Dangers of Bayesian Model Averaging under Covariate Shift

Pavel Izmailov[†] **Patrick Nicholson**[‡] **Sanae Lotfi**[†] **Andrew Gordon Wilson**[†]
[†]New York University [‡]Stevens Capital Management

Abstract

Approximate Bayesian inference for neural networks is considered a robust alternative to standard training, often providing good performance on out-of-distribution data. However, Bayesian neural networks (BNNs) with high-fidelity approximate inference via full-batch Hamiltonian Monte Carlo achieve poor generalization under covariate shift, even underperforming classical estimation. We explain this surprising result, showing how a Bayesian model average can in fact be problematic under covariate shift, particularly in cases where linear dependencies in the input features cause a lack of posterior contraction. We additionally show why the same issue does not affect many approximate inference procedures, or classical maximum a-posteriori (MAP) training. Finally, we propose novel priors that improve the robustness of BNNs to many sources of covariate shift.

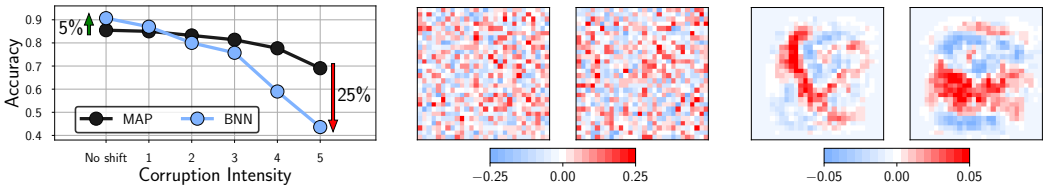
1 Introduction

The predictive distributions of deep neural networks are often deployed in critical applications such as medical diagnosis [17, 13, 14], and autonomous driving [6, 1, 36]. These applications typically involve *covariate shift*, where the target data distribution is different from the distribution used for training [19, 2]. Accurately reflecting uncertainty is crucial for robustness to these shifts [43, 47]. Since Bayesian methods provide a principled approach to representing model (epistemic) uncertainty, they are commonly benchmarked on out-of-distribution (OOD) generalization tasks [e.g., 25, 43, 8, 12, 56].

However, Izmailov et al. [22] recently showed that Bayesian neural networks (BNNs) with high fidelity inference through Hamiltonian Monte Carlo (HMC) provide shockingly poor OOD generalization performance, despite the popularity and success of approximate Bayesian inference in this setting [16, 29, 43, 35, 56, 12, 4].

In this paper, we seek to understand, further demonstrate, and help remedy this concerning behaviour. We show that Bayesian neural networks perform poorly for different types of covariate shift, namely test data corruption, domain shift, and spurious correlations. In Figure 1(a) we see that a ResNet-20 BNN approximated with HMC underperforms a maximum a-posteriori (MAP) solution by 25% on the *pixelate*-corrupted CIFAR-10 test set. This result is particularly surprising given that on the in-distribution test data, the BNN outperforms the MAP solution by over 5%.

Intuitively, we find that Bayesian model averaging (BMA) can be problematic under covariate shift as follows. Due to dependencies in the features of the train data distribution, model parameters corresponding to these dependencies do not affect the predictions on the train data. For example, parameters connected to dead pixels, i.e. pixels with intensity zero across all train images, do not affect predictions. For these parameters, the posterior coincides with the prior. The MAP solution sets the values of these parameters to zero, due to regularization from the prior that penalizes the parameter norm, while the BMA samples these weights



(a) ResNet-20, CIFAR-10-C (b) BNN weights (c) MAP weights

Figure 1: **Bayesian neural networks under covariate shift**. **(a)**: Performance of a ResNet-20 on the *pixelate* corruption in CIFAR-10-C. For the highest degree of corruption, a Bayesian model average underperforms a MAP solution by 25% (44% against 69%) accuracy. See Izmailov et al. [22] for details. **(b)**: Visualization of the weights in the first layer of a Bayesian fully-connected network on MNIST sampled via HMC. **(c)**: The corresponding MAP weights. We visualize the weights connecting the input pixels to a neuron in the hidden layer as a 28×28 image, where each weight is shown in the location of the input pixel it interacts with.

from the prior. At test time, the model is applied to a different data distribution, where the features do not have the same dependence, and the parameters that did not affect the predictions on train can negatively affect predictions on test.

