Appendices

A Understanding the Soap Bubble

The emergence of a 'soap-bubble' is a well-known property in multi-variate Gaussian distributions as the number of dimensions increases (e.g., see Bishop [2006]). The observation is that, even though the highest probability density is near the mean, because there is just *so much more volume* further from the mean in high-dimensional spaces it ends up being the case that most of the probability mass is far from the mean.

One way to understand this is to examine the probability density function of the multivariate Gaussian along its radius. Consider a D-dimensional isotropic Gaussian. We examine a thin shell with thickness η, which tends to zero, at distance r between a sampled point, $\mathbf{w}$, and the mean of the multivariate Gaussian, $\boldsymbol{\mu}$. The probability density function over the radius is given by:

$$\lim_{\eta \to 0} p(r - \eta < \|\mathbf{w} - \boldsymbol{\mu}\| < r + \eta) = \frac{S_D}{(2\pi\sigma^2)^{D/2}} \cdot r^{D-1} \cdot e^{-\frac{r^2}{2\sigma^2}} \tag{6}$$

where S_D is the surface area of a hypersphere in a D-dimensional space.

The first term of the product is just a normalizing constant (S_D is the surface area of a D-dimensional hypersphere).

The second term, r^{D-1}, reflects the growing *volume* in shells away from the origin. In the region where r is small, and for the large D found in BNNs with many parameters, this term (red in figure 8) dominates and drives the probability density towards zero.

The exponential term $e^{-\frac{r^2}{2\sigma^2}}$ reflects the Gaussian density (inverse shown in green in figure 8). For larger r the exponential term becomes very small and drives the probability density towards zero. Almost all the probability mass is in the 'soap-bubble' in the region where neither term becomes very small. We consider the isotropic case here for simplicity, but the non-isotropic Gaussian has a similar 'soap-bubble'.[6]

A.1 Radial Approximate Posterior Over Each Weight

In order to achieve an approximate posterior distribution which does not have a soap bubble, we must use a lighter tailed distribution over each weight than a Gaussian would.

[6] Oh et al. [2018] consider 'soap-bubbles' in Bayesian optimization. But it has not been considered for MFVI.

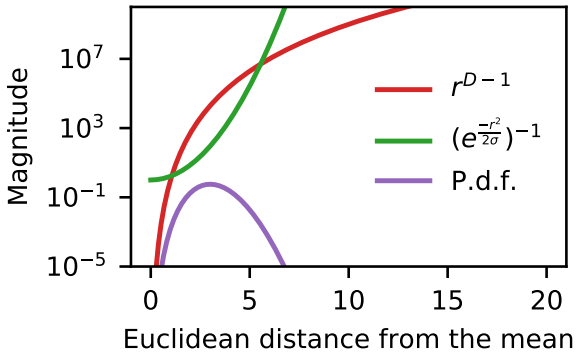

Figure 8: We can understand the 'soap-bubble' by looking at components of the p.d.f. in eq. 6. For small r the volume term (red) dominates and the normalized p.d.f. is very small. For big r the Gaussian density term (inverse shown in green) dominates and the p.d.f. is small again. Almost all the probability mass is in the intermediate region where neither term dominates: the 'soap bubble'. The intermediate region becomes narrower and further from the mean as D is bigger. Here we show $D = 10$.

In Figure 9 we show the distribution for a single weight from a Radial BNN layer with 10 weights. It is much more sharply peaked than for a typical multivariate Gaussian and has lighter tails. As a result, each individual weight in a sample from a Radial BNN is much more similar to other samples of that weight. From the perspective of viewing the model function as a sample from the entire weight-space, however, the Radial BNN distribution is more attractive, because it does not display the 'soap-bubble' pathology.

B Derivation of the Entropy Term of the KL-divergence

In this section, we show that the component of KL-divergence term of the loss which is the entropy of the posterior distribution over the weights $q(\mathbf{w}^{(x)})$ can be estimated as:

$$\mathcal{L}_{\text{entropy}} := \int q(\mathbf{w}^{(x)}) \log[q(\mathbf{w}^{(x)})] d\mathbf{w}^{(x)} \tag{7}$$

$$= -\sum_i \log[\sigma_i^{(x)}] + \text{const} \tag{8}$$

where i is an index over the weights of the model.

Throughout this section we use a superscript indicates

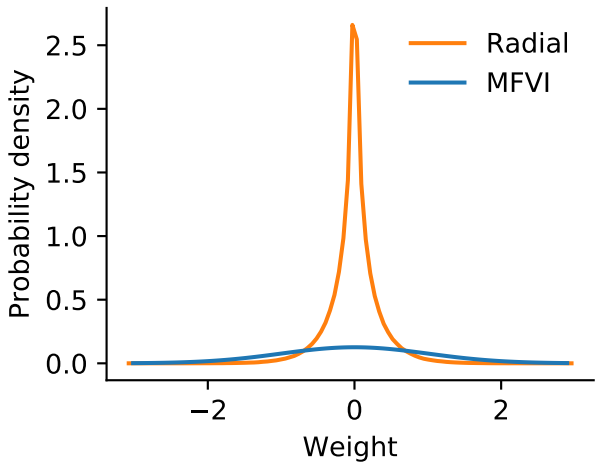

Figure 9: One-dimensional marginal distribution for a single weight. For each single weight, the Radial approximate posterior is lighter-tailed than the MFVI Gaussian, and more so the larger the layer. Here, we show a 10-dimensional layer.

the basis—an (x) means we are in the Cartesian coordinate system tied to the weight-space while (r) is the hyperspherical coordinate system (the letter is the canonical 'first' coordinate of that coordinate system).

We begin by applying the reparameterization trick [Kingma et al., 2014, Rezende et al., 2014]. Following the auxiliary variable formulation of Gal [2016], we express the probability density function of $q(\mathbf{w}^{(x)})$ with an auxiliary variable.

$$q(\mathbf{w}^{(x)}) = \int q(\mathbf{w}^{(x)}, \boldsymbol{\epsilon}^{(r)}) d\boldsymbol{\epsilon}^{(r)} \tag{9}$$

$$= \int q(\mathbf{w}^{(x)}|\boldsymbol{\epsilon}^{(r)}) q(\boldsymbol{\epsilon}^{(r)}) d\boldsymbol{\epsilon}^{(r)} \tag{10}$$

$$= \int \delta(\mathbf{w}^{(x)} - g(\mu, \sigma, \boldsymbol{\epsilon}^{(r)}) q(\boldsymbol{\epsilon}^{(r)}) d\boldsymbol{\epsilon}^{(r)}. \tag{11}$$

In equation (11), we have used a reparameterization trick transformation:

$$g(\mu, \sigma, \boldsymbol{\epsilon}^{(r)}) = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \mathbf{T}_{rx}(\boldsymbol{\epsilon}^{(r)}) \tag{12}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are parameters of the model and where $\mathbf{T}_{rx}$ is the standard transformation from hyperspherical into Cartesian coordinates.

Substituting equation (11) into the definition of the entropy loss term in equation (7), and applying the definition of the Kronecker delta we can eliminate dependence on $\mathbf{w}^{(x)}$:

$$\mathcal{L}_{\text{entropy}} = \int q(\mathbf{w}^{(x)}) \log[q(\mathbf{w}^{(x)})] d\mathbf{w}^{(x)} \tag{13}$$

$$= \int \left(\int \delta(\mathbf{w}^{(x)} - g(\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\epsilon}^{(r)}) q(\boldsymbol{\epsilon}^{(r)}) d\boldsymbol{\epsilon}^{(r)} \right) \log[q(\mathbf{w}^{(x)})] d\mathbf{w}^{(x)} \tag{14}$$

$$= \int q(\boldsymbol{\epsilon}^{(r)}) \log[q(g(\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\epsilon}^{(r)}))] d\boldsymbol{\epsilon}^{(r)}. \tag{15}$$

Then, we perform a coordinate transformation from $g(\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\epsilon}^{(r)})$ to $\boldsymbol{\epsilon}^{(r)}$ using the Jacobian of the transformation and simplify.

$$= \int q(\boldsymbol{\epsilon}^{(r)}) \log \left[q(\boldsymbol{\epsilon}^{(r)}) \left| \frac{\partial g(\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\epsilon}^{(r)})}{\partial \boldsymbol{\epsilon}^{(r)}} \right|^{-1} \right] d\boldsymbol{\epsilon}^{(r)} \tag{16}$$

$$= \int q(\boldsymbol{\epsilon}^{(r)}) \log \left[q(\boldsymbol{\epsilon}^{(r)}) \left| \prod_i \sigma_i^{(x)} \frac{\partial \epsilon_i^{(x)}}{\partial \epsilon_j^{(r)}} \right|^{-1} \right] d\boldsymbol{\epsilon}^{(r)} \tag{17}$$

$$= \int q(\boldsymbol{\epsilon}^{(r)}) \log \left[q(\boldsymbol{\epsilon}^{(r)}) \left| \text{diag}(\boldsymbol{\sigma}) \frac{\partial \epsilon_i^{(x)}}{\partial \epsilon_j^{(r)}} \right|^{-1} \right] d\boldsymbol{\epsilon}^{(r)} \tag{18}$$

$$= \int q(\boldsymbol{\epsilon}^{(r)}) \log \left[\frac{q(\boldsymbol{\epsilon}^{(r)})}{\prod_i \sigma_i^{(x)}} \left| \frac{\partial \epsilon_i^{(x)}}{\partial \epsilon_j^{(r)}} \right|^{-1} \right] d\boldsymbol{\epsilon}^{(r)} \tag{19}$$

In the last line we have used the fact that $\forall i : \sigma_i^{(x)} \geq 0$ allowing us to pull the determinant of this diagonal matrix out.

$\left| \frac{\partial \epsilon_i^{(x)}}{\partial \epsilon_j^{(r)}} \right|$ is the determinant of the Jacobian for the transformation from Cartesian to hyperspherical coordinates for which we use the result by Muleshkov and Nguyen [2016]:

$$\left| \frac{\partial \boldsymbol{\epsilon}_i^{(x)}}{\partial \boldsymbol{\epsilon}_j^{(r)}} \right| = \text{abs} \left((-1)^{D-1} (\epsilon_0^{(r)})^{D-1} \prod_{i=2}^{D} (\sin(\epsilon_i^{(r)}))^{i-1} \right). \tag{20}$$

We know that $\epsilon_0^{(r)} \geq 0$ because the radial dimension in hyperspherical coordinates can be assumed positive without loss of generality. We also know $0 \leq \epsilon_i^{(r)} \leq \pi$ for $2 \leq i \leq D$ for the hyperspherical coordinate system. So we can simplify the signs:

$$= (\epsilon_0^{(r)})^{D-1} \prod_{i=2}^{D} (\sin(\epsilon_i^{(r)}))^{i-1}. \tag{21}$$

Therefore, plugging equation (21) into (19):

$$\mathcal{L}_{\text{entropy}} = \int q(\boldsymbol{\epsilon}^{(r)}) \log \left[\frac{q(\boldsymbol{\epsilon}^{(r)})}{\text{abs}(\prod_i \sigma_i^{(x)})} \left| \frac{\partial \epsilon_i^{(x)}}{\partial \epsilon_j^{(r)}} \right|^{-1} \right] d\boldsymbol{\epsilon}^{(r)} \tag{22}$$

$$\begin{aligned} &= \int q(\boldsymbol{\epsilon}^{(r)}) \log[q(\boldsymbol{\epsilon}^{(r)})] \\ &\quad - \log[\text{abs}(\prod_i \sigma_i^{(x)})] \\ &\quad - \log \left[\left(\epsilon_0^{(r)}\right)^{D-1} \prod_{i=2}^{D} \left(\sin(\epsilon_i^{(r)}) \right)^{i-1} \right] d\boldsymbol{\epsilon}^{(r)}. \end{aligned} \tag{23}$$

Very simply, we can observe that only the middle term depends on the parameters and we must therefore only compute this term in order to compute gradients. For sake of completeness, we address the other integrals below, in case one wants to have the full value of the loss (though since it is a lower bound in any case, the full value is not very useful).

The probability density function of the noise variable is separable into independent distributions. The distribution of $\epsilon_0^{(r)}$ is a unit Gaussian. The angular dimensions are distributed so that sampling is uniform over the hypersphere. However, this does **not** mean that the distribution over each angle is uniform, as this would lead to bunching near the n-dimensional generalization of the poles. (Intuitively, there is more surface area per unit of angle near the equator, as is familiar from cartography.) Instead, we use the fact that the area element over the hypersphere is:

$$dA = d\epsilon_D^{(r)} \prod_{i=1}^{D-1} \sin(\epsilon_i^{(r)})^{D-i} d\epsilon_i^{(r)} \tag{24}$$

where we remember that $\epsilon_D^{(r)}$ is between $-\pi$ and π, and the rest of the angular elements of $\boldsymbol{\epsilon}^{(r)}$ are between 0 and π. The resulting probability density function is:

$$q(\boldsymbol{\epsilon}^{(r)}) = \prod_{i=0}^{D} q(\epsilon_i^{(r)}) \tag{25}$$

$$= \frac{1}{\sqrt{2\pi}} e^{-\frac{\epsilon_0^2}{2}} \cdot \prod_{i=1}^{D-1} \sin(\epsilon_i^{(r)})^{D-i}. \tag{26}$$

As a result, all three of the terms in equation (23) are analytically tractable. Inserting the probability density function from equation (26) into the first term of the loss, splitting up the product inside the logarithm, and separating independent terms we get:

$$\begin{aligned} &\int q(\boldsymbol{\epsilon}^{(r)}) \log[q(\boldsymbol{\epsilon}^{(r)})] d\boldsymbol{\epsilon}^{(r)} \\ &= \int_0^{\infty} d\epsilon_0^{(r)} \int_{-\pi}^{\pi} d\epsilon_D^{(r)} \int_0^{\pi} \prod_{i=1}^{D-1} d\epsilon_i^{(r)} \\ &\quad \cdot \frac{1}{\sqrt{2\pi}} e^{-\frac{\epsilon_0^2}{2}} \cdot \prod_{i=1}^{D-1} \sin(\epsilon_i^{(r)})^{D-i} \\ &\quad \cdot \log \left[\frac{1}{\sqrt{2\pi}} e^{-\frac{\epsilon_0^2}{2}} \cdot \prod_{i=1}^{D-1} \sin(\epsilon_i^{(r)})^{D-i} \right] \end{aligned} \tag{27}$$

$$\begin{aligned} &= \int_0^{\infty} d\epsilon_0^{(r)} \int_{-\pi}^{\pi} d\epsilon_D^{(r)} \int_0^{\pi} \prod_{i=1}^{D-1} d\epsilon_i^{(r)} \\ &\quad \cdot \frac{1}{\sqrt{2\pi}} e^{-\frac{\epsilon_0^2}{2}} \cdot \prod_{i=1}^{D-1} \sin(\epsilon_i^{(r)})^{D-i} \\ &\quad \cdot \log \left[\frac{1}{\sqrt{2\pi}} e^{-\frac{\epsilon_0^2}{2}} \right] + \sum_{i=1}^{D-1} \log \left[\sin(\epsilon_i^{(r)})^{D-i} \right] \end{aligned} \tag{28}$$

$$\begin{aligned} &= \int_0^{\infty} d\epsilon_0^{(r)} \frac{1}{\sqrt{2\pi}} e^{-\frac{\epsilon_0^2}{2}} \log \left[\frac{1}{\sqrt{2\pi}} e^{-\frac{\epsilon_0^2}{2}} \right] \\ &\quad + \int_{-\pi}^{\pi} d\epsilon_D^{(r)} \\ &\quad + \sum_{i=1}^{D-1} \int_0^{\pi} d\epsilon_i^{(r)} \sin(\epsilon_i^{(r)})^{D-i} \log \left[\sin(\epsilon_i^{(r)})^{D-i} \right] \end{aligned} \tag{29}$$

$$\begin{aligned} &= -\frac{\log[2\pi] + 1}{4} + 2\pi \\ &\quad + \sum_{i=1}^{D-1} \int_0^{\pi} d\epsilon_i^{(r)} \sin(\epsilon_i^{(r)})^{D-i} \log \left[\sin(\epsilon_i^{(r)})^{D-i} \right] \end{aligned} \tag{30}$$

We can simplify the first two of the terms in equation 29, but the third is difficult to solve in general (though tractable for any specific D). Regardless, this term is constant and therefore not needed for our optimization.

Inserting the probability density function from equation (26) into the second term of the loss in equation (23) we get

$$\int q(\boldsymbol{\epsilon}^{(r)}) \log[\prod_i \sigma_i^{(x)}] d\boldsymbol{\epsilon}^{(r)} = \log[\prod_i \sigma_i^{(x)}] \tag{31}$$

$$= \sum_i \log[\sigma_i^{(x)}]. \tag{32}$$

This second term is identical to the entropy of the multivariate Gaussian variational posterior typically used in MFVI.

And for the third term we begin by expanding the

logarithm and simplifying:

$$\int q(\boldsymbol{\epsilon}^{(r)}) \log \left[\left(\epsilon_0^{(r)}\right)^{D-1} \prod_{i=2}^{D} \left(\sin(\epsilon_i^{(r)}) \right)^{i-1} \right] d\boldsymbol{\epsilon}^{(r)} \quad (33)$$

$$= (D-1) \int_0^{\infty} q(\epsilon_0^{(r)}) \log[\epsilon_0^{(r)}] d\epsilon_0^{(r)} + \sum_{i=2}^{D} \frac{(i-1)}{\pi} \int_0^{\pi} \log \left[\sin(\epsilon_i^{(r)}) \right] d\epsilon_i^{(0)} \quad (34)$$

$$(35)$$

and then inserting the p.d.f. from equation (26) and solving analytically tractable integrals:

$$= \frac{(D-1)}{\sqrt{2\pi}} \int_0^{\infty} e^{-\frac{\left(\epsilon_0^{(r)}\right)^2}{2}} \log[\epsilon_0^{(r)}] d\epsilon_0^{(r)} + \sum_{i=2}^{D} \frac{(i-1)}{\pi} \cdot -\pi \log[2] \quad (36)$$

$$= \frac{(D-1)}{\sqrt{2\pi}} \cdot -\frac{1}{2}\sqrt{\frac{\pi}{2}}(\gamma + \log[2]) - \sum_{i=2}^{D}(i-1) \cdot \log[2] \quad (37)$$

$$= -\frac{(D-1)}{4} \cdot (\gamma + \log[2]) - \frac{(D-1)(D-2)}{2} \log[2] \quad (38)$$

$$= -\frac{(D-1)\gamma}{4} - \frac{(D-1)(2D-3)}{4} \log[2]. \quad (39)$$

$$(40)$$

where γ is the Euler-Mascheroni constant. This is, again, constant and may be neglected for optimization.

As a result, we can minimize the entropy term of the loss simply by finding

$$\mathcal{L}_{\text{entropy}} = -\sum_i \log[\sigma_i^{(x)}] + \text{const} \quad (41)$$

C Setting a Radial Prior

In most of our experiments, we use a typical multivariate Gaussian unit prior in order to ensure comparability with prior work. However, in some settings, such as the Variational Continual Learning setting, it is useful to use the radial posterior as a prior. We begin similarly to the previous derivation, with all unchanged expect that we are estimating

$$\mathcal{L}_{\text{cross-entropy}} = \int q(\mathbf{w}^{(x)}) \log[p(\mathbf{w}^{(x)})] d\mathbf{w}^{(x)}. \quad (42)$$

The derivation proceeds similarly until equation (23), and the second and third terms are identical except the second term taking a product over elements of $\boldsymbol{\sigma}^{(x)}_{(\text{prior})}$ of the prior, not the posterior.

Evaluating the gradient of the log probability density function of the prior depends only on the radial term, since the distribution is uniform in all angular dimensions. We therefore find

$$\int q(\boldsymbol{\epsilon}^{(r)}) \log[p\Big(\frac{\mathbf{w}^{(x)} - \boldsymbol{\mu}^{(x)}_{(\text{prior})}}{\boldsymbol{\sigma}^{(x)}_{(\text{prior})}}\Big)] d\boldsymbol{\epsilon}^{(r)}. \quad (43)$$

Rather than solve the integral, we can estimate this as a Monte Carlo approximation:

$$\approx \frac{1}{N} \sum_{i=1}^{N} -\frac{1}{2} \left\| \frac{\mathbf{w}^{(x)} - \boldsymbol{\mu}^{(x)}_{(\text{prior})}}{\boldsymbol{\sigma}^{(x)}_{(\text{prior})}} \right\|^2. \quad (44)$$

By adding the three terms we estimate the cross-entropy term of the ELBO loss function.

D Experimental Settings

D.1 Diabetic Retinopathy Settings

The diabetic retinopathy data are publicly available at `https://www.kaggle.com/c/diabetic-retinopathy-detection/data`. We augment and preprocess them similarly to Leibig et al. [2017]. The images for our main experiments in §4.1 are downsampled to 512x512 while the smaller robustness experiment in §4.1.3 uses images downsampled to 256x256 We randomly flip horizontally and vertically. Then randomly rotate 180 degrees in either direction. Then we pad by between 0 and 5% of the width and height and randomly crop back down to the intended size. We then randomly crop to between 90% and 110% of the image size, padding with zeros if needed. We finally resize again to the intended size and normalize the means and standard deviations of each channel separately based on the training set means and standard deviations. The training set has 44,594 RGB images. There are 7,026 validation and 10,000 test images.

The smaller model used for robustness experiments is loosely inspired by VGG-16, with only 16 channels, except that it is a Bayesian neural network with mean and standard deviations for each weight, and that instead of fully connected networks at the end it uses a concatenated global mean and average pool. The larger model used in the main experiments is VGG-16 but with the concatenated global mean and average pool instead of fully connected layers as above. The only difference is that we use only 46 channels, rather than 64 channels as in VGG-16, because the BNN has twice as many parameters as a similarly sized deterministic

network and we wanted to compare models with the same number of parameters. For the dropout model we use VGG-16 with the full 64 channels, and similarly for each of the models in the deep ensemble. The prior for training MFVI and Radial BNNs was a unit multivariate Gaussian. (We also tried using the scale mixture prior used in Blundell et al. [2015] and found it made no difference.) Instead of optimizing σ directly we in fact optimize ρ such that $\sigma = \log(1 + e^{\rho})$ which guarantees that σ is always positive. In some cases, as described in the paper, the first epoch only trained the means and uses a NLL loss function. This helps the optimization, but in principle can still allow the variances to train fully if early stopping is not employed (unlike reweighting the KL-divergence). Thereafter we trained using the full ELBO loss over all parameters. Unlike some prior work using MFVI, we have not downweighted the KL-divergence during training.

For the larger models, we searched for hyperparameters using Bayesian optimization. We searched between 0 and -10 as the initial value of ρ (equivalent to σ values of $\log(2)$ and $2 \cdot 10^{-9}$). For the learning rate we considered 10^{-3} to 10^{-5} using Adam with a batch size of 16. Otherwise, hyperparameters we based on exploration from the smaller model.

We then computed the test scores using a Monte Carlo estimate from averaging 16 samples from the variational distribution. We estimate the model's uncertainty about a datapoint using the mutual information between the posterior's parameters and the prediction on a datapoint. This estimate is used to rank the datapoints in order of confidence and compute the model's accuracy under the assumption of referring increasingly many points to medical experts.

For the smaller models, we performed an extensive random hyperparameter search. We tested each configuration with both MFVI and Radial BNNs. We tested each configuration for both an SGD optimizer and Amsgrad. When training with SGD we used Nesterov momentum 0.9 and uniformly sampled from 0.01, 0.001 and 0.0001 as learning rates, with a learning rate decay each epoch of either 1.0 (no decay), 0.98 or 0.96. When training with Amsgrad we uniformly sampled from learning rates of 0.001, 0.0001, and 0.00001 and did not use decay. We uniformly selected batch sizes from 16, 32, 64, 128, and 256. We uniformly selected the number of variational distribution samples used to estimate the loss from 1, 2, and 4. However, because we discarded all runs where there was insufficient graphics memory, we were only able to test up to 64x4 or 256x1 and batch sizes above 64 were proportionately less likely to appear in the final results. We selected the initial variance from ρ values of -6, -4, -2, or 0. We also tried reducing the number of convolutional channels by a factor of 5/8 or 3/8 and found that this did not seem to improve performance. We ran our hyperparameter search runs for 150 epochs. We selected the best hyperparameter configurations based on the best validation accuracy at any point during the training. We trained the models for 500 epochs but selected the models saved from 300 epochs as all models had started to overfit by the end of training. For MFVI, this was using the SGD optimizer with learning rate 0.001, decay rate 0.98 every epoch, batch size 16, 4 variational samples for estimating the loss during training and ρ of -6. This outperformed the others by a significant margin. Using our code on a V100 GPU with 8 vCPUs and an SSD this took slightly over 13 hours to train each model. For the radial posterior, this was the Adam optimizer with learning rate 0.0001, batch size 64, 1 variational sample for estimating the loss during training and a ρ of -6. Using our code on the same GPU, this took slightly over 3h to run. However, for the radial posterior there were very many other configurations with similar validation accuracies (one of the advantages of the posterior).

For the experiment shown in Figure 7, we have selected slightly different hyperparameters in order to train more quickly. For both models, we use Adam with learning rate 0.0001 and train for 500 epochs. The models have 5/8 the number of channels of VGG-16. The models are trained with batch size 64 and 4 variational samples to estimate the loss and its standard deviation.

D.2 Variational Continual Learning Settings

We build on the code provided by Nguyen et al. [2018] at `https://github.com/nvcuong/variational-continual-learning` adapted for FashionMNIST. The FashionMNIST dataset was downloaded using pytorch's built in vision datasets. The data were normalized by subtracting the training set mean and dividing by the training set standard deviation.

The classes are ordered in the conventional order. The model is initialized randomly—without pretraining the means (unlike Nguyen et al. [2018]). The model is then trained on the first two classes. The weights are carried over to the next task and set as a prior, while the model is trained on the next two classes, and so on. Note that we perform the tasks in a multi-headed way—each task has its own output head. This may not be an ideal exemplar of the continual learning problem [Chaudhry et al., 2018, Farquhar and Gal, 2018a] but it forms an effective test of the posterior. We do *not* use coresets, unlike Nguyen et al. [2018], as this would *not* form an effective test of the quality of the posterior.

Models are Bayesian MLPs with four hidden layers with 200 units in each. The prior for training was a unit multivariate Gaussian. Instead of optimizing σ directly we in fact optimize ρ such that $\sigma = \log(1 + e^{\rho})$ which guarantees that σ is always positive. Models are optimized using Amsgrad [Reddi et al., 2018] with learning rate 0.001 with shuffling and discarding final incomplete batches each epoch. We perform a grid search over the number of epochs each task is trained over (3, 5, 10, 15, 20, 60, 120) and batch sizes (1024, 2048, 10000). We used 90% of the standard training dataset (54000 points) as a training dataset, with 10% (6000 points) withheld as a validation dataset. We initialize ρ to -6 and use the initialization by He et al. [2016] for the means. The radial posterior would work with a much larger ρ, but we wanted to use comparable initializations for each. We optimized for average validation accuracy over all models on the final task. We used the standard 10000 points as a test dataset. The best configuration for the MFVI posterior was found to be 60 epochs of batch size 1024 (note that this differs from the 120 epochs of batch size 12000 reported in Nguyen et al. [2018] perhaps because they pretrain the means). The best configuration for the radial posterior was found to be 20 epochs of batch size 1024. We report the individual accuracies for each head on the test dataset.

D.3 Single-headed FashionMNIST continual learning

Previous authors have noted that for *continual learning* the single-headed environment—where the model has a single output head shared over all tasks and must therefore identify the task as well as the output based on the input—is a much harder task, possibly more reflective of continual learning [Chaudhry et al., 2018, Farquhar and Gal, 2018a]. While the multi-headed setting suffices to demonstrate improvement to the posterior, we offer some results for the single-headed setting here in the appendix for the interest of continual learning specialists, though we do not find that our posterior solves the problem.

We perform a similar grid search as before, selecting the hyperparameters that offer the highest average validation set accuracy for the final model over all five tasks. Note that in our grid search each task gets the same hyperparameters, reflecting the idea that the task distribution is not known in advance.

Our Radial BNN does not solve the continual learning single-headed problem, but it does show improved performance relative to the MFVI baseline. As we show in Figure 10, the Radial BNN shows some remembering on old tasks (which includes identifying the task that the image comes from). Moreover it is able to maintain

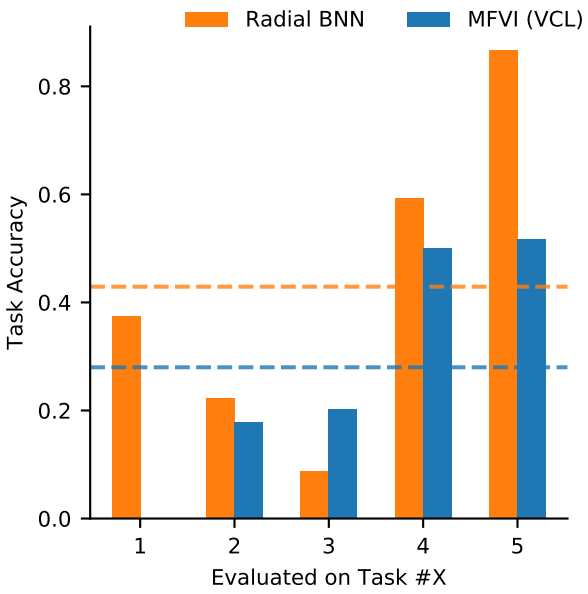

Figure 10: FashionMNIST. In the single-headed setting, where all training and testing ignores the task label, the situation is more difficult. MFVI (VCL) forgets tasks—any hyperparameter configuration that allows it to fully learn the most recent task makes it forget old tasks completely. The shown run offers the best average accuracy over all tasks for the last model. Our Radial BNN preserves information somewhat better even while training to a higher accuracy on the final task. Average accuracy is the dotted line.

good accuracy on the newest task. Meanwhile, the hyperparameters that allow MFVI to optimize last-task average accuracy mean it learns a very uncertain model which has bad accuracy on the newest tasks. This is because hyperparameters that would let it learn a high-accuracy model for the newest task would cause it to forget everything it saw earlier.

E Results on the UCI datasets

We do not believe that the standard UCI Bayesian learning experiments which are heavily used in the field offer much insight in this case. This is because all of the problems have low dimension (4-16) and because the experimental design allows only for a single hidden layer with 50 units. This is required because many of the expensive techniques that researchers develop and evaluate on the UCI datasets only scale to very small models and inputs.

For sake of completeness we show some results of our methods on the UCI datasets. As expected, our method does not outperform the more expensive techniques with complex covariances within the approximate posterior. Moreover, as expected our method performs

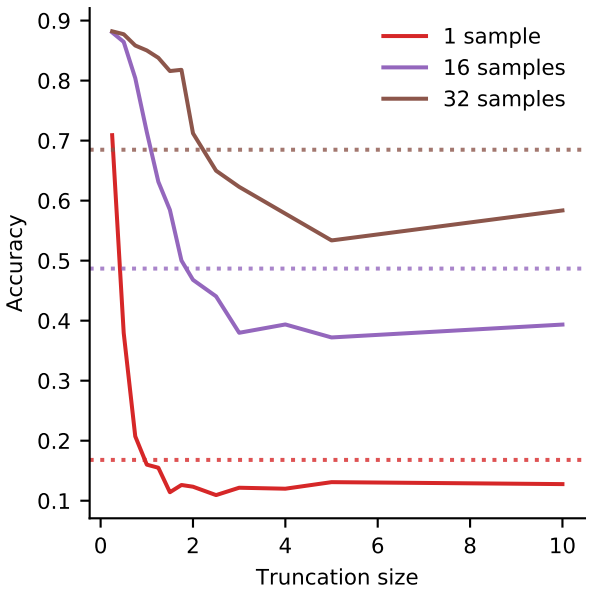

Figure 11: Dotted lines show untruncated Gaussian performance. Highly truncated Gaussians *improve* MFVI. This effect is most significant when small numbers of samples from the posterior are used to estimate the gradient. We conclude that despite bias, the low variance offered by truncation improves gradient estimates. Results averaged over 10 initial seeds for each truncation size.

very similarly to MFVI. In the small number of parameters involved in the UCI dataset experimental settings, the sampling problems for MFVI do not become severe. In this low-dimensional regime, we do not expect any particular advantage of Radial BNNs over MFVI, which is what we find.

Note that in some cases the MFVI and Radial BNN results we show are somewhat worse than those reported in other papers. We believe this is because of the fact that the resources devoted to hyperparameter search are not always the same in different papers. We only searched over learning rates of 0.001 and 0.0001 using Adam and batch sizes of 16, 64, and 1000. In the majority of datasets listed our results are competitive with what previous authors report for MFVI.

F Further Gradient Experiment

A further analysis demonstrates that we can improve the performance of MFVI models by using a low-variance but highly biased estimator of the NLL loss. We do this by estimating the NLL with a truncated version of the Gaussian sampling distribution, without changing how we analytically calculate the KL terms of the loss. We use rejection sampling, selecting only samples from a Gaussian distribution which fall under a threshold.

Our new estimate of the loss is biased (because we are not sampling from the distribution used to compute the KL divergence) but has lower variance (because only samples near the mean are used).

In Figure 11 we show that the truncated models to outperform 'correct' MFVI with standard deviations initialized slightly too high (we used $\sigma = 0.12$). This is despite the fact that we are using a biased estimator. This supports the hypothesis that MFVI training is hamstrung by high gradient variance.

Moreover, the smaller the number of samples, the higher the variance of the estimator will be, and the bigger a problem we might expect variance to be for training. Indeed, we show that the effect of truncation is smaller for larger numbers of samples. This suggests that estimating the gradient of the loss function for MFVI is hampered by sampling far from the mean, and that this effect is linked to the variance of estimates of the gradient.

Dataset	**MFVI**	**Radial**	**Dropout**	**VMG**	**FBNN**	**PBP_MV**	**DVI**
				Avg. Test LL and Std. Errors			
Boston	-2.58±0.06	-2.58± 0.05	-2.46±0.25	-2.46±0.09	-2.30±0.04	02.54±0.08	-2.41±0.02
Concrete	-5.08±0.01	-5.08±0.01	-3.04±0.09	-3.01±0.03	-3.10±0.01	-3.04±0.03	-3.06±0.01
Energy	-1.05±0.01	-0.91±0.03	-1.99±0.09	-1.06±0.03	**-0.68±0.02**	-1.01±0.01	-1.01 ± 0.06
Kin8nm	1.08±0.01	**1.35±0.00**	0.95±0.03	1.10±0.01	-	1.28±0.01	1.13±0.00
Naval	-1.57±0.01	-1.58±0.01	3.80±0.05	2.46±0.00	**7.13±0.02**	4.85±0.06	6.29±0.04
Pow. Plant	-7.54±0.00	-7.54±0.00	-2.80±0.05	-2.82±0.01	-	-2.78±0.01	-2.80±0.00
Protein	-3.67±0.00	-3.66±0.00	-2.89±0.01	-2.84±0.00	-2.89±0.00	-2.77±0.01	-2.85±0.01
Wine	-3.15±0.01	-3.15±0.01	-0.93±0.06	-0.95±0.01	-1.04±0.01	-0.97±0.01	**-0.90±0.01**
Yacht	-4.20±0.05	-4.20±0.05	-1.55±0.12	-1.30±0.02	-1.03±0.03	-1.64±0.02	**-0.47±0.03**
				Avg. Test RMSE and Std. Errors			
Boston	3.42±0.23	3.36±0.23	2.97±0.85	2.70±0.13	2.38±0.10	3.11±0.15	-
Concrete	5.71±0.15	5.62±0.14	5.23± 0.53	4.89±0.12	4.94±0.18	5.08±0.14	-
Energy	0.81±0.08	0.66±0.03	1.66±0.19	0.54±0.02	0.41±0.20	0.45±0.01	-
Kin8nm	0.37±0.00	0.16±0.00	0.10±0.00	0.08±0.00	-	**0.07±0.00**	-
Naval	0.01±0.00	0.01±0.00	0.01±0.00	0.00±0.00	0.00±0.00	0.00±0.00	-
Pow. Plant	4.02±0.04	4.04±0.04	4.02±0.18	4.04±0.04	-	3.91±0.14	-
Protein	4.40±0.02	4.34±0.03	4.36±0.04	4.13±0.02	4.33±0.03	3.94±0.02	-
Wine	0.65±0.01	0.64±0.01	0.62±0.04	0.63±0.01	0.67±0.01	0.64±0.01	-
Yacht	1.75±0.42	1.86±0.37	1.11±0.38	0.71±0.05	**0.61±0.07**	0.81±0.06	-

Table 2: Avg. test RMSE, predictive log-likelihood and s.e. for UCI regression datasets. Bold where one model is better than the next best ± their standard error. Results are from multiple papers and *hyperparameter search is not consistent*. MFVI and Radial are our implementations of standard MFVI and our proposed model respectively. Dropout is Gal and Ghahramani [2015]. Variational Matrix Gaussian (VMG) is Louizos and Welling [2016]. Functional Bayesian Neural Networks (FBNN) is Sun et al. [2019]. Probabilistic Backpropagation Matrix Variate Gaussian (PBP_MV) is Sun et al. [2017]. Deterministic VI (DVI) is Wu et al. [2019].