-0.96 \pm 0.01}$	$\mathbf{-0.96 \pm 0.01}$
yacht - $\mathcal{N}(0,1)$	-1.41 ± 0.05	-2.39 ± 0.05	$\mathbf{-0.68 \pm 0.03}$	-1.12 ± 0.02
NealPrior	-1.58 ± 0.04	-1.84 ± 0.05	$\mathbf{-0.81 \pm 0.03}$	-1.04 ± 0.01
ScalePrior	-4.12 ± 0.03	-2.71 ± 0.22	$\mathbf{-0.79 \pm 0.02}$	-0.97 ± 0.05

Table 6. Test RMSEs for BNNs on UCI datasets (errors are ± 1 standard error)

	factorised	local inducing	global inducing	factorised → global
boston - $\mathcal{N}(0,1)$	3.60 ± 0.21	3.85 ± 0.26	$\mathbf{3.13 \pm 0.20}$	3.14 ± 0.20
NealPrior	3.64 ± 0.24	3.55 ± 0.23	$\mathbf{3.14 \pm 0.18}$	3.33 ± 0.21
ScalePrior	9.03 ± 0.26	3.57 ± 0.20	$\mathbf{2.97 \pm 0.19}$	3.27 ± 0.20
concrete - $\mathcal{N}(0,1)$	5.73 ± 0.11	6.34 ± 0.11	$\mathbf{5.39 \pm 0.09}$	5.55 ± 0.10
NealPrior	5.96 ± 0.11	6.35 ± 0.12	$\mathbf{5.64 \pm 0.10}$	5.70 ± 0.11
ScalePrior	12.66 ± 1.04	6.48 ± 0.12	$\mathbf{5.56 \pm 0.10}$	5.68 ± 0.09
energy - $\mathcal{N}(0,1)$	0.51 ± 0.01	1.35 ± 0.02	0.50 ± 0.01	$\mathbf{0.47 \pm 0.02}$
NealPrior	0.51 ± 0.01	1.95 ± 0.14	0.49 ± 0.01	$\mathbf{0.47 \pm 0.01}$
ScalePrior	3.02 ± 0.05	2.67 ± 0.06	0.50 ± 0.01	$\mathbf{0.49 \pm 0.01}$
kin8nm - $\mathcal{N}(0,1)$	$\mathbf{0.07 \pm 0.00}$	0.08 ± 0.00	$\mathbf{0.07 \pm 0.00}$	$\mathbf{0.07 \pm 0.00}$
NealPrior	$\mathbf{0.07 \pm 0.00}$	0.08 ± 0.00	$\mathbf{0.07 \pm 0.00}$	$\mathbf{0.07 \pm 0.00}$
ScalePrior	$\mathbf{0.07 \pm 0.00}$	0.08 ± 0.00	$\mathbf{0.07 \pm 0.00}$	$\mathbf{0.07 \pm 0.00}$
naval - $\mathcal{N}(0,1)$	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
NealPrior	$\mathbf{0.00 \pm 0.00}$	0.01 ± 0.00	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
ScalePrior	$\mathbf{0.00 \pm 0.00}$	0.01 ± 0.00	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
power - $\mathcal{N}(0,1)$	4.00 ± 0.03	4.06 ± 0.03	3.96 ± 0.04	$\mathbf{3.93 \pm 0.04}$
NealPrior	4.03 ± 0.04	4.13 ± 0.03	4.06 ± 0.03	$\mathbf{4.00 \pm 0.04}$