In Figure 1(b, c) we visualize the weights in the first layer of a fully-connected network for a sample from the BNN posterior and the MAP solution on the MNIST dataset. The MAP solution weights are highly structured, while the BNN sample appears extremely noisy, similar to a draw from the Gaussian prior. In particular the weights corresponding to *dead pixels* (i.e. pixel positions that are black for all the MNIST images) near the boundary of the input image are set near zero (shown in white) by the MAP solution, but sampled randomly by the BNN. If at test time the data is corrupted, e.g. by Gaussian noise, and the pixels near the boundary of the image are activated, the MAP solution will ignore these pixels, while the predictions of the BNN will be significantly affected.

We further demonstrate and analyze these results, showing that the surprising lack of robustness under covariate shift for Bayesian neural networks is fundamentally caused by linear dependencies in the inputs.

Based on our understanding, we introduce a novel prior that assigns a low variance to the weights in the first layer corresponding to directions orthogonal to the data manifold, leading to improved generalization under covariate shift. We additionally study the effect of non-zero mean corruptions and accordingly propose a second prior that constrains the sum of the weights, resulting in further improvements in OOD generalization for Bayesian neural networks.

The accompanying code is available here.

2 Background

Bayesian neural networks. A Bayesian neural network model is specified by the prior distribution $p(w)$ over the weights w of the model, and the likelihood function $p(y|x, w)$, where x represents the input features and y represents the target value. Following Bayes' rule, the *posterior* distribution over the parameters w after observing the dataset $\mathcal{D} = \{(x_i, y_i)|i = 1, \ldots, n\}$ is given by

$$p(w|\mathcal{D}) = \frac{p(D|w) \cdot p(w)}{\int_{w'} p(D|w') \cdot p(w')dw'} = \frac{\prod_{i=1}^{n} p(y_i|x_i, w) \cdot p(w)}{\int_{w'} \prod_{i=1}^{n} p(y_i|x_i, w') \cdot p(w')dw'}, \tag{1}$$

where we assume that the likelihood is independent over the data points. The posterior in Equation 1 is then used to make predictions for a new input x according to the *Bayesian model average*:

$$p(y|x) = \int p(y|x, w) \cdot p(w|\mathcal{D})dw. \tag{2}$$

Unfortunately, computing the BMA in Equation 2 is intractable for Bayesian neural networks. Hence, a number of approximate inference methods have been developed. In this work, we use full-batch HMC [39, 22], as it provides a high-accuracy posterior approximation. For a more detailed discussion of Bayesian deep learning see Wilson and Izmailov [56].

Maximum a-posteriori (MAP) estimation. In contrast with Bayesian model averaging, a MAP estimator uses the single setting of weights (hypothesis) that maximizes the posterior density $w_{\text{MAP}} = \underset{w}{\text{argmax}}\, p(w|\mathcal{D}) = \underset{w}{\text{argmax}}(\log p(\mathcal{D}|w) + \log p(w))$, where the log prior can be viewed as a regularizer. For example, if we use a Gaussian prior on w, then $\log p(w)$ will penalize the ℓ_2 norm of the parameters, driving parameters that do not improve the log likelihood $\log p(\mathcal{D}|w$ to zero. MAP is the standard approach to training neural networks and our baseline for *classical training* throughout the paper. We perform MAP estimation with SGD unless otherwise specified.

Covariate shift. In this paper, we focus on the covariate shift setting. We assume the training dataset $\mathcal{D}_{\text{train}}$ consists of i.i.d. samples from the distribution $p_{\text{train}}(x, y) = p_{\text{train}}(x) \cdot p(y|x)$. However, the test data may come from a different distribution $p_{\text{test}}(x, y) = p_{\text{test}}(x) \cdot p(y|x)$. We assume the conditional distribution $p(y|x)$ remains unchanged, but the marginal distribution of the input features $p_{\text{test}}(x)$ differs from $p_{\text{train}}(x)$. Arjovsky [2] provides a detailed discussion of covariate shift.

3 Related work

Methods to improve robustness to shift between train and test often explicitly make use of the test distribution in some fashion. For example, it is common to apply semi-supervised methods to the labelled training data augmented by the unlabelled test inputs [e.g., 11, 3], or to learn a shared feature transformation for both train and test [e.g., 10]. In a Bayesian setting, Storkey and Sugiyama [50] and Storkey [49] propose such approaches for linear regression and Gaussian processes under covariate shift, assuming the data comes from multiple sources. Moreover, Shimodaira [48] propose to re-weight the train data points according to their density in the test data distribution. Sugiyama et al. [51, 52] adapt this importance-weighting approach to the cross-validation setting.

