A Introduction to Variational Inference in Bayesian Neural Networks

A Bayesian neural network (BNN) places a distribution over the weights of a neural network [MacKay, 1992]. We hope to infer the posterior distribution over the weights given the data, \( p(\theta | D) \)—although ultimately we are interested in a posterior distribution over functions, as described by Sun et al. [2019].

Because this is intractable, we seek an approximate posterior \( q(\theta) \) to be as close as possible to the posterior over the weights. Variational inference (VI) is one method for estimating that approximate posterior, in which we pick an approximating distribution and minimize the KL-divergence between it and the true posterior. This KL divergence is the Evidence Lower Bound (ELBO), expressed using the prior over the weight distributions \( p(\theta) \). We therefore minimize the negative ELBO:

\[
L_{MFVI} = \text{KL}(q(\theta) \| p(\theta)) - \mathbb{E} \left[ \log p(y | x, \theta) \right].
\] (6)

For the full-covariance Gaussian approximate posterior [Barber and Bishop, 1998], the model weights for each layer \( \theta_i \) are distributed according to the multivariate Gaussian distribution \( \mathcal{N}(\mu_i, \Sigma_i) \). This is already a slight approximation, as it assumes layers are independent of each other. This is important for our analysis in §3 and §4 though note that the experiment in §6.1 does not assume any independence between layers.

The mean-field approximation restricts \( \Sigma_i \) to be a diagonal covariance matrix, or equivalently assumes that the probability distribution can be expressed as a product of individual weight distributions:

\[
\mathcal{N}(\mu_i, \Sigma_i) = \prod_j \mathcal{N}(\mu_{ij}, \sigma_{ij}).
\] (7)

The mean-field approximation greatly reduces the computational cost of both forward and backwards propagation, and reduces the number of parameters required to store the model from order \( n^2 \) in the number of weights to order \( n \). The implementation of mean-field variational inference which we use is based on Blundell et al. [2015], who show how to use a stochastic estimator of the ELBO.

B Discussions of Foong et al. 2020

Our work discusses some similar topics to those discussed by Foong et al. [2020], in work which was developed in parallel to this paper. In particular, they reach a different conclusion as to the value of mean-field variational inference in deep BNNs. We find their work insightful, but we think it is important to be very precise about where their results do and do not apply, in order to best understand their implications. Here, we briefly describe several of their main results, emphasising that our analysis is not in conflict with any of their proofs.

Empirically, they focus on the posterior distribution over function outputs in small models, where they find that the learned function distributions with mean-field variational inference for regression tend to be overconfident, even in slightly deeper models. We also find that in relatively small regression tasks MFVI does not perform particularly well, which aligns with their results. However, below, we note that in fact their theoretical results suggest that MFVI may have problems with regression that do not extend to classification tasks. Moreover, it is important to note that the largest of their experiments considers data with only 16-dimensional inputs, with only 55 training points and very small models with 4 layers of 50 hidden units. In contrast, our work analyses much larger models and datasets, where it is admittedly harder to compare to a reference posterior in function-space. However, these are the situations where MFVI would be more typical for deep learning. We therefore feel that, while there is significant room for further investigation, on the balance of the current evidence MFVI seems quite appropriate for large-scale deep learning especially in classification tasks.

Foong et al. [2020] base much of their interpretation on the inability of single-layer mean-field networks to have appropriate ‘in-between’ uncertainty. That is, they observe that if a model is trained on data in two regions of input space which are separated, it ought to be able to be significantly more uncertain between those two regions. They prove their Theorem 1 which states that the variance of the function expressed by a single-layer mean-field network between two points cannot be greater than the sum of the variances of the function at those points, subject to a number of very important caveats. Our work is largely concerned with models with more than a single hidden layer, where
their theorem does not apply. Nevertheless, Foong et al. [2020] hypothesize the pathologies that they identify might extend to deeper settings, so we note some further limitations of their proof.

1. Theorem 1 applies to single- or multi-output regression models, and to the individual logits in classification models, but makes no predictions about the variance of classification decisions (because this depends on the variance of the argmax of the logits).

2. Theorem 1 is strongest in a 1-dimensional input space. In higher dimensions, Foong et al. [2020] show as a corollary that the ‘in-between’ variance is bounded by the sum of the variances of the hypercube of points including that space. But the number of such points grows exponentially with the dimensionality, meaning that in even only 10 dimensions, the in-between variance could be as much as 1024 times greater than the average edge variance. This means that in high-dimensional input spaces, this bound can be extremely loose and does not necessarily preclude even single layer models from having significant ‘in between’ uncertainty.

3. The line-segments where Theorem 1 applies are not fully general. They must either go through the origin, or be orthogonal to one of the input basis vectors and cross a projection of the origin. This makes their result sensitive to translation and rotation of the input space. However, the authors do provide some empirical evidence that ‘in between’ uncertainty is too low on more general lines in input space for small models.

They also prove their Theorem 3, which establishes that deeper mean-field networks do not have the pathologies that apply in the limited single-layer regression settings identified in Theorem 1. We consider a similar result (Proposition 3), which is more general because it considers more than the first two moments of the distribution. Unlike Foong et al. [2020] we see this result as potentially promising that deep mean-field networks can be very expressive. This is perhaps because their empirical results suggest that deeper mean-field networks have poor performance in practice. This may be because their experiments mostly focus on low-dimensional data with small numbers of datapoints and comparatively small networks, while we consider the larger settings.

We note also that we find that deeper networks are able to show in-between uncertainty in regression, if not necessarily capture it as fully as something like HMC. In Figure 5 we show how increasing depth increases the ability of MFVI neural networks to capture in-between uncertainty even on low-dimensional data.

Here, each layer has 100 hidden units trained using mean-field variational inference on a sythetic dataset in order to demonstrate the possibility of ‘in-between’ uncertainty. Full experimental settings are provided in Table 3. The toy function used is $y = \sin(4(x - 4.3)) + \epsilon$ where $\epsilon \sim N(0, 0.05^2)$. We sample 750 points in the interval $-2 \leq x \leq -1.4$ and another 750 points in the interval $1.0 \leq x \leq 1.8$. We considered a range of temperatures between 0.1 and 100 in order to select the right balance between prior and data. Note of course that while our figure in the main body suffices to demonstrate the existence claim that there are deep networks that perform well, of course a single case of a one-layer network performing badly does not show that all one-layer networks perform badly.

C Experimental Details

C.1 Full Description of Covariance Visualization

Here we provide details on the method used to produce Figure 1. The linear version of the visualization is discussed in §3 and the piecewise-linear version is discussed in §4.

In all cases, we train a neural network using mean-field variational inference in order to visualize the covariance of the product matrix. The details of training are provided in Table 4. The product matrix is calculated from the weight matrices of an $L$-layer network. In the linear case, this is just the matrix product of the $L$ layers. In the piecewise-linear case the definition of the product matrix is described in more detail in Appendix D.4.2. All covariances are calculated using 10,000 samples from the converged approximate posterior. Note that for $L$ weight matrices there are $L - 1$ layers of hidden units.

We compare these learned product matrices, in Figure 6, to a randomly sampled product matrix. To do so, we sample weight layers whose entries are distributed normally. Each weight is sampled with
Figure 5: In-between uncertainty. With a single hidden layer, mean-field variational inference cannot capture in-between uncertainty at all in low-dimensional settings. But by adding more layers, we increase the ability to capture in-between uncertainty, though still not to the point of HMC. Note, of course, that neural networks are at their most useful with higher dimensional data where in-between uncertainty results are exponentially weaker.

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Setting description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>MLP</td>
</tr>
<tr>
<td>Number of hidden layers</td>
<td>3</td>
</tr>
<tr>
<td>Layer Width</td>
<td>100</td>
</tr>
<tr>
<td>Activation</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Approximate Inference Algorithm</td>
<td>Mean-field VI [Wen et al., 2018]</td>
</tr>
<tr>
<td>Optimization algorithm</td>
<td>Amsgrad [Reddi et al., 2018]</td>
</tr>
<tr>
<td>Learning rate</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>Batch size</td>
<td>250</td>
</tr>
<tr>
<td>Variational training samples</td>
<td>1</td>
</tr>
<tr>
<td>Variational test samples</td>
<td>1</td>
</tr>
<tr>
<td>Temperature</td>
<td>65</td>
</tr>
<tr>
<td>Noise scale</td>
<td>0.05</td>
</tr>
<tr>
<td>Epochs</td>
<td>6000</td>
</tr>
<tr>
<td>Variational Posterior Initialization</td>
<td>Tensorflow Probability default</td>
</tr>
<tr>
<td>Prior</td>
<td>$\mathcal{N}(0, 1.0^2)$</td>
</tr>
<tr>
<td>Dataset</td>
<td>Toy (see text)</td>
</tr>
<tr>
<td>Number of training runs</td>
<td>1</td>
</tr>
<tr>
<td>Number of evaluation runs</td>
<td>1</td>
</tr>
<tr>
<td>Measures of central tendency</td>
<td>n.a.</td>
</tr>
<tr>
<td>Runtime per result</td>
<td>$&lt; 5m$</td>
</tr>
<tr>
<td>Computing Infrastructure</td>
<td>Nvidia GTX 1060</td>
</tr>
</tbody>
</table>

Table 3: Experimental Setting—Toy Regression Visualization. Note that for these visualizations we are purely demonstrating the possibility of in-between uncertainty. As a result, a single training/evaluation run suffices to make an existence claim, so we do not do multiple runs in order to calculate a measure of central tendency.
Unlike the covariance matrices of product matrix entries in models trained on real data, a randomly sampled product matrix does not show obvious block structure, though the noise makes it hard to be sure. This model has 5 linear layers.

The local product matrix also allows the unimodal Gaussian layers to approximate multimodal distributions over product matrix entries. Here, we show an example density over a product matrix element, from a three-layer Leaky ReLU model with mean-field Gaussian distributions over each weight trained on FashionMNIST.

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Setting description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>MLP</td>
</tr>
<tr>
<td>Number of hidden layers</td>
<td>0-9</td>
</tr>
<tr>
<td>Layer Width</td>
<td>16</td>
</tr>
<tr>
<td>Activation</td>
<td>Linear or Leaky Relu with $\alpha = 0.1$</td>
</tr>
<tr>
<td>Approximate Inference Algorithm</td>
<td>Mean-field Variational Inference</td>
</tr>
<tr>
<td>Optimization algorithm</td>
<td>Amsgrad [Reddi et al., 2018]</td>
</tr>
<tr>
<td>Learning rate</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>Batch size</td>
<td>64</td>
</tr>
<tr>
<td>Variational training samples</td>
<td>16</td>
</tr>
<tr>
<td>Variational test samples</td>
<td>16</td>
</tr>
<tr>
<td>Epochs</td>
<td>10</td>
</tr>
<tr>
<td>Variational Posterior Initial Mean</td>
<td>He et al. [2016]</td>
</tr>
<tr>
<td>Variational Posterior Initial Standard Deviation</td>
<td>$\log[1 + e^{-3}]$</td>
</tr>
<tr>
<td>Prior</td>
<td>$\mathcal{N}(0, 0.23^2)$</td>
</tr>
<tr>
<td>Dataset</td>
<td>FashionMNIST [Xiao et al., 2017]</td>
</tr>
<tr>
<td>Preprocessing</td>
<td>Data normalized $\mu = 0, \sigma = 1$</td>
</tr>
<tr>
<td>Validation Split</td>
<td>90% train - 10% validation</td>
</tr>
<tr>
<td>Number of training runs</td>
<td>1</td>
</tr>
<tr>
<td>Number of evaluation runs</td>
<td>1</td>
</tr>
<tr>
<td>Measures of central tendency</td>
<td>n.a.</td>
</tr>
<tr>
<td>Runtime per result</td>
<td>&lt; 3m</td>
</tr>
<tr>
<td>Computing Infrastructure</td>
<td>Nvidia GTX 1080</td>
</tr>
</tbody>
</table>

Table 4: Experimental Setting—Covariance Visualization. Note that for these visualizations we are purely demonstrating the possibility of off-diagonal covariance. As a result, a single training/evaluation run suffices to make an existence claim, so we do not do multiple runs in order to calculate a measure of central tendency.
Hyperparameter Setting description

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Setting description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>MLP</td>
</tr>
<tr>
<td>Number of hidden layers</td>
<td>1-4</td>
</tr>
<tr>
<td>Layer Width</td>
<td>4</td>
</tr>
<tr>
<td>Activation</td>
<td>Leaky Relu</td>
</tr>
<tr>
<td>Approximate Inference Algorithm</td>
<td>Variational Inference</td>
</tr>
<tr>
<td>Optimization algorithm</td>
<td>Amsgrad [Reddi et al., 2018]</td>
</tr>
<tr>
<td>Learning rate</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>Batch size</td>
<td>16</td>
</tr>
<tr>
<td>Variational training samples</td>
<td>1</td>
</tr>
<tr>
<td>Variational test samples</td>
<td>1</td>
</tr>
<tr>
<td>Epochs</td>
<td>1000 (early stopping patience=30)</td>
</tr>
<tr>
<td>Variational Posterior Initial Mean</td>
<td>[He et al., 2016]</td>
</tr>
<tr>
<td>Variational Posterior Initial Standard Deviation</td>
<td>$\log[1 + e^{-6}]$</td>
</tr>
<tr>
<td>Prior</td>
<td>$\mathcal{N}(0, 1.0^2)$</td>
</tr>
<tr>
<td>Dataset</td>
<td>Iris [Xiao et al., 2017]</td>
</tr>
<tr>
<td>Preprocessing</td>
<td>None</td>
</tr>
<tr>
<td>Validation Split</td>
<td>100 train - 50 test</td>
</tr>
<tr>
<td>Number of training runs</td>
<td>100</td>
</tr>
<tr>
<td>Number of evaluation runs</td>
<td>100</td>
</tr>
<tr>
<td>Measures of central tendency</td>
<td>(See text.)</td>
</tr>
<tr>
<td>Runtime per result</td>
<td>&lt; 5m</td>
</tr>
<tr>
<td>Computing Infrastructure</td>
<td>Nvidia GTX 1080</td>
</tr>
</tbody>
</table>

Table 5: Experimental Setting—Full Covariance.

standard deviation 0.3 and with a mean 0.01 and each weight matrix is 16x16. This visualization is with a linear product matrix of 5 layers.

Further, since researchers often critique a Gaussian approximate posterior because it is unimodal, we confirm empirically that multiple mean-field layers can induce a multi-modal product matrix distribution. In Figure 7, we show a density over an element of the local product matrix from three layers of weights in a Leaky ReLU BNN with $\alpha = 0.1$. The induced distribution is multi-modal. We visually examined the distributions over 20 randomly chosen entries of this product matrix and found that 12 were multi-modal. We found that without the non-linear activation, none of the product matrix entry distributions examined were multimodal, suggesting that the non-linearities in fact play an important role in inducing rich predictive distributions by creating modes corresponding to activated sign patterns.

C.2 Effect of Depth Measured on Iris Experimental Settings

We describe the full- and diagonal-covariance experiment settings in Table 5. We use a very small model on a small dataset because full-covariance variational inference is unstable, requiring a matrix inversion of a $K^4$ matrix for hidden unit width $K$. Unfortunately, for deeper models the initializations still resulted in failed training for some seeds. To avoid this issue, we selected the 10 best seeds out of 100 training runs, and report the mean and standard error for these. Because we treat full- and diagonal-covariance in the same way, the resulting graph is a fair reflection of their relative best-case merits, but not intended as anything resembling a ‘real-world’ performance benchmark.

Readers may consider the Iris dataset to be unhelpfully small, however this was a necessary choice. We note that the small number of training points creates a broad posterior, which is the best-case scenario for a full-covariance approximate posterior.

C.3 HMC Experimental Settings

We begin by sampling from the true posterior using HMC.
Table 6: HMC samples for ReLU networks are most accurate for smaller numbers of layers, the samples from deeper models may therefore be slightly less reliable. Acceptance rates tend to be with 10-20 percentage points of 65%, regarded as a good balance of exploration to avoiding unnecessary resampling. The more linear models are less accurate, as one would expect for a dataset that is not linearly separable.

![Figure 8](image-url) Figure 8: Example density for randomly chosen parameter from a ReLU network with three hidden layers. The HMC histogram is multimodal. If we picked the naive Gaussian fit, we would lie between the modes. By using a mixture model, we select the dominant mode, for which the Gaussian is a better fit.

![Figure 9](image-url) Figure 9: CIFAR-100. Accuracy for diagonal and low-rank covariance SWAG. Like log-likelihood, there is no clear difference in performance between these models, all of which are above the depth threshold implied by our work.

We use the simple two-dimensional binary classification ‘make moons’ task. We use 500 training points (generated using random_state = 0). Using Cobb et al. [2019], we apply the No-U-turn Sampling scheme [Hoffman and Gelman, 2014] with an initial step size of 0.01. We use a burn-in phase with 10,000 steps targeting a rejection rate of 0.8. We then sample until we collect 1,000 samples from the true posterior, taking 100 leapfrog steps in between every sample used in order to ensure samples are less correlated. For each result, we recalculate the HMC samples 20 times with a different random seed. All chains are initialized at the optimum of a mean-field variational inference model in order to help HMC rapidly find a mode of the true posterior. We use a prior precision, normalizing constant, and $\tau$ of 1.0. The model is designed to have as close to 1000 non-bias parameters each time as possible, adjusting the width given the depth of the model. We observe that the accuracies for the ReLU network fall for the deeper models, suggesting that after about 7 layers the posterior estimate may become slightly less reliable (see Table 6). Acceptance rates are broadly in a sensible region for most of the chains.

---

\[https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html#sklearn.datasets.make_moons\]
<table>
<thead>
<tr>
<th>Architecture</th>
<th>Method</th>
<th>Covariance</th>
<th>Accuracy</th>
<th>NLL</th>
<th>ECE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18</td>
<td>VOGN(\dagger)</td>
<td>Diagonal</td>
<td>67.4% ± 0.263</td>
<td>1.37 ± 0.010</td>
<td>0.029 ± 0.001</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>Noisy K-FAC(\dagger)†</td>
<td>MVG</td>
<td>66.4% ± n.d.</td>
<td>1.44 ± n.d.</td>
<td>0.080 ± n.d.</td>
</tr>
<tr>
<td>DenseNet-161</td>
<td>SWAG-Diag(\dagger)</td>
<td>Diagonal</td>
<td>78.6% ± 0.000</td>
<td>0.86 ± 0.000</td>
<td>0.046 ± 0.000</td>
</tr>
<tr>
<td>DenseNet-161</td>
<td>SWAG(\dagger)</td>
<td>Low-rank + Diag</td>
<td>78.6% ± 0.000</td>
<td>0.83 ± 0.000</td>
<td>0.020 ± 0.000</td>
</tr>
<tr>
<td>ResNet-152</td>
<td>SWAG-Diag(\dagger)</td>
<td>Diagonal</td>
<td>80.0% ± 0.000</td>
<td>0.86 ± 0.000</td>
<td>0.057 ± 0.000</td>
</tr>
<tr>
<td>ResNet-152</td>
<td>SWAG(\dagger)</td>
<td>Low-rank + Diag</td>
<td>79.1% ± 0.000</td>
<td>0.82 ± 0.000</td>
<td>0.028 ± 0.000</td>
</tr>
</tbody>
</table>

Table 7: Imagenet. Comparison of diagonal-covariance/mean-field (in grey) and structured-covariance methods on Imagenet. The differences on a given architecture between comparable methods is slight. \(\dagger\) [Maddox et al., 2019]. † [Osawa et al., 2019]. †† [Zhang et al., 2018] as reported by Osawa et al. [2019].

Using these samples, we find a Gaussian fit. For each model we fit a Gaussian mixture model with between 1 and 4 components and pick the one with the best Bayesian information criterion (see Figure 8). We then find the best diagonal fit to this distribution, which is a Gaussian distribution with the same mean and with a precision matrix equal to the inverse of the diagonal precision of the full-covariance Gaussian. Note that we are therefore only considering one of the many modes of the true posterior—this is inevitable given the fact that there is a many-to-one correspondence between weight-distributions and function-distributions for neural networks.

Finally, we calculate the KL divergence between these two distributions. The graph reports the mean and shading reflects the standard error of the mean, though note that because all runs are initialized from the same point, this underestimates the overall standard error. All experiments in this and other sections were run on a desktop workstation with an Nvidia 1080 GPU.

For the Wasserstein divergence, we estimate the distance between the empirical distributions formed by the HMC samples and samples from the full- and diagonal-covariance posterior approximation. We used the Python Optimal Transport package to estimate the divergence.

C.4 Diagonal- and Structured-SWAG at Varying Depths

We use the implementation of SWAG available publicly at [https://github.com/wjmaddox/swa-gaussian](https://github.com/wjmaddox/swa-gaussian). We adapt their code to vary the depth of the PreResNet architecture for the values 2, 8, 14, 20, 26, 32, 38. We use the hyperparameter settings used by the Maddox et al. [2019] for PreResNet154 on CIFAR100 on all datasets to train the models. We use 10 seeds to generate the error bars, which are plotted with one standard deviation. We use the same SWAG run to fit both the full and diagonal approximations, and use 30 samples in the forward pass.

C.5 Large Scale Experiments Descriptions

In Table 7 we show a complete version of Table 2 including the standard deviations over three runs (except for the Noisy K-FAC result where standard deviation was not provided). The standard deviations, of course, underestimate the true variability of the method in question on Imagenet as they only consider difference in random seed with the training configuration otherwise identical. Fuller descriptions of the experimental settings used by the authors are provided in the cited papers.

For CIFAR-10, we show similar results in Table 8. Here, authors compare a wider range of architectures, which show substantial variation in resulting accuracy. However, within the same architecture, there is little evidence of systematic differences between mean-field and structured-covariance methods and any differences which do appear are marginal. Note that Zhang et al. [2018] report difficulty applying batch normalization to mean-field methods, but Osawa et al. [2019] report no difficulties applying batch normalization for their mean-field variant of Noisy Adam. For this reason, we report the version of Noisy KFAC run without batch normalization to make it comparable with the results shown for Bayes-by-Backprop (BBB) and Noisy Adam. With batch normalization, Noisy KFAC gains some accuracy, reaching 92.0%, but this seems to be because of the additional regularization, not a property of the approximate posterior family.
<table>
<thead>
<tr>
<th>Architecture</th>
<th>Method</th>
<th>Covariance</th>
<th>Accuracy</th>
<th>NLL</th>
<th>ECE</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG-16</td>
<td>SWAG-Diag†</td>
<td>Diagonal</td>
<td>93.7% ± 0.15</td>
<td>0.220 ± 0.008</td>
<td>0.027 ± 0.003</td>
</tr>
<tr>
<td>VGG-16</td>
<td>SWAG†</td>
<td>Low-rank + Diag</td>
<td>93.6% ± 0.10</td>
<td>0.202 ± 0.003</td>
<td>0.016 ± 0.003</td>
</tr>
<tr>
<td>VGG-16</td>
<td>Noisy Adam††</td>
<td>Diagonal</td>
<td>88.2% ± n.d.</td>
<td>n.d.</td>
<td>n.d.</td>
</tr>
<tr>
<td>VGG-16</td>
<td>BBB‡‡</td>
<td>Diagonal</td>
<td>88.3% ± n.d.</td>
<td>n.d.</td>
<td>n.d.</td>
</tr>
<tr>
<td>VGG-16</td>
<td>Noisy KFAC‡‡</td>
<td>MVG</td>
<td>89.4% ± n.d.</td>
<td>n.d.</td>
<td>n.d.</td>
</tr>
<tr>
<td>PreResNet-164</td>
<td>SWAG-Diag†</td>
<td>Diagonal</td>
<td>96.0% ± 0.10</td>
<td>0.125 ± 0.003</td>
<td>0.008 ± 0.001</td>
</tr>
<tr>
<td>PreResNet-164</td>
<td>SWAG†</td>
<td>Low-rank + Diag</td>
<td>96.0% ± 0.02</td>
<td>0.123 ± 0.002</td>
<td>0.005 ± 0.000</td>
</tr>
<tr>
<td>WideResNet28x10</td>
<td>SWAG-Diag†</td>
<td>Diagonal</td>
<td>96.4% ± 0.08</td>
<td>0.108 ± 0.001</td>
<td>0.005 ± 0.001</td>
</tr>
<tr>
<td>WideResNet28x10</td>
<td>SWAG†</td>
<td>Low-rank + Diag</td>
<td>96.3% ± 0.08</td>
<td>0.112 ± 0.001</td>
<td>0.009 ± 0.001</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>VOGN‡</td>
<td>Diagonal</td>
<td>84.3% ± 0.20</td>
<td>0.477 ± 0.006</td>
<td>0.040 ± 0.002</td>
</tr>
<tr>
<td>AlexNet</td>
<td>VOGN‡</td>
<td>Diagonal</td>
<td>75.5% ± 0.48</td>
<td>0.703 ± 0.006</td>
<td>0.016 ± 0.001</td>
</tr>
</tbody>
</table>

Table 8: CIFAR-10. For a given architecture, it does not seem that mean-field (grey) methods systematically perform worse than methods with structured covariance, although there is some difference in the results reported by different authors. † † [Maddox et al., 2019]. † † [Osawa et al., 2019].

D Proof

D.1 Full Derivation of the Product Matrix Covariance

**Proposition 1.** For \( L \geq 3 \), the product matrix \( M^{(L)} \) of factorized weight matrices can have non-zero covariance between any and all pairs of elements. That is, there exists a set of mean-field weight matrices \( \{W^{(l)}\}_{1 \leq l \leq L} \) such that \( M^{(L)} = \prod W^{(l)} \) and the covariance between any possible pair of elements of the product matrix:

\[
\text{Cov}(m_{ab}^{(L)}, m_{cd}^{(L)}) \neq 0,
\]

where \( m_{ij}^{(L)} \) are elements of the product matrix in the \( i \)th row and \( j \)th column, and for any possible indexes \( a, b, c, \) and \( d \).

**Proof.** We begin by explicitly deriving the covariance between elements of the product matrix.

Consider the product matrix, \( M^{(L)} \), which is the matrix product of an arbitrary weight matrix, \( W^{(L)} \), with a mean field distribution over it’s entries, and the product matrix with one fewer layers, \( M^{(L-1)} \). Expressed in terms of the elements of each matrix in row-column notation this matrix multiplication can be written:

\[
m_{ab}^{(L)} = \sum_{i=1}^{K_{L-1}} w_{ai}^{(L)} m_{ib}^{(L-1)}. \tag{8}
\]

We make no assumption about \( K_{L-1} \) except that it is non-zero and hence the weights can be any rectangular matrix.\(^3\) The weight matrix \( W^{(L)} \) is assumed to have a mean-field distribution (the covariance matrix is zero for all off diagonal elements) with arbitrary means:

\[
\text{Cov}(w_{ac}^{(L)}, w_{bd}^{(L)}) = \mu_{ac}^{(L)} \mu_{bd}^{(L)}; \\
\mathbb{E} w_{ab}^{(L)} = \mu_{ab}^{(L)}. \tag{9}
\]

\( \delta \) are the Kronecker delta. Note that the weight matrix is 2-dimensional, but the covariance matrix is defined between every element of the weight matrix. While it can be helpful to regard it as 2-dimensional also, we index it with the four indices that define a pair of elements of the weight matrix.

We begin by deriving the expression for the covariance of the \( L \)-layer product matrix \( \text{Cov}(m_{ab}^{(L)}, m_{cd}^{(L)}) \). Using the definition of the product matrix in equation (8):

\[
\text{Cov}(m_{ab}^{(L)}, m_{cd}^{(L)}) = \text{Cov}\left( \sum_{i} w_{ai}^{(L)} m_{ib}^{(L-1)} \right, \sum_{j} w_{cj}^{(L)} m_{jd}^{(L-1)}). \tag{10}
\]

\(^3\) We set aside bias parameters, as they complicate the algebra, but adding them only strengthens the result because each bias term affects an entire row.
We then simplify this using the linearity of covariance (for brevity call the covariance of the product matrix \( \Sigma_{abcd} \)):

\[
\Sigma_{abcd} = \sum_{ij} \text{Cov}(w_{ai}^{(L)} m_{ib}^{(L-1)}, w_{cj}^{(L)} m_{jd}^{(L-1)}),
\]

rewriting using the definition of covariance in terms of a difference of expectations:

\[
= \sum_{ij} \mathbb{E}[w_{ai}^{(L)} m_{ib}^{(L-1)} w_{cj}^{(L)} m_{jd}^{(L-1)}] - \mathbb{E}[w_{ai}^{(L)} m_{ib}^{(L-1)}] \mathbb{E}[w_{cj}^{(L)} m_{jd}^{(L-1)}],
\]

using the fact that by assumption the new layer is independent of the previous product matrix:

\[
= \sum_{ij} \mathbb{E}[w_{ai}^{(L)} w_{cj}^{(L)}] \mathbb{E}[m_{ib}^{(L-1)} m_{jd}^{(L-1)}] - \mathbb{E}[w_{ai}^{(L)}] \mathbb{E}[w_{cj}^{(L)}] \mathbb{E}[m_{ib}^{(L-1)}] \mathbb{E}[m_{jd}^{(L-1)}],
\]

and rewriting to expose the dependence on the covariance of \( M^{(L-1)} \):

\[
= \sum_{ij} \left( \mathbb{E}[w_{ai}^{(L)} w_{cj}^{(L)}] - \mathbb{E}[w_{ai}^{(L)}] \mathbb{E}[w_{cj}^{(L)}] \right)
\cdot \left( \mathbb{E}[m_{ib}^{(L-1)} m_{jd}^{(L-1)}] - \mathbb{E}[m_{ib}^{(L-1)}] \mathbb{E}[m_{jd}^{(L-1)}] \right)
+ \mathbb{E}[w_{ai}^{(L)}] \mathbb{E}[w_{cj}^{(L)}] \left( \mathbb{E}[m_{ib}^{(L-1)} m_{jd}^{(L-1)}] - \mathbb{E}[m_{ib}^{(L-1)}] \mathbb{E}[m_{jd}^{(L-1)}] \right)
+ \mathbb{E}[m_{ib}^{(L-1)}] \mathbb{E}[m_{jd}^{(L-1)}] \left( \mathbb{E}[w_{ai}^{(L)} w_{cj}^{(L)}] - \mathbb{E}[w_{ai}^{(L)}] \mathbb{E}[w_{cj}^{(L)}] \right),
\]

substituting the covariance:

\[
= \sum_{ij} \text{Cov}(w_{ai}^{(L)}, w_{cj}^{(L)}) \cdot \text{Cov}(m_{ib}^{(L-1)}, m_{jd}^{(L-1)})
+ \mathbb{E}[w_{ai}^{(L)}] \mathbb{E}[w_{cj}^{(L)}] \text{Cov}(m_{ib}^{(L-1)}, m_{jd}^{(L-1)})
+ \mathbb{E}[m_{ib}^{(L-1)}] \mathbb{E}[m_{jd}^{(L-1)}] \text{Cov}(w_{ai}^{(L)}, w_{cj}^{(L)}).
\]

This gives us a recursive expression for the covariance of the product matrix.

It is straightforward to substitute in our expressions for mean and variance in a mean-field network provided in equation (9), where we use the fact that the initial \( M^{(1)} \) product matrix is just a single mean-field layer.

In this way, we show that:

\[
\Sigma_{abcd}^{(2)} = \sum_{ij} \left( \delta_{ac} \delta_{ij} \delta_{ai}^{(2)} \cdot (\delta_{ij} \delta_{bd} \sigma_{ib}^{(1)}) \right)
+ \mu_{ai}^{(2)} \mu_{cj}^{(2)} \delta_{ij} \delta_{bd} \sigma_{ib}^{(1)} + \mathbb{E}[m_{ib}^{(1)}] \mathbb{E}[m_{jd}^{(1)}] \left( \delta_{ac} \delta_{ij} \delta_{ai}^{(2)} \right)
\]

\[
= \sum_{ij} \delta_{ac} \delta_{bd} \sigma_{ai}^{(2)} \sigma_{ib}^{(1)} + \delta_{bd} \mu_{ai}^{(2)} \mu_{cj}^{(2)} \sigma_{ib}^{(1)} + \delta_{ac} \mu_{ib} \mu_{id}^{(1)} \sigma_{ai}^{(2)}. 
\]

The first term of equation (17) has the Kronecker deltas \( \delta_{ac} \delta_{bd} \) meaning that it contains diagonal entries in the covariance matrix. The second term has only \( \delta_{bd} \) meaning it contains entries for the covariance between weights that share a column. The third term has only \( \delta_{ac} \) meaning it contains entries for the covariance between weights that share a row.

This covariance of the product matrix already has some off-diagonal terms, but it does not yet contain non-zero covariance for weights that share neither a row nor a column.
We can gain insight into the richness of the possible covariances by considering the limited case of
the distribution over elements of the product matrix. In particular, for $M(L) = ABC$ such that the elements of $M(L)$ have non-zero covariances everywhere therefore entails that there is a weight matrix $W(L)$ that the covariances which can be induced with

propagate through the layers. Proposition 2. The Matrix Variate Gaussian (Kronecker-factored) distribution is a special case of the distribution over elements of the product matrix. In particular, for $M(3) = ABC$, $M(3)$ is distributed as an MVG random variable when $A$ and $C$ are deterministic and $B$ has its elements distributed as fully factorized Gaussians with unit variance.

Proof. Consider the product matrix $M(3) = ABC$, where $B$ is a matrix whose elements are independent Gaussian random variables and $A$ and $C$ are deterministic. The elements of $B$ are distributed with mean $\mu_B$ and have a diagonal covariance matrix $\Sigma_B$.

We begin by recalling the property of the Kronecker product that:

$$\text{vec}(ABC) = (C^T \otimes A)\text{vec}(B).$$

D.2 Matrix Variate Gaussian as a Special Case of Three-Layer Product Matrix

We can gain insight into the richness of the possible covariances by considering the limited case of

the distribution over elements of the product matrix. In particular, for $M(3) = ABC$, $M(3)$ is distributed as an MVG random variable when $A$ and $C$ are deterministic and $B$ has its elements distributed as fully factorized Gaussians with unit variance.
By definition $\text{vec}(M^{(3)}) = \text{vec}(ABC) = (C^T \otimes A)\text{vec}(B)$. Because $C^T \otimes A$ is deterministic, it follows from a basic property of the covariance that the covariance of the product matrix $\Sigma_{M^{(3)}}$ is given by:

$$\Sigma_{M^{(3)}} = (C^T \otimes A)\Sigma_B (C^T \otimes A)^T. \quad (21)$$

Using the fact that the transpose is distributive over the Kronecker product, this is equivalent to:

$$\Sigma_{M^{(3)}} = (C^T \otimes A)\Sigma_B (C \otimes A^T). \quad (22)$$

Because we only want to establish that the family of distributions expressible contains the matrix variate Gaussians, we do not need to use all the possible freedom, and we can set $\Sigma_B = I$. In this special case:

$$\Sigma_{M^{(3)}} = (C^T \otimes A)(C \otimes A^T). \quad (23)$$

Using the mixed-product property, this is equivalent to:

$$\Sigma_{M^{(3)}} = (C^T C) \otimes (AA^T). \quad (24)$$

Now, we note that any positive semi-definite matrix can be written in the form $A = M^T M$, so this implies that, defining the positive semi-definite matrices $V = C^T C$ and $U = AA^T$, we have that the covariance $\Sigma_{M^{(3)}}$ is of the form,

$$\Sigma_{M^{(3)}} = V \otimes U. \quad (25)$$

Similarly, we can consider the mean of the product matrix $\mu_{M^{(3)}}$. From equation $20$, we can see that:

$$\mu_{M^{(3)}} = (C^T \otimes A)\text{vec}(\mu_B). \quad (26)$$

But since we have not yet constrained $\mu_B$, it is clear that this allows us to set any $\mu_{M^{(3)}}$ we desire by choosing $\mu_B = (C^T \otimes A)^{-1} \mu_{M^{(3)}}$.

So far, we have only discussed the first- and second-moments, and the proof has made no assumptions about specific distributions. However, we now observe that a random variable $X$ is distributed according to the Matrix Variate Gaussian distribution according to some mean $\mu_X$ and with scale matrices $U$ and $V$ if and only if $\text{vec}(X)$ is a multivariate Gaussian with mean $\vec{\mu}_X$ and covariance $U \otimes V$.

Therefore, given equations $26$ and $25$, the special case of $M^{(3)}$ where the first and last matrices are deterministic and the middle layer has a fully-factorized Gaussian distribution over the weights with unit variance is a Matrix Variate Gaussian distribution where:

$$\text{vec}(\mu_X) = (C^T \otimes A)\text{vec}(\mu_B); \quad (27)$$

$$V = C^T C; \quad (28)$$

$$U = A^T A. \quad (29)$$

D.3 Distribution of the Product Matrix

In general the probability density function of a product of random variables is not the product of their density functions. In the scalar case, the product of two independent Gaussian distributions is a generalized $\chi^2$ distribution. The product of arbitrarily many Gaussians with arbitrary non-i.i.d. mean and variance is difficult to calculate (special cases are much better understood e.g., Springer and Thompson [1970]). An example of a distribution family that is closed under multiplication is the log-normal distribution.

In the case of matrix multiplication, important for neural network weights, because each element of a product of matrix multiplication is the sum of the product of individual elements we would ideally like a distribution to be closed under both addition and multiplication (such as the Generalized Gamma convolution [Bondesson, 2015]) but these are not practical.
Instead, it would be helpful if we could make use of a simple distribution like the Gaussian but maintain roughly similar distributions over product matrix elements as the network becomes deeper. For only one layer of hidden units, provided $K$ is sufficiently large, we can use the central limit theorem to show that the elements of the product matrix composed of i.i.d. Gaussian priors tends to a Gaussian as the width of the hidden layer increases. For two or more layers, however, the central limit theorem fails because the elements of the product matrix are no longer independent. However, even though the resulting product matrix is not a Gaussian, we show through numerical simulation that products of matrices with individual weights distributed as $\mathcal{N}(0, 0.23^2)$ have roughly the same distribution over their weights. This, combined with the fact that our choice of Gaussian distributions over weights was somewhat arbitrary in the first place, might reassure us that the increase in depth does not change the model prior in an important way. In Figure 10 we plot the probability density function of an arbitrarily chosen entry in the product matrix with varying depths of diagonal Gaussian prior weights. The p.d.f. for 7 layers is approximately the same as the single-layer Gaussian distribution with variance $0.23^2$.

D.4 Proof of Linearized Product Matrix Covariance

D.4.1 Proof of Local Linearity

We consider local linearity in the case of piecewise-linear activations like ReLU.

**Lemma 1.** Consider an input point $\mathbf{x}^* \in \mathcal{D}$. Consider a realization of the model weights $\theta$. Then, for any $\mathbf{x}^*$, the neural network function $f_\theta$ is linear over some compact set $\mathcal{A}_\theta \subseteq \mathcal{D}$ containing $\mathbf{x}^*$. Moreover, $\mathcal{A}_\theta$ has non-zero measure for almost all $\mathbf{x}^*$ w.r.t. the Lebesgue measure.

**Proof.** Neural networks with finitely many piecewise-linear activations are themselves piecewise-linear. Therefore, for a finite neural network, we can decompose the input domain $\mathcal{D}$ into regions $\mathcal{D}_i \subseteq \mathcal{D}$ such that

1. $\cup \mathcal{D}_i = \mathcal{D}$,
2. $\mathcal{D}_i \cap \mathcal{D}_j = \emptyset \quad \forall i \neq j$,
3. $f_\theta$ is a linear function on points in $\mathcal{D}_i$ for each $i$.

For a finite neural network, there are at most finitely many regions $\mathcal{D}_i$. In particular, with hidden layer widths $n_i$ in the $i$th layer, with an input domain $\mathcal{D}$ with dimension $n_0$, [Montúfar et al.] show that the network can define maximally a number of regions in input space bounded above by:

$$
\left( \prod_{i=1}^{L-1} \frac{n_i}{n_0} \right)^{n_0} \sum_{j=0}^{n_L} \binom{n_L}{j}.
$$

(30)

Except in the trivial case where the input domain has measure zero, this along with (1) and (2) jointly entail that at least one of the regions $\mathcal{D}_i$ has non-zero measure. This, with (3) entails that only a set of input points of zero measure do not fall in a linear region of non-zero measure. These points correspond to inputs that lie directly on the inflection points of the ReLU activations.

We visualize $\mathcal{A}_\theta$ in Figure 11a. This shows a two-dimensional input space (from the two moons dataset). Parts of the space within which a neural network function is linear are shown in one color. The regions are typically smallest where the most detail is required in the trained function.
Figure 11: Visualization of the linear regions in input-space for a two-dimensional binary classification problem (two moons). Colored regions show contiguous areas within which a neural network function is linear. We use an arbitrary numerical encoding of these regions (we interpret the sign pattern of activated relus as an integer in base 2) and a cyclic colour scheme for visualisation, so the color of each region is arbitrary, and two non-contiguous regions with the same color are not the same region. The neural network has one hidden layer with 100 units and is trained for 1000 epochs on 500 datapoints from scipy’s two moons using Adam. (a) a single model has fairly large linear regions, with the most detail clustered near the region of interest. (b) The regions within which all samples are linear (the intersection set $A$) are smaller, but finite. The local product matrix is valid within one of these regions for any input point.

D.4.2 Defining the Local Product Matrix

We define a random variate representing the local product matrix, for an input point $x^*$, using the following procedure.

To draw a finite $N$ samples of the random variate, we sample $N$ realizations of the weight parameters $\Theta = \{\theta_i\}$ for $1 \leq i \leq N$. For each $\theta_i$, given $x^*$ there is a compact set $A_{\theta_i} \subset D$ within which $f_{\theta_i}$ is linear (and $x^* \in A_{\theta_i}$) by lemma 1. Therefore, all samples of the neural network function are linear in the intersection region $A = \bigcap_i A_{\theta_i}$. We note that $A$ at least contains $x^*$.

For each $\theta_i$ we can compute a local product matrix within $A$. Ordinarily, setting aside the bias term for simplicity, a neural network hidden layer $h_{l+1}$ can be written in terms of the hidden layer before

$$h_{l+1} = \mathbf{W}_{\theta_i} h_l + \mathbf{b}_{\theta_i}$$

Intuitively, we know that $x^*$ is in all $A_{\theta_i}$, so when we add a new sample we know that there is either overlap around $x^*$ or the point $x^*$ is on the boundary of the new subset, which means we could equally well pick a different set that has $x^*$ on its boundary and does have non-zero-measure overlap with the previous sets.

More formally, consider some compact set $A_{\theta_0} \subset D$ with non-zero measure such that $x^* \in A_{\theta_0}$. Take some new compact set $A_{\theta_1} \subset D$ with non-zero measure also such that $x^* \in A_{\theta_1}$. Define the intersection between those sets $B = A_{\theta_0} \cap A_{\theta_1}$. Suppose that $B$ has zero measure. But both $A_{\theta_0}$ and $A_{\theta_1}$ contain $x^*$, so the only way that $B$ could have zero measure is if $x^*$ is an element in the boundary of both sets. But if $A_{\theta_1}$ has $x^*$ on its boundary, then, by the continuity of the real space, there is at least one other compact set $A'_{\theta_1}$, different to $A_{\theta_1}$, such that $x^*$ is on its boundary. But, since by hypothesis $A_{\theta_0}$ has non-zero measure, there exists such a set $A'_{\theta_0}$ which has a non-zero-measure intersection with $A_{\theta_0}$. We can therefore select $A'_{\theta_0}$ instead of $A_{\theta_0}$ when building $A$, such that the intersection with $A_{\theta_1}$ has non-zero measure. By repeated application of this argument, we can guarantee that for any finite $\Theta$ we are able to find a set of $A_{\theta_i} \subset D$ such that $\forall i : x^* \in A_{\theta_i}$ and $A$ has non-zero measure. This argument does not guarantee that the measure of $A$ in the limit as $N$ tends to infinity is non-zero.
it, a weight matrix $W_l$, and an activation function.

$$h_{t+1} = \sigma(W_l h_t)$$

We observe that within $\mathcal{A}$ the activation function becomes linear. This allows us to define an activation vector $a_{x^*}$ within $\mathcal{A}$ such that the equation can be written:

$$h_{t+1} = a_{x^*} \cdot (W_l h_t).$$

The activation vector can be easily calculated by calculating $W_l h_t$, seeing which side of the (Leaky) ReLU the activation is on within that linear region for each hidden unit, and selecting the correct scalar (0 or 1 for a ReLU, or $\alpha$ or 1 for a Leaky ReLU).

This allows us to straightforwardly construct a product matrix for each $\theta$ which takes the activation function into account (in the linear case, we effectively always set $a_{x^*}$ to equal the unit vector).

The random variate $P_{x^*}$ is constructed with these product matrices for realizations of the weight distribution.

Samples from the resulting random variate $P_{x^*}$ are therefore distributed such that samples from $P_{x^*}x^*$ have the same distribution as samples of the predictive posterior $y$ given $x^*$ within $\mathcal{A}$.

**D.4.3 Proof that the Local Product Matrix has Non-zero Off-diagonal Covariance**

**Proposition 3.** Given a mean-field distribution over the weights of neural network $f$ with piecewise linear activations, $f$ can be written in terms of the local product matrix $P_{x^*}$ within $\mathcal{A}$.

For $L \geq 3$, for activation functions which are non-zero everywhere, there exists a set of weight matrices $\{W^{(l)}|1 \leq l < L\}$ such that all elements of the local product matrix have non-zero off-diagonal covariance:

$$\text{Cov}(p^{x^*}_{ij}, p^{x^*}_{cd}) \neq 0,$$

where $p^{x^*}_{ij}$ is the element at the $i$th row and $j$th column of $P_{x^*}$.

**Proof.** First, we show that the covariance between arbitrary entries of each realization of the product matrix of linearized functions can be non-zero. Afterwards, we will show that this implies that the covariance between arbitrary entries of the product matrix random variate, $P_{x^*}$ can be non-zero.

Consider a local product matrix constructed as above. Then for each realization of the weight matrices, the product matrix realization $M^{(3)}_i$, defined in the region around $x^*$ following lemma 1. We can derive the covariance between elements of this product matrix within that region in the same way as in Proposition 1, finding similarly that:

$$\hat{\Sigma}^{(3)}_{abcd} = \alpha_{abcd} \sum_{ij} \delta_{ac} \delta_{ij} \sigma^{(3)}_{ai} \cdot \sum_k \left( \delta_{ij} \delta_{bd} \sigma^{(2)}_{ik} \sigma^{(1)}_{kb} + \delta_{bd} \mu^{(2)}_{ik} \mu^{(1)}_{jk} \sigma^{(1)}_{kb} + \delta_{ij} \mu^{(1)}_{bk} \mu^{(2)}_{kd} \sigma^{(2)}_{ik} \right) + \mu^{(3)}_{ai} \mu^{(3)}_{cj} \sum_k \left( \delta_{ij} \delta_{bd} \sigma^{(2)}_{ik} \sigma^{(1)}_{kb} + \delta_{bd} \mu^{(2)}_{ik} \mu^{(2)}_{jk} \sigma^{(1)}_{kb} + \delta_{ij} \mu^{(1)}_{bk} \mu^{(1)}_{kd} \sigma^{(2)}_{ik} \right) + \mu^{(2)}_{ib} \mu^{(2)}_{jd} \left( \delta_{ac} \delta_{ij} \sigma^{(3)}_{ai} \right),$$

where $\alpha$ is a constant determined by the piecewise-linearity in the linear region we are considering. Note that we must assume here that $\alpha_{ij} \neq 0$ except in a region of zero measure, for example a
LeakyReLU, otherwise it is possible that the constant introduced by the activation could eliminate non-zero covariances. We discuss this point further below.

Now note that the covariance of the sum of independent random variables is the sum of their covariances. Therefore the covariance of $I$ realizations of $P$ (suppressing the notation $(L)$) is:

$$\text{Cov}(P_{ab}, P_{cd}) = \frac{1}{I} \sum_{i=1}^{I} \hat{\Sigma}_{abcd}$$ (34)

As before, consider the case of positive means and standard deviations. Just like before, this results in a positive entry in the covariance between any two elements for each realization of the product matrix, and by equation 34 the entry for any element of the local product matrix remains positive as well. This suffices to prove the proposition for the case of $L = 3$. Just as Proposition 1 extends to all larger $L$, this result does also.

**Remark 4.** The above proof assumes that $\alpha_{ij} \neq 0$. In fact, for common piecewise non-linearities like ReLU, $\alpha$ may indeed be zero. This means that the non-linearity can, in principle, ‘disconnect’ regions of the network such that the ‘effective depth’ falls below 3 and there is not a covariance between every element.

We cannot rule this out theoretically, as it depends on the data and learned function. In practice, we find it very unlikely that a trained neural network will turn off all its activations for any typical input, nor that enough activations will be zero that the product matrix does not have shared elements after some depth.

However, we do observe that for at least some network structures and datasets it is uncommon in practice that all the activations in several layers are ‘switched off’. We show in Figure 2 an example of a local product matrix covariance which does not suffer from this problem. We find that for a model trained with mean-field VI on the FashionMNIST test dataset the number of activations switched on is on average 48.5% with standard deviation 4.7%. There were only four sampled models out of 100 samples on each of 10,000 test points where an entire row of activations was ‘switched off’, reducing the effective depth by one, and this never occurred in more than one row. Indeed, [Goldblum et al., 2019] describe settings with all activations switched off as a pathological case where SGD fails.

**D.5 Existence Proof of a Two-Hidden-Layer Mean-field Approximate Posterior Inducing the True Posterior Predictive**

In this section we prove that:

**Proposition 4.** Let $p(y|x, D)$ be some fixed posterior predictive distribution for univariate $y$. Let $\epsilon > 0$ and $\delta > 0$. Then, there exists a sufficiently wide BNN with two hidden layers and for which there exists a mean-field Gaussian approximate posterior distribution $q(\theta)$ over its weights such that the probability density function over the outputs $p_{\theta}(\hat{y}|x)$ approximates the posterior predictive probability density function in that:

$$\forall x : \Pr(|p_{\theta}(\hat{y}|x) - p(y|x, D)| > \epsilon) < \delta,$$

provided that the cumulative density function of the posterior predictive is monotonically increasing and continuous.

**Proof.** We extend an informal construction by [Gal, 2016] which aimed to show that a sufficiently deep network with a unimodal approximate posterior could induce a multi-modal posterior predictive by learning the inverse cumulative distribution function (c.d.f.) of the multi-modal distribution. We adapt this using the Universal Approximation Theorem (UAT) due to [Leshno et al., 1993] in order to approximate the inverse c.d.f. arbitrarily closely. In order to apply the proof in the context of Bayesian neural networks (BNNs) we apply an extension to the UAT by [Foong et al., 2020]. Our proposition is similar to Theorem 3 by [Foong et al., 2020], but they consider only the first two moments of the predictive function distribution. Like [Leshno et al., 1993] we offer a proof for a single-dimensional output, noting that the extension to multiple dimensions follows from the existence of a mapping from $\mathbb{R} \rightarrow \mathbb{R}^K$.

We first give some intuition as to how the proof works. In outline:
Step 1: We note that a two-hidden-layer Bayesian neural network can be thought of in two parts. The first weight layer—the random variable introducer—is the first weight layer and elementwise nonlinearity. The second part—the random variable mapper—is the rest of the network, and is itself a neural network with a single hidden layer.

Step 2: We show that the random variable introducer can ‘copy’ forward the features concatenated with a random variable with known distribution.

Step 3: We show that there exists a transformation mapping samples from this random variable onto samples from the ‘target’—the true posterior distribution over outputs from the network conditioned on the input.

Step 4: By the probabilistic adaptation of the Universal Approximation Theorem, we show that the random variable mapper can approximate this mapping arbitrarily closely with arbitrary probability.

Step 5: We therefore show that the neural network outputs have a probability density function that is at most boundedly different from the probability density function of the true posterior over outputs from the network.

Step 1: Consider inputs $x \in \mathbb{R}^D$. We define a two-hidden-layer neural network with an invertible non-polynomial activation function $\phi : \mathbb{R} \to \mathbb{R}$. The first component of the network is a single weight matrix mapping onto a vector of hidden units: $h_1 = \phi(\theta_1 x + b_0)$. The second component is a neural network with a layer of hidden units defined relative to the first layer of units $h_2 = \phi(\theta_1 h_1 + b_1)$, and outputs $y = \theta_2 h_2 + b_2$. Here, $\theta_0, \theta_1, \theta_2$ are matrices of independent Gaussian random variables and $b_0, b_1, b_2$ are vectors of independent Gaussian random variables. Given some dataset $D$ the predictive posterior distribution over outputs is $p(y|x,D)$ which we associate with the random variable $Y$. This is our (intractable) target.

Step 2: First, consider only the first component of the neural network, which maps $x$ onto $h_1$. We can construct simple constraints on $\theta_0$ and $b_0$ such that:

$$h_1 = \begin{pmatrix} \phi(z) \\ \phi(x') \end{pmatrix} ,$$

where $z \sim Z$, a unit Gaussian random variable, and $x' \in \mathbb{R}^D$, such that $\Pr (\| x' - x \| > \epsilon_1) < \delta_1$. In particular, suppose that for $\theta_0 \in \mathbb{R}^{D \times D+1}$ and $b_0 \in \mathbb{R}^{D+1}$:

$$\theta_0 = N(M_{\theta_0}, \Sigma_{\theta_0}) \text{ where } M_{\theta_0} = (0_D, I_{D \times D}); \Sigma_{\theta_0} = \sigma^2 I_{D \times D+1};$$

$$b_0 = N(\mu_{b_0}, \sigma_{b_0}) \text{ where } \mu_{b_0} = 0_{D+1}, \sigma_{b_0} = (1, \sigma, \ldots, \sigma).$$

By multiplication, straightforwardly $x' = N(\mu_{b_0}, \sigma_{b_0})$. It follows trivially that for any $\epsilon_1$ and $\delta_1$ there exists some $\sigma$ such that the bound holds.

Step 3: We show that there is a function which maps $Z$ and $x'$ onto $Y$, under reasonable assumptions. For brevity, we denote the probability density function (p.d.f.) of the true posterior predictive distribution of the random variable $Y$ conditioned on $x'$ and $D$ as $f_{Y|x'} \equiv p(Y=y|x',D)$ and the cumulative density function (c.d.f.) as $F_{Y|x'}$. We similarly write the inverse c.d.f. as $F_{Y|x'}^{-1}$. We show below:

**Lemma 2.** There exists continuous function $G_{x'}^{-1} = F_{Y|x'}^{-1} \cdot F_Z \cdot \phi^{-1}$, where $F_Z$ is the c.d.f. of the unit Gaussian and $\phi^{-1}$ is the inverse of the activation function, such that the random variable $G_{x'}^{-1}(\phi(Z))$ is equal in distribution to $Y|x$, if the p.d.f. of the posterior predictive is absolutely continuous.

The limitation to p.d.f.s is modest, this requires only that we do not completely rule out any $y$ values given some input. This is equivalent to the requirement that the c.d.f. is monotonically increasing and therefore invertible. The naming of the function $G_{x'}^{-1}$ is suggestive, and indeed its inverse exists if $F_{Y|x'}^{-1}$ is invertible, since the c.d.f. of a unit Gaussian is invertible (though this function cannot be easily expressed).

Step 4: We then use the adapted UAT to show that, by approximating $G_{x'}^{-1}(z)$, the second component of the neural network can transform $x', z$ onto $\hat{y}$ such that $\hat{Y}$ is appropriately similar to $Y$. First we show that we can approximate the function $G_{x'}^{-1}$:
Lemma 3. For a uniformly continuous function \( G_{x}^{-1}(z) : z, x' \mapsto \hat{y} \), for any \( \epsilon_2 > 0 \) and \( \delta_2 > 0 \), there exist fully-factorized Gaussian approximating distributions \( q(\theta_1), q(\theta_2), q(b_1), \) and \( q(b_2) \), and a function over the outputs of the ‘random variable mapper’ part of the neural network: \( \hat{G}^{-1}(\phi, z) \equiv \theta_2(\sigma(\theta_1 h_1) + b_1) + b_2 \) (remembering that \( h_1 \equiv (z, x') \)), such that:

\[
\Pr\left( |\hat{G}^{-1}(\phi, z) - G_{x}^{-1}(z)| > \epsilon_2 \right) < \delta_2, \quad \forall x', z.
\]

The probability measure is over the weight distributions.

Step 5: Having shown that the second component can approximate the inverse c.d.f. to within a bound, we further show that the random variable created by this transformation has a p.d.f. within a bound of the p.d.f. of \( Y \), which suffices to prove the desired result.

For this, we show this below for the transformed variable \( x' \):

Lemma 4. For any \( \epsilon_4 > 0 \) and \( \delta_2 > 0 \) there exists a mean-field weight distribution \( q(\theta) \) such that:

\[
\Pr\left( |f_{Y|x'} - f_{Y|x'}| > \epsilon_4 \right) < \delta_2, \quad \forall x'.
\]

We then move the bounds onto an expression in the original features, \( x \). Recall from step 2 that:

\[
\Pr\left( ||x' - x|| > \epsilon_1 \right) < \delta_1, \quad \forall x, x'
\]

Moreover, since we have assumed that the probability density function is continuous, these two bounds jointly ensure that:

\[
\Pr\left( |p_{\theta}(\hat{y}|x) - p(y|x, \mathcal{D})| > \epsilon \right) < \delta, \quad \forall x, y,
\]

where \( p_{\theta}(\hat{y}) \) is the p.d.f. of the output of the neural network and \( p(y|x, \mathcal{D}) \) is the p.d.f. of the true posterior distribution over the outputs.

Proof. Trivially \( \phi^{-1}(\phi(Z)) = Z \).

Let \( F_{Z} \) be the cumulative distribution function (c.d.f.) of the unit Gaussian random variable \( Z \). By the Universality of the Uniform, \( U = F_{Z}(Z) \) has a standard uniform distribution.

Let \( F_{Y|x} \) be the c.d.f. of \( Y \) conditioned on \( x \), and \( \mathcal{D} \). Suppose that the posterior predictive is non-zero everywhere (that is, you cannot rule out that there’s even the remotest chance of any \( y \) given some input \( x \), however small). Then, since \( Y \) is continuous by assumption, \( F_{Y|x} \) is invertible.

Again, by the Universality of the Uniform \( U' = F_{Y|x}(y) \) has a standard uniform distribution. So \( \forall u : p(U = u) = p(U' = u) \). Moreover \( F_{Y|x} \) is invertible. So \( p(Y = y) = F_{Y|x}^{-1}(F_{Z}(Z)) \).

It follows that there exists a continuous function as required.

Lemma 3. For a uniformly continuous function \( G_{x}^{-1}(z) : z, x' \mapsto \hat{y} \), for any \( \epsilon_2 > 0 \) and \( \delta_2 > 0 \), there exist fully-factorized Gaussian approximating distributions \( q(\theta_1), q(\theta_2), q(b_1), \) and \( q(b_2) \), and a function over the outputs of the ‘random variable mapper’ part of the neural network: \( \hat{G}^{-1}(\phi, z) \equiv \theta_2(\sigma(\theta_1 h_1) + b_1) + b_2 \) (remembering that \( h_1 \equiv (z, x') \)), such that:

\[
\Pr\left( |\hat{G}^{-1}(\phi, z) - G_{x}^{-1}(z)| > \epsilon_2 \right) < \delta_2, \quad \forall x', z.
\]

The probability measure is over the weight distributions.
Proof. The Universal Approximation Theorem (UAT) states that for any continuous function $f$, and an arbitrary fixed error, $\epsilon$, there exists a deterministic neural network with an arbitrarily wide single layer of hidden units and a non-polynomial activation, $\sigma$:

$$\forall x : |\sigma(w_2(\sigma(w_1 x) + b_1) + b_2) - f(x)| < \epsilon.$$  \hspace{1cm} (42)

In addition, we make use of Lemma 7 of \cite{Foong et al., 2020}. This states that for any $\epsilon', \delta_2 > 0$, for some fixed means $\mu_1, \mu_2, \mu_{b_1}, \mu_{b_2}$ of $q(\theta_1), \hat{q}(\theta_2), \hat{q}(b_1)$, and $q(b_2)$ respectively, there exists some standard deviation $s' > 0$ for all those approximate posteriors such that for all $s < s'$, for any $h_1 \equiv (z, x') \in \mathbb{R}^{N+1}$

$$\Pr(\|\theta\|_2(\theta_1 h_1 + b_1) + b_2) - \sigma(\mu_2(\sigma(\mu_1 h_1) + \mu_{b_1} + \mu_{b_2})) > \epsilon') < \delta_2.$$  \hspace{1cm} (43)

Note that the deterministic weights of equation (42) can just be these means. As a result:

$$\Pr(\|\theta\|_2(\theta_1 h_1 + b_1) + b_2) - f(h_1) > \epsilon + \epsilon') < \delta_2.$$  \hspace{1cm} (44)

We note that we define $\tilde{G}^{-1}(x', z) \equiv \sigma(\theta_2(\sigma(\theta_1 h_1) + b_1) + b_2)$ as above, and that $f(h_1)$ may be $G_{x'}^{-1}(z)$, which is assumed to be uniformly continuous. It follows, allowing $\epsilon_2 = \epsilon + \epsilon'$:

$$\Pr(\|\tilde{G}^{-1}(x', z) - G_{x'}^{-1}(z)\| > \epsilon_2) < \delta_2.$$  \hspace{1cm} (45)

as required.

**Lemma 4.** For any $\epsilon_4 > 0$ and $\delta_2 > 0$ there exists a mean-field weight distribution $q(\theta)$ such that:

$$\Pr(\|f_{Y|x'} - f_{Y|x'}\| > \epsilon_4) < \delta_2, \ \forall x'.$$  \hspace{1cm} (39)

In this lemma, we show that a bound on the inverse c.d.f. used to map $Z$ onto our target implies a bound in the p.d.f. of that constructed random variable to the p.d.f. of our target.

We first note that the result of Lemma 3 can be applied to $z' = \tilde{G}(x', y)$ using an inverted version of our network function such that:

$$\Pr(\|\tilde{G}^{-1}(x', \tilde{G}(x', y)) - G_{x'}^{-1}(\tilde{G}(x', y))\| > \epsilon_2) < \delta_2, \ \forall x', y.$$  \hspace{1cm} (46)

We further note that by the triangle inequality:

$$\left|G_{x'}^{-1}(G_{x'}(y)) - G_{x'}^{-1}(\tilde{G}(x', y))\right| \leq \left|G_{x'}^{-1}(G_{x'}(y)) - \tilde{G}^{-1}(x', \tilde{G}(x', y))\right|$$  \hspace{1cm} (47)

$$\left|\tilde{G}^{-1}(x', \tilde{G}(x', y)) - G_{x'}^{-1}(\tilde{G}(x', y))\right|,$$  \hspace{1cm} (48)

and since $G_{x'}^{-1}(G_{x'}(y)) = \tilde{G}^{-1}(x', \tilde{G}(x', y)) = x'$:

$$\leq \left|\tilde{G}^{-1}(x', \tilde{G}(x', y)) - G_{x'}^{-1}(\tilde{G}(x', y))\right|.$$  \hspace{1cm} (49)

Inserting this inequality into the result of Lemma 3, we have that:

$$\Pr(\left|G_{x'}^{-1}(G_{x'}(y)) - G_{x'}^{-1}(\tilde{G}(x', y))\right| > \epsilon_2) < \delta_2, \ \forall x', y.$$  \hspace{1cm} (50)

But we further note that we have assumed that the c.d.f. $G_{x'}$ is uniformly continuous (a weaker assumption than Lipschitz continuity) so for any $y'$ and $y''$, and for any $\epsilon_3 > 0$ there is an $\epsilon_4 > 0$ and vice versa, such that if:

$$|y' - y''| < \epsilon_3,$$  \hspace{1cm} (51)

then:

$$|G_{x'}(y') - G_{x'}(y'')| < \epsilon_4.$$  \hspace{1cm} (52)

It follows that for any $\epsilon_4$ and $\delta_2$ there is a $q(\theta)$ such that:

$$\Pr(\left|G_{x'}^{-1}(G_{x'}(y)) - G_{x'}^{-1}(\tilde{G}(x', y))\right| > \epsilon_4) < \delta_2, \ \forall x', y.$$  \hspace{1cm} (53)
and therefore:

$$\Pr \left( \left| (G_{x'}(y) - \hat{G}(x', y)) > \epsilon_4 \right| < \delta_2, \hspace{1em} \forall x', y. \right)$$

(54)

Next, we remember that $G_{x'} = \phi \cdot F_{Y|x'}^{-1} \cdot F_{Y|x'}$, and that $\phi$ and $F_{Y|x'}^{-1}$ are continuous, and therefore:

$$\Pr \left( \left| (F_{Y|x'}(y) - \hat{F}_{Y|x'}(y)) > \epsilon_4 \right| < \delta_2, \hspace{1em} \forall x', y, \right)$$

(55)

where $\hat{F}_{Y|x'}(y) = F_Z(\phi^{-1}(\hat{G}(x', y)))$.

As a final step, we remember that the cumulative density is the integral of the probability density function. Therefore, by Theorem 7.17 of [Rudin 1976] and the uniform convergence in the c.d.f.s, it follows that there for any bounds there exists $q(\theta)$ such that:

$$\Pr \left( |f_{Y|x'} - f_{Y|x'}| > \epsilon_4 \right) < \delta_2, \hspace{1em} \forall x',$$

(56)

as required.