ScalePrior	4.06 ± 0.04	4.12 ± 0.04	4.05 ± 0.04	$\mathbf{4.01 \pm 0.04}$
protein - $\mathcal{N}(0,1)$	4.12 ± 0.02	4.54 ± 0.02	4.14 ± 0.02	$\mathbf{4.04 \pm 0.01}$
NealPrior	4.21 ± 0.01	4.50 ± 0.03	4.27 ± 0.02	$\mathbf{4.06 \pm 0.01}$
ScalePrior	4.22 ± 0.02	4.46 ± 0.01	4.19 ± 0.02	$\mathbf{4.00 \pm 0.02}$
wine - $\mathcal{N}(0,1)$	0.65 ± 0.01	0.65 ± 0.01	$\mathbf{0.63 \pm 0.01}$	$\mathbf{0.63 \pm 0.01}$
NealPrior	0.66 ± 0.01	0.65 ± 0.01	$\mathbf{0.64 \pm 0.01}$	$\mathbf{0.64 \pm 0.01}$
ScalePrior	0.82 ± 0.01	0.65 ± 0.01	$\mathbf{0.64 \pm 0.01}$	$\mathbf{0.64 \pm 0.01}$
yacht - $\mathcal{N}(0,1)$	0.98 ± 0.07	2.35 ± 0.13	0.56 ± 0.04	$\mathbf{0.50 \pm 0.04}$
NealPrior	1.15 ± 0.07	1.37 ± 0.11	$\mathbf{0.57 \pm 0.04}$	0.63 ± 0.05
ScalePrior	14.55 ± 0.59	5.75 ± 1.34	$\mathbf{0.56 \pm 0.04}$	0.63 ± 0.04

Table 7. ELBOs per datapoint in nats for BNNs on UCI datasets (errors are ± 1 standard error)

	factorised	local inducing	global inducing	factorised → global
boston - $\mathcal{N}(0,1)$	-1.55 ± 0.00	-1.54 ± 0.00	$\mathbf{-1.02 \pm 0.01}$	-1.03 ± 0.00
NealPrior	-1.03 ± 0.00	-0.99 ± 0.00	$\mathbf{-0.63 \pm 0.00}$	-0.70 ± 0.00
ScalePrior	-1.54 ± 0.00	-0.96 ± 0.00	$\mathbf{-0.59 \pm 0.00}$	-0.70 ± 0.00
concrete - $\mathcal{N}(0,1)$	-1.10 ± 0.00	-1.08 ± 0.00	$\mathbf{-0.71 \pm 0.00}$	-0.78 ± 0.00
NealPrior	-0.88 ± 0.00	-0.87 ± 0.00	$\mathbf{-0.59 \pm 0.00}$	-0.65 ± 0.00
ScalePrior	-1.45 ± 0.01	-0.88 ± 0.00	$\mathbf{-0.57 \pm 0.00}$	-0.63 ± 0.00
energy - $\mathcal{N}(0,1)$	-0.13 ± 0.02	-0.53 ± 0.01	$\mathbf{0.72 \pm 0.00}$	0.59 ± 0.00
NealPrior	0.21 ± 0.00	-0.33 ± 0.04	$\mathbf{0.95 \pm 0.00}$	0.79 ± 0.01
ScalePrior	-1.12 ± 0.00	-0.47 ± 0.01	$\mathbf{0.96 \pm 0.01}$	0.80 ± 0.01
kin8nm - $\mathcal{N}(0,1)$	-0.38 ± 0.00	-0.43 ± 0.00	$\mathbf{-0.26 \pm 0.00}$	-0.31 ± 0.00
NealPrior	-0.35 ± 0.00	-0.43 ± 0.00	-0.31 ± 0.00	$\mathbf{-0.28 \pm 0.00}$
ScalePrior	-0.51 ± 0.00	-0.39 ± 0.01	-0.29 ± 0.00	$\mathbf{-0.27 \pm 0.00}$
naval - $\mathcal{N}(0,1)$	1.68 ± 0.01	1.65 ± 0.09	$\mathbf{2.89 \pm 0.04}$	2.11 ± 0.02
NealPrior	2.02 ± 0.02	-0.09 ± 0.34	2.53 ± 0.04	$\mathbf{2.62 \pm 0.02}$
ScalePrior	1.91 ± 0.03	-1.42 ± 0.00	2.30 ± 0.03	$\mathbf{2.71 \pm 0.02}$
power - $\mathcal{N}(0,1)$	-0.08 ± 0.00	-0.06 ± 0.00	$\mathbf{-0.02 \pm 0.00}$	-0.03 ± 0.00
NealPrior	-0.05 ± 0.00	-0.05 ± 0.00	$\mathbf{-0.01 \pm 0.00}$	$\mathbf{-0.01 \pm 0.00}$
ScalePrior	-0.13 ± 0.00	-0.05 ± 0.00	$\mathbf{-0.01 \pm 0.00}$	$\mathbf{-0.01 \pm 0.00}$
protein - $\mathcal{N}(0,1)$	-1.09 ± 0.00	-1.14 ± 0.00	$\mathbf{-1.06 \pm 0.01}$	-1.09 ± 0.00
NealPrior	-1.11 ± 0.00	-1.13 ± 0.00	$\mathbf{-1.09 \pm 0.00}$	$\mathbf{-1.09 \pm 0.00}$
ScalePrior	-1.13 ± 0.00	-1.12 ± 0.00	$\mathbf{-1.07 \pm 0.00}$	$\mathbf{-1.07 \pm 0.00}$
wine - $\mathcal{N}(0,1)$	-1.48 ± 0.00	-1.47 ± 0.00	$\mathbf{-1.36 \pm 0.00}$	$\mathbf{-1.36 \pm 0.00}$
NealPrior	-1.31 ± 0.00	-1.30 ± 0.00	$\mathbf{-1.22 \pm 0.00}$	-1.23 ± 0.00
ScalePrior	-1.46 ± 0.00	-1.29 ± 0.00	$\mathbf{-1.22 \pm 0.00}$	-1.23 ± 0.00
yacht - $\mathcal{N}(0,1)$	-1.04 ± 0.02	-1.30 ± 0.02	$\mathbf{0.08 \pm 0.01}$	-0.23 ± 0.01
NealPrior	-0.46 ± 0.02	-0.39 ± 0.01	$\mathbf{0.74 \pm 0.01}$	0.31 ± 0.01
ScalePrior	-1.61 ± 0.00	-0.77 ± 0.10	$\mathbf{0.79 \pm 0.01}$	0.30 ± 0.01

Table 8. Average test log likelihoods for our rerun of Salimbeni & Deisenroth (2017), and our implementations of local and global inducing points for deep GPs of various depths.

{dataset} - {depth}	DSVI	local inducing	global inducing
boston - 2	-2.50 ± 0.05	$\mathbf{-2.42 \pm 0.05}$	$\mathbf{-2.42 \pm 0.05}$
3	-2.51 ± 0.05	-2.43 ± 0.06	$\mathbf{-2.40 \pm 0.05}$
4	-2.51 ± 0.05	-2.41 ± 0.04	$\mathbf{-2.40 \pm 0.05}$
5	-2.51 ± 0.05	-2.41 ± 0.04	$\mathbf{-2.36 \pm 0.05}$
concrete - 2	-3.11 ± 0.01	-3.08 ± 0.02	$\mathbf{-3.06 \pm 0.02}$
3	-3.11 ± 0.01	-3.10 ± 0.02	$\mathbf{-3.06 \pm 0.02}$
4	-3.11 ± 0.01	-3.12 ± 0.02	$\mathbf{-3.05 \pm 0.02}$
5	-3.11 ± 0.01	-3.13 ± 0.02	$\mathbf{-3.06 \pm 0.02}$
energy - 2	-0.73 ± 0.02	-0.71 ± 0.03	$\mathbf{-0.70 \pm 0.03}$
3	-0.76 ± 0.02	-0.71 ± 0.03	$\mathbf{-0.70 \pm 0.03}$
4	-0.75 ± 0.02	-0.71 ± 0.03	$\mathbf{-0.70 \pm 0.03}$
5	-0.75 ± 0.02	-0.71 ± 0.03	$\mathbf{-0.70 \pm 0.03}$
kin8nm - 2	1.34 ± 0.00	$\mathbf{1.36 \pm 0.01}$	1.35 ± 0.00
3	1.36 ± 0.00	$\mathbf{1.38 \pm 0.00}$	1.36 ± 0.00
4	1.35 ± 0.00	$\mathbf{1.39 \pm 0.01}$	1.37 ± 0.01
5	1.35 ± 0.00	$\mathbf{1.38 \pm 0.01}$	1.37 ± 0.00
naval - 2	6.77 ± 0.07	7.59 ± 0.04	$\mathbf{8.24 \pm 0.07}$
3	6.61 ± 0.07	7.54 ± 0.04	$\mathbf{7.91 \pm 0.10}$
4	6.54 ± 0.14	7.54 ± 0.06	$\mathbf{8.28 \pm 0.05}$
5	5.02 ± 0.41	7.51 ± 0.06	$\mathbf{8.24 \pm 0.04}$
power - 2	-2.78 ± 0.01	$\mathbf{-2.76 \pm 0.01}$	$\mathbf{-2.76 \pm 0.01}$
3	$\mathbf{-2.76 \pm 0.01}$	-2.77 ± 0.01	-2.77 ± 0.01
4	$\mathbf{-2.75 \pm 0.01}$	-2.77 ± 0.01	-2.77 ± 0.01
5	$\mathbf{-2.75 \pm 0.01}$	-2.77 ± 0.01	-2.77 ± 0.01
protein - 2	$\mathbf{-2.80 \pm 0.00}$	-2.82 ± 0.00	-2.83 ± 0.00
3	-2.73 ± 0.00	$\mathbf{-2.71 \pm 0.01}$	-2.76 ± 0.01
4	$\mathbf{-2.71 \pm 0.01}$	-2.71 ± 0.01	-2.73 ± 0.01
5	$\mathbf{-2.70 \pm 0.01}$	-2.71 ± 0.01	-2.74 ± 0.01
wine - 2	$\mathbf{-0.95 \pm 0.01}$	-0.96 ± 0.01	-0.96 ± 0.01
3	$\mathbf{-0.95 \pm 0.01}$	-0.96 ± 0.01	-0.96 ± 0.01
4	$\mathbf{-0.95 \pm 0.01}$	-0.96 ± 0.01	-0.96 ± 0.01
5	$\mathbf{-0.95 \pm 0.01}$	-0.96 ± 0.01	-0.96 ± 0.01
yacht - 2	-0.40 ± 0.03	$\mathbf{0.05 \pm 0.14}$	-0.38 ± 0.10
3	-0.47 ± 0.02	$\mathbf{0.17 \pm 0.11}$	-0.41 ± 0.13
4	-0.50 ± 0.02	$\mathbf{-0.31 \pm 0.29}$	-0.47 ± 0.19
5	-0.50 ± 0.02	$\mathbf{-0.31 \pm 0.20}$	-0.48 ± 0.19

Table 9. Test RMSEs for our rerun of Salimbeni & Deisenroth (2017), and our implementations of local and global inducing points for deep GPs of various depths.

{dataset} - {depth}	DSVI	local inducing	global inducing
boston - 2	2.95 ± 0.18	$\mathbf{2.78 \pm 0.15}$	2.82 ± 0.14
3	2.98 ± 0.18	$\mathbf{2.78 \pm 0.14}$	2.79 ± 0.14
4	2.99 ± 0.18	$\mathbf{2.75 \pm 0.12}$	2.80 ± 0.15
5	3.01 ± 0.19	2.78 ± 0.13	$\mathbf{2.73 \pm 0.13}$
concrete - 2	5.51 ± 0.10	5.24 ± 0.11	$\mathbf{5.21 \pm 0.12}$
3	5.53 ± 0.10	5.38 ± 0.11	$\mathbf{5.18 \pm 0.12}$
4	5.50 ± 0.09	5.47 ± 0.11	$\mathbf{5.16 \pm 0.13}$
5	5.53 ± 0.11	5.50 ± 0.10	$\mathbf{5.23 \pm 0.13}$
energy - 2	0.50 ± 0.01	0.49 ± 0.01	$\mathbf{0.48 \pm 0.01}$
3	0.50 ± 0.01	0.49 ± 0.01	$\mathbf{0.48 \pm 0.01}$
4	0.50 ± 0.01	0.49 ± 0.01	$\mathbf{0.48 \pm 0.01}$
5	0.50 ± 0.01	0.49 ± 0.01	$\mathbf{0.48 \pm 0.01}$
kin8nm - 2	$\mathbf{0.06 \pm 0.00}$	$\mathbf{0.06 \pm 0.00}$	$\mathbf{0.06 \pm 0.00}$
3	$\mathbf{0.06 \pm 0.00}$	$\mathbf{0.06 \pm 0.00}$	$\mathbf{0.06 \pm 0.00}$
4	$\mathbf{0.06 \pm 0.00}$	$\mathbf{0.06 \pm 0.00}$	$\mathbf{0.06 \pm 0.00}$
5	$\mathbf{0.06 \pm 0.00}$	$\mathbf{0.06 \pm 0.00}$	$\mathbf{0.06 \pm 0.00}$
naval - 2	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
3	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
4	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
5	0.01 ± 0.00	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
power - 2	3.88 ± 0.03	3.82 ± 0.04	$\mathbf{3.81 \pm 0.04}$
3	$\mathbf{3.80 \pm 0.04}$	3.85 ± 0.04	3.87 ± 0.04
4	$\mathbf{3.78 \pm 0.04}$	3.84 ± 0.04	3.84 ± 0.04
5	$\mathbf{3.78 \pm 0.04}$	3.83 ± 0.04	3.84 ± 0.04
protein - 2	$\mathbf{4.01 \pm 0.01}$	4.07 ± 0.02	4.11 ± 0.01
3	3.75 ± 0.01	$\mathbf{3.73 \pm 0.03}$	3.88 ± 0.03
4	$\mathbf{3.73 \pm 0.01}$	$\mathbf{3.73 \pm 0.03}$	3.77 ± 0.03
5	$\mathbf{3.70 \pm 0.02}$	3.73 ± 0.03	3.80 ± 0.03
wine - 2	$\mathbf{0.06 \pm 0.00}$	$\mathbf{0.06 \pm 0.00}$	$\mathbf{0.06 \pm 0.00}$
3	$\mathbf{0.06 \pm 0.00}$	$\mathbf{0.06 \pm 0.00}$	$\mathbf{0.06 \pm 0.00}$
4	$\mathbf{0.06 \pm 0.00}$	$\mathbf{0.06 \pm 0.00}$	$\mathbf{0.06 \pm 0.00}$
5	$\mathbf{0.06 \pm 0.00}$	$\mathbf{0.06 \pm 0.00}$	$\mathbf{0.06 \pm 0.00}$
yacht - 2	0.40 ± 0.03	$\mathbf{0.36 \pm 0.03}$	0.41 ± 0.03
3	0.42 ± 0.03	0.37 ± 0.03	$\mathbf{0.36 \pm 0.03}$
4	0.44 ± 0.03	0.37 ± 0.03	$\mathbf{0.36 \pm 0.03}$
5	0.44 ± 0.03	0.40 ± 0.03	$\mathbf{0.35 \pm 0.03}$

Table 10. ELBOs per datapoint for our rerun of Salimbeni & Deisenroth (2017), and our implementations of local and global inducing points for deep GPs of various depths.

{dataset} - {depth}	DSVI	local inducing	global inducing
boston - 2	-0.52 ± 0.04	-0.35 ± 0.00	$\mathbf{-0.28 \pm 0.01}$
3	-0.54 ± 0.04	-0.35 ± 0.01	$\mathbf{-0.25 \pm 0.01}$
4	-0.55 ± 0.04	-0.36 ± 0.00	$\mathbf{-0.25 \pm 0.01}$
5	-0.58 ± 0.04	-0.35 ± 0.01	$\mathbf{-0.24 \pm 0.01}$
concrete - 2	-0.61 ± 0.02	-0.41 ± 0.00	$\mathbf{-0.36 \pm 0.00}$
3	-0.63 ± 0.01	-0.42 ± 0.00	$\mathbf{-0.34 \pm 0.00}$
4	-0.64 ± 0.01	-0.42 ± 0.00	$\mathbf{-0.33 \pm 0.00}$
5	-0.64 ± 0.01	-0.42 ± 0.00	$\mathbf{-0.33 \pm 0.00}$
energy - 2	0.93 ± 0.02	$\mathbf{1.48 \pm 0.00}$	$\mathbf{1.48 \pm 0.00}$
3	0.84 ± 0.02	1.47 ± 0.00	$\mathbf{1.48 \pm 0.00}$
4	0.87 ± 0.01	1.47 ± 0.00	$\mathbf{1.48 \pm 0.00}$
5	0.86 ± 0.01	1.47 ± 0.00	$\mathbf{1.48 \pm 0.00}$
kin8nm - 2	-0.18 ± 0.01	-0.11 ± 0.00	$\mathbf{-0.10 \pm 0.00}$
3	-0.19 ± 0.01	$\mathbf{-0.09 \pm 0.00}$	$\mathbf{-0.09 \pm 0.00}$
4	-0.19 ± 0.01	$\mathbf{-0.08 \pm 0.00}$	$\mathbf{-0.08 \pm 0.00}$
5	-0.19 ± 0.01	-0.09 ± 0.00	$\mathbf{-0.08 \pm 0.00}$
naval - 2	2.29 ± 0.08	3.01 ± 0.05	$\mathbf{3.89 \pm 0.07}$
3	2.07 ± 0.10	2.93 ± 0.05	$\mathbf{3.55 \pm 0.12}$
4	1.90 ± 0.25	2.92 ± 0.07	$\mathbf{3.93 \pm 0.05}$
5	0.61 ± 0.37	2.99 ± 0.06	$\mathbf{3.93 \pm 0.04}$
power - 2	0.02 ± 0.00	0.04 ± 0.00	$\mathbf{0.05 \pm 0.00}$
3	0.02 ± 0.00	$\mathbf{0.04 \pm 0.00}$	$\mathbf{0.04 \pm 0.00}$
4	0.03 ± 0.00	0.04 ± 0.00	$\mathbf{0.05 \pm 0.00}$
5	0.02 ± 0.00	$\mathbf{0.04 \pm 0.00}$	$\mathbf{0.04 \pm 0.00}$
protein - 2	-1.06 ± 0.00	$\mathbf{-1.05 \pm 0.00}$	-1.06 ± 0.00
3	-1.02 ± 0.00	$\mathbf{-0.98 \pm 0.00}$	-1.01 ± 0.00
4	-1.01 ± 0.00	$\mathbf{-0.97 \pm 0.00}$	-0.98 ± 0.00
5	-1.01 ± 0.00	$\mathbf{-0.97 \pm 0.00}$	$\mathbf{-0.97 \pm 0.00}$
wine - 2	-1.18 ± 0.02	$\mathbf{-1.17 \pm 0.00}$	$\mathbf{-1.17 \pm 0.00}$
3	-1.18 ± 0.02	$\mathbf{-1.17 \pm 0.00}$	$\mathbf{-1.17 \pm 0.00}$
4	-1.18 ± 0.02	$\mathbf{-1.17 \pm 0.00}$	$\mathbf{-1.17 \pm 0.00}$
5	-1.18 ± 0.02	-1.18 ± 0.00	$\mathbf{-1.17 \pm 0.00}$
yacht - 2	1.05 ± 0.06	$\mathbf{2.53 \pm 0.01}$	2.12 ± 0.05
3	1.00 ± 0.06	$\mathbf{2.54 \pm 0.01}$	2.26 ± 0.01
4	0.97 ± 0.06	$\mathbf{2.46 \pm 0.02}$	2.26 ± 0.01
5	0.95 ± 0.06	$\mathbf{2.43 \pm 0.01}$	2.25 ± 0.01