We focus on the setting of robustness to covariate shift without any access to the test distribution [e.g., 11]. Bayesian methods are frequently applied in this setting, often motivated by the ability for a Bayesian model average to provide a principled representation of epistemic uncertainty: there are typically many consistent explanations for out of distribution points, leading to high uncertainty for these points. Indeed, approximate inference approaches for Bayesian neural networks are showing good and increasingly better performance under covariate shift [e.g., 16, 29, 43, 35, 56, 12, 4].

Many works attempt to understand robustness to covariate shift. For example, for classical training Neyshabur et al. [41] show that models relying on features that encode semantic structure in the data are more robust to covariate shift. Nagarajan et al. [38] also provide insights into how classically trained max-margin classifiers can fail under covariate shift due to a reliance on spurious correlations between class labels and input features. BNN robustness to adversarial attacks is a related area of study, but generally involves much smaller perturbations to the test covariates than covariate shift. Carbone et al. [7] prove that BNNs are robust to gradient-based adversarial attacks in the large data, overparameterized

limit, while Wicker et al. [55] present a framework for training BNNs with guaranteed robustness to adversarial examples.

In general, understanding and addressing covariate shift is a large area of study. For a comprehensive overview, see Arjovsky [2]. To our knowledge, no prior work has attempted to understand, further demonstrate, or remedy the poor robustness of Bayesian neural networks with high fidelity approximate inference recently discovered in Izmailov et al. [22].

4 Bayesian neural networks are not robust to covariate shift

In this section, we evaluate Bayesian neural networks under different types of covariate shift. Specifically, we focus on two types of covariate shift: test data corruption and domain shift. In Appendix C, we additionally evaluate BNNs in the presence of spurious correlations in the data.

Methods. We evaluate BNNs against two deterministic baselines: a MAP solution approximated with stochastic gradient descent (SGD) with momentum [46, 44] and a deep ensemble of 10 independently trained MAP solutions [29]. For BNNs, we provide the results using a Gaussian prior and a more heavy-tailed Laplace prior following Fortuin et al. [15]. Izmailov et al. [22] conjectured that *cold posteriors* [54] can improve the robustness of BNNs under covariate shift; to test this hypothesis, we provide results for BNNs with a Gaussian prior and cold posteriors at temperature 10^{-2}. For all BNN models, we run a single chain of HMC for 100 iterations discarding the first 10 iterations as burn-in, following Izmailov et al. [22]. We provide additional experimental details in Appendix A.

Datasets and data augmentation. We run all methods on the MNIST [31] and CIFAR-10 [28] datasets. Following Izmailov et al. [22] we do not use data augmentation with any of the methods, as it is not trivially compatible with the Bayesian neural network framework [e.g., 22, 54].

Neural network architectures. On both the CIFAR-10 and MNIST datasets we use a small convolutional network (CNN) inspired by LeNet-5 [30], with 2 convolutional layers followed by 3 fully-connected layers. On MNIST we additionally consider a fully-connected neural network (MLP) with 2 hidden layers of 256 neurons each. We note that high-fidelity posterior sampling with HMC is extremely computationally intensive. Even on the small architectures that we consider, the experiments take multiple hours on 8 NVIDIA Tesla V-100 GPUs or 8-core TPU-V3 devices [24]. See Izmailov et al. [22] for details on the computational requirements of full-batch HMC for BNNs.

4.1 Test data corruption

We start by considering the scenario where the test data is corrupted by some type of noise. In this case, there is no semantic distribution shift: the test data is collected in the same way as the train data, but then corrupted by a generic transformation.

In Figure 2, we report the performance of the methods trained on MNIST and evaluated on the MNIST-corrupted (MNIST-C) test sets [37] under various corruptions[1]. We report the results for a fully-connected network and a convolutional network.

With the CNN architecture, deep ensembles consistently outperform BNNs. Moreover, even a single MAP solution significantly outperforms BNNs on many of the corruptions. The results are especially striking on *brightness* and *fog* corruptions, where the BNN with Laplace prior shows accuracy at the level of random guessing, while the deep ensemble retains accuracy

[1] In addition to the corruptions from MNIST-C, we consider Gaussian noise with standard deviation 3, as this corruption was considered by Izmailov et al. [22].

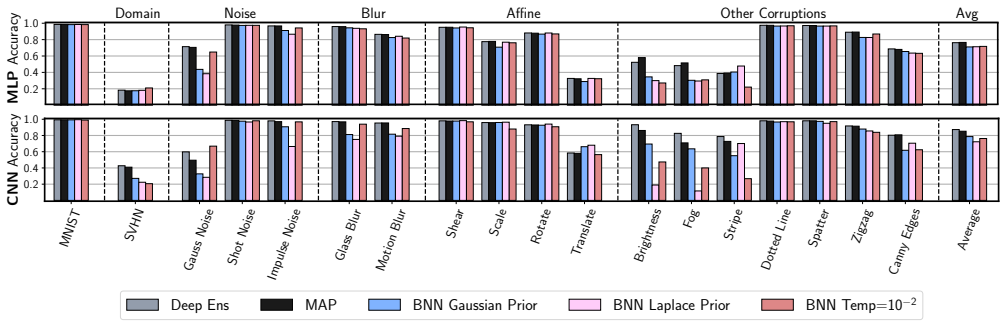


Figure 2: **Robustness on MNIST.** Accuracy for deep ensembles, MAP and Bayesian neural networks trained on MNIST under covariate shift. **Top**: Fully-connected network; **bottom**: Convolutional neural network. While on the original MNIST test set BNNs provide competitive performance, they underperform deep ensembles on most of the corruptions. With the CNN architecture, all BNN variants lose to MAP when evaluated on SVHN by almost 20%.

above 90%. *Gaussian noise* and *impulse noise* corruptions also present significant challenges for BNNs. The BNNs show the most competitive performance in-distribution and on the corruptions representing affine transformations: *shear*, *scale*, *rotate* and *translate*. The results are generally analogous for the MLP: across the board the BNNs underperform deep ensembles, and often even a single MAP solution.

While the cold posteriors provide an improvement on some of the *noise* corruptions, they also hurt performance on *Brightness* and *Stripe*, and do not improve the performance significantly on average across all corruptions compared to a standard BNN with a Gaussian prior. We provide additional results on cold posterior performance in Appendix D.

Next, we consider the CIFAR-10-corrupted dataset (CIFAR-10-C) [19]. CIFAR-10-C consists of 18 transformations that are available at different levels of intensity ($1-5$). We report the results using corruption intensity 4 (results for other intensities are in Appendix B) for each of the transformations in Figure 3. Similarly to MNIST-C, BNNs outperform deep ensembles and MAP on in-distribution data, but underperform each over multiple corruptions. On CIFAR-10-C, BNNs are especially vulnerable to different types of *noise* (*Gaussian noise*, *shot noise*, *impulse noise*, *speckle noise*). For each of the *noise* corruptions, BNNs underperform even the classical MAP solution. The cold posteriors improve the performance on the *noise* corruptions, but only provide a marginal improvement across the board.

We note that Izmailov et al. [22] evaluated a ResNet-20 model on the same set of CIFAR-10-C corruptions. While they use a much larger architecture, the qualitative results for both architectures are similar: BNNs are the most vulnerable to *noise* and *blur* corruptions. We thus expect that our paper's analysis is not specific to smaller architectures and will equally apply to deeper models.

4.2 Domain shift

Next, we consider a different type of covariate shift where the test data and train data come from different, but semantically related distributions.

First, we apply our CNN and MLP MNIST models to the SVHN test set [40]. The MNIST-to-SVHN domain shift task is a common benchmark for unsupervised domain adaptation: both datasets contain images of digits, although MNIST contains hand-written digits while SVHN represents house numbers. In order to apply our MNIST models to SVHN, we crop the SVHN images and convert them to grayscale. We report the results in Figure 2. While

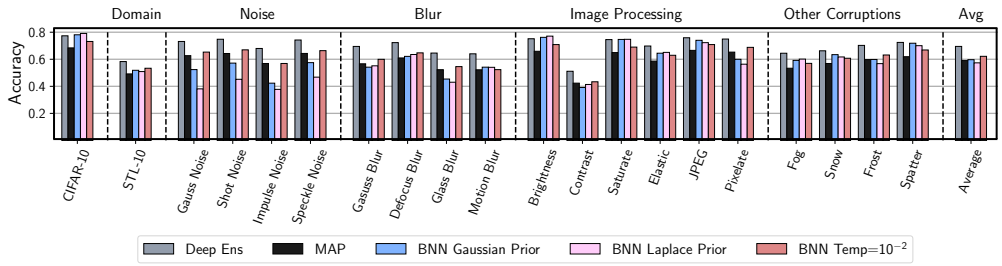


Figure 3: **Robustness on CIFAR-10.** Accuracy for deep ensembles, MAP and Bayesian neural networks using a CNN architecture trained on CIFAR-10 under covariate shift. For the corruptions from CIFAR-10-C, we report results for corruption intensity 4. While the BNNs with both Laplace and Gaussian priors outperform deep ensembles on the in-distribution accuracy, they underperform even a single MAP solution on most corruptions.

for MLPs all methods perform similarly, with the CNN architecture BNNs underperform deep ensembles and MAP by nearly 20%.

Next, we apply our CIFAR-10 CNN model to the STL-10 dataset [9]. Both datasets contain natural images with 9 shared classes between the two datasets[2]. We report the accuracy of the CIFAR-10 models on these 9 shared classes in STL-10 in Figure 3. While BNNs outperform the MAP solution, they still significantly underperform deep ensembles.

The results presented in this section highlight the generality and practical importance of the lack of robustness in BNNs: despite showing strong performance in-distribution, BNNs underperform even a single MAP solution (classical training) over an extensive variety of covariate shifts.

5 Understanding Bayesian neural networks under covariate shift

Now that we have established that Bayesian neural networks are highly susceptible to many types of covariate shift, we seek to understand why this is the case. In this section, we identify the linear dependencies in the input features as one of the key issues undermining the robustness of BNNs. We emphasize that *linear* dependencies in particular are *not* simply chosen for the simplicity of analysis, and their key role follows from the structure of the fully-connected and convolutional layers.

5.1 Motivating example: dead pixels and fully-connected layers

To provide an intuition for the results presented in this section, we start with a simple but practically relevant motivating example. Suppose we use a fully-connected Bayesian neural network on a dataset $D = \{(x_i, y_i)\}_{i=1}^n$, with m input features, where $x_i \in \mathbb{R}^m$. We use the upper indices to denote the features $x^1, \ldots, x^m$ and the lower indices to denote the datapoints x_i. Then, for each neuron j in the first hidden layer of the network, the activation can be written as $z_j = \phi(\sum_{i=1}^n x^i w_{ij}^1 + b_j^1)$, where w_{ij}^1 is the weight of the first layer of the network corresponding to the input feature i and hidden neuron j, and b_j^1 is the corresponding bias. We show the following Lemma:

Lemma 1 *Using the notation introduced above, suppose that the input feature x_k^i is equal to zero for all the examples x_k in the training dataset D. Suppose the prior distribution over the parameters $p(W)$ factorizes as $p(W) = p(w_{ij}^1) \cdot p(W \setminus w_{ij}^1)$ for some neuron j in*

[2]CIFAR-10 has a class *frog* and STL-10 has *monkey*. The other nine classes coincide.

the first layer, where $W \setminus w_{ij}^1$ represents all the parameters W of the network except w_{ij}^1. Then, the posterior distribution $p(W|D)$ will also factorize and the marginal posterior over the parameter w_{ij}^1 will coincide with the prior:

$$p(W|D) = p(W \setminus w_{ij}^1|D) \cdot p(w_{ij}^1). \tag{3}$$

Consequently, the MAP solution will set the weight w_{ij}^1 to the value with maximum prior density.

Intuitively, Lemma 1 says that if the prior for some parameter in the network is independent of the other parameters, and the value of the parameter does not affect the predictions of the model on any of the training data, then the posterior of this parameter will coincide with its prior. In particular, if one of the input features is always zero, then the corresponding weights will always be multiplied by zero, and will not affect the predictions of the network. To prove Lemma 1, we simply note that the posterior is proportional to the product of prior and likelihood, and both terms factorize with respect to w_{ij}^1. We present a formal proof in Appendix E.

So, for any sample from the posterior, the weight w_{ij}^1 will be a random draw from the prior distribution. Now suppose at test time we evaluate the model on data where the feature x^i is no longer zero. Then for these new inputs, the model will be effectively multiplying the input feature x^i by a random weight w_{ij}^1, leading to instability in predictions. In Appendix E, we formally state and prove the following proposition:

Proposition 1 (informal) *Suppose the assumptions of Lemma 1 hold. Assume also that the prior distribution $p(w_{ij}^1)$ has maximum density at 0 and that the network uses ReLU activations. Then for any test input $\bar{x}$, the expected prediction under Bayesian model averaging (Equation 2) will depend on the value of the feature $\bar{x}^i$, while the MAP solution will ignore this feature.*

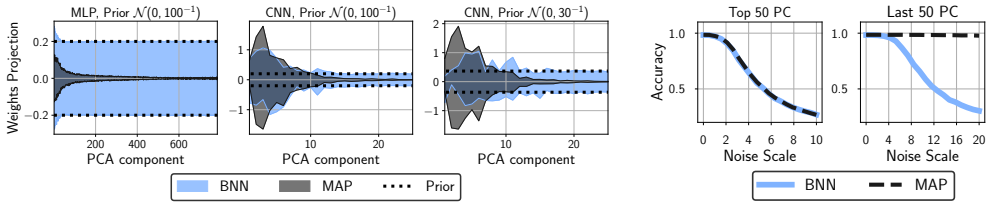
For example, in the MNIST dataset there is a large number of *dead pixels*: pixels near the boundaries of the image that have intensity zero for all the inputs. In practice, we often use independent zero-mean priors (e.g. Gaussian) for each parameter of the network. So, according to Lemma 1, the posterior over all the weights in the first layer of the network corresponding to the dead pixels will coincide with the prior. If at test time the data is corrupted by e.g., Gaussian noise, the dead pixels will receive non-zero intensities, leading to a significant degradation in the performance of the Bayesian model average compared to the MAP solution.

While the situation where input features are constant and equal to zero may be rare and easily addressed, the results presented in this section can be generalized to *any* linear dependence in the data. We will now present our results in the most general form.

5.2 General linear dependencies and fully-connected layers

We now present our general results for fully-connected Bayesian neural networks when the features are linearly dependent. Intuitively, if there exists a direction in the input space such that all of the training data points have a constant projection on this direction (i.e. the data lies in a hyper-plane), then posterior coincides with the prior in this direction. Hence, the BMA predictions are highly susceptible to perturbations that move the test inputs in a direction orthogonal to the hyper-plane. The MAP solution on the other hand is completely robust to such perturbations. In Appendix E we prove the following proposition.

Proposition 2 *Suppose that the prior over the weights w_{ij}^1 and biases b_j^1 in the first layer is an i.i.d. Gaussian distribution $\mathcal{N}(0, \alpha^2)$, independent of the other parameters in the model. Suppose all the inputs $x_1 \dots x_n$ in the training dataset D lie in an affine subspace of the input space: $\sum_{j=1}^m x_i^j c_j = c_0$ for all $i = 1, \dots, n$ and some constants c such that $\sum_{i=0}^m c_i^2 = 1$. Then,*



(a) Weight projections

(b) Robustness along PCs

Figure 4: **Bayesian inference samples weights along low-variance principal components from the prior, while MAP sets these weights to zero. (a)**: The distribution (mean $\pm$ 2 std) of projections of the weights of the first layer on the directions corresponding to the PCA components of the data for BNN samples and MAP solution using MLP and CNN architectures with different prior scales. In each case, MAP sets the weights along low-variance components to zero, while BNN samples them from the prior. **(b)**: Accuracy of BNN and MAP solutions on the MNIST test set with Gaussian noise applied along the 50 highest and 50 lowest variance PCA components of the train data (left and right respectively). MAP is very robust to noise along low-variance PCA directions, while BMA is not; the two methods are similarly robust along the highest-variance PCA components.

(1) For any neuron j in the first hidden layer, the posterior distribution of random variable $w_j^c = \sum_{i=1}^m c_i w_{ij}^1 - c_0 b_j^1$ (the projection of parameter vector $(w_{1j}^1, \ldots, w_{mj}^1, b_j^1)$ on direction $(c_1, \ldots, c_m, -c_0)$) will coincide with the prior $\mathcal{N}(0, \alpha^2)$.

(2) The MAP solution will set w_j^c to zero.

(3) (Informal) Assuming the network uses ReLU activations, at test time, the BMA prediction will be susceptible to the inputs $\bar{x}$ that lie outside of the subspace, i.e. the predictive mean will depend on $\sum_{j=1}^m \bar{x}^j c_j - c_0$. The MAP prediction will not depend on this difference.

Empirical support. To test Proposition 2, we examine the performance of a fully-connected BNN on MNIST. The MNIST training dataset is not full rank, meaning that it has linearly dependent features. For a fully-connected BNN, Proposition 2 predicts that the posterior distribution of the first layer weights projected onto directions corresponding to these linearly dependent features will coincide with the prior. In Figure 4(a) we test this hypothesis by projecting first layer weights onto the principal components of the data. As expected, the distribution of the projections on low-variance PCA components (directions that are constant or nearly constant in the data) almost exactly coincides with the prior. The MAP solution, on the other hand, sets the weights along these PCA components close to zero, confirming conclusion (2) of the proposition. Finally, in Figure 4(b) we visualize the performance of the BMA and MAP solution as we apply noise along high-variance and low-variance directions in the data. As predicted by conclusion (3) of Proposition 2, the MAP solution is very robust to noise along the low-variance directions, while BMA is not.

5.3 Linear dependencies and convolutional layers

Finally, we can extend Proposition 2 to convolutional layers. Unlike fully-connected layers, convolutional layers include weight sharing such that no individual weight corresponds to a specific pixel in the input images. For example, dead pixels will not necessarily present an issue for convolutional layers, unlike what is described in subsection 5.1. However, convolutional layers are still susceptible to a special type of linear dependence. Intuitively, we can think of the outputs of the first convolutional layer on all the input images as the outputs of a fully-connected layer applied to all the $K \times K$ patches of the input image. Therefore the reasoning in Proposition 2 applies to the convolutional layers, with the difference that

the linear dependencies in the $K \times K$ patches cause the instability rather than dependencies in the full feature space. In Appendix E we prove the following proposition.

Proposition 3 *Suppose that the prior over the parameters of the convolutional filters and biases in the first layer is an i.i.d. Gaussian distribution $\mathcal{N}(0, \alpha^2)$, independent of the other parameters in the model. Suppose that the convolutional filters in the first layer are of size $K \times K \times C$, where C is the number of input channels. Then, consider the set $\hat{D}$ of size N of all the patches of size $K \times K \times C$ extracted from the training images in D after applying the same padding as in the first convolutional layer. Suppose all the patches $z_1 \dots z_N$ in the dataset $\hat{D}$ lie in an affine subspace of the space $\mathbb{R}^{K \times K \times C}$: $\sum_{c=1}^{C} \sum_{a=1}^{K} \sum_{b=1}^{K} z_i^{a,b} \gamma_{c,a,b} = \gamma_0$ for all $i = 1, \dots, N$ and some constants c_i such that $\sum_{c=1}^{C} \sum_{a=1}^{K} \sum_{b=1}^{K} \gamma_{c,a,b}^2 + \gamma_0^2 = 1$. Then, we can prove results analogous to (1)-(3) in Proposition 2 (see the Appendix E for the details).*

Empirical support. In Figure 4(a), we visualize the projections of the weights in the first layer of the CNN architecture on the PCA components of the $K \times K$ patches extracted from MNIST. Analogously to fully-connected networks, the projections of the MAP weights are close to zero for low-variance components, while the projections of the BNN samples follow the prior.

5.4 What corruptions will hurt performance?

Based on Propositions 2, 3, we expect that the corruptions that are the most likely to break linear dependence structure in the data will hurt the performance the most. In Appendix F we argue that *noise* corruptions are likely to break linear dependence, while the *affine* corruptions are more likely to preserve it, agreeing with our observations in section 4.

5.5 Why do some approximate Bayesian inference methods work well under covariate shift?

Unlike BNNs with HMC inference, some approximate inference methods such as SWAG [35], MC dropout [16], and deep ensembles [29] provide strong performance under covariate shift [43, 22]. For deep ensembles, we can easily understand why: a deep ensemble represents an average of approximate MAP solutions, and we have seen in conclusion (3) of Proposition 2 that MAP is robust to covariate shift in the scenario introduced in subsection 5.2. Similarly, other methods are closely connected to MAP via characterizing the posterior using MAP optimization iterates (SWAG), or modifying the training procedure for the MAP solution (MC Dropout). We provide further details in Appendix G.

6 Towards more robust Bayesian model averaging under covariate shift

In this section, we propose a new simple prior inspired by our theoretical analysis. In section 5 we showed that linear dependencies in the input features cause the posterior to coincide with the prior along the corresponding directions in the parameter space. In order to address this issue, we explicitly design the prior for the first layer of the network so that the prior variance is low along these directions.

6.1 Data empirical covariance prior

Let us consider the empirical covariance matrix of the inputs x_i. Assuming the input features are all preprocessed to be zero-mean $\sum_{i=1}^{n} x_i = 0$, we have $\Sigma = \frac{1}{n-1} \sum_{i=1}^{n} x_i x_i^T$. For fully-connected networks, we propose to use the *EmpCov* prior $p(w^1) = \mathcal{N}(0, \alpha\Sigma + \epsilon I)$ on the weights w^1 of the first layer of the network, where ϵ is a small positive constant that is used

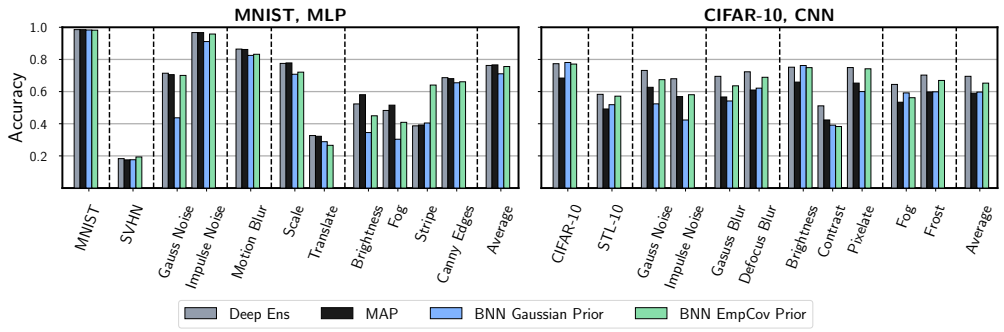


Figure 5: **EmpCov prior improves robustness.** Test accuracy under covariate shift for deep ensembles, MAP solution approximated by SGD, and BNN with Gaussian and *EmpCov* priors. **Left**: MLP architecture trained on MNIST. **Right**: CNN architecture trained on CIFAR-10. The *EmpCov* prior provides consistent improvement over the standard Gaussian prior. The improvement is particularly noticeable on the *noise* corruptions and in domail shift experiments (SVHN, STL-10).

to ensure that the covariance matrix is positive definite. The parameter $\alpha > 0$ determines the scale of the prior.

Suppose there is a linear dependence in the input features of the data: $x_i^T p = c$ for some direction p and constant c. Then p will be an eigenvector of the empirical covariance matrix with the corresponding eigenvalue equal to 0 : $\Sigma p = \frac{1}{n-1}\sum_{i=1}^{n} x_i x_i^T p = \frac{c}{n-1}\sum_{i=1}^{n} x_i = 0$. Hence the prior over w^1 will have a variance of ϵ along the direction p.

More generally, the *EmpCov* prior is aligned with the principal components of the data, which are the eigenvectors of the matrix Σ. The prior variance along each principal component p_i is equal to $\alpha\sigma_i^2 + \epsilon$ where σ_i^2 is its corresponding explained variance. In Appendix H, we discuss a more general family of priors aligned with the principal components of the data.

Generalization to convolutions. We can generalize the *EmpCov* prior to convolutions by replacing the empirical covariance of the data with the empirical covariance of the patches that interact with the convolutional filter, denoted by $\hat{D}$ in Proposition 3.

Is EmpCov a valid prior? *EmpCov* constructs a valid prior by evaluating the empirical covariance matrix of the inputs. This prior does not depend on the train data labels y_i, unlike the approach known as *Empirical Bayes* [see e.g. 5, section 3.5], which is commonly used to specify hyperparameters in Gaussian process and neural network priors [45, 34].

6.2 Experiments

In Figure 5 we report the performance of the BNNs using the *EmpCov* prior. In each case, we apply the *EmpCov* prior to the first layer, and a Gaussian prior to all other layers. For more details, please see Appendix A.

On both MLP on MNIST and CNN on CIFAR-10, the *EmpCov* prior significantly improves performance across the board. In both cases, the BNN with *EmpCov* prior shows competitive performance with deep ensembles, especially in the domain shift experiments. *EmpCov* is also particularly useful on the *noise* corruptions.

In Appendix I, we provide a detailed analysis of the performance of the CNN architecture on MNIST. Surprisingly, we found that using the *EmpCov* prior by itself does not provide a large improvement in this case. In Appendix I, we identify an issue specific to this particular setting, and propose another targeted prior that substantially improves performance.

7 Discussion

We consider the generality of our results, additional perspectives, and future directions.

Why focus on linear dependencies? This focus is dictated by the structure of both fully-connected and convolutional neural networks, which apply activation functions to linear combinations of features. Our results can be directly extended to other models, where different types of dependencies between input features would lead to the same lack of robustness to covariate shift. For example, in Appendix J we derive analogous results to subsection 5.2 for multiplicative neural network architectures [53], where a different form of dependence between features causes poor robustness.

Intermediate layers. While our analysis in section 5 is focused on the linear dependencies in the input features, similar conclusions can be made about intermediate layers of the network. For example, weights connected to *dead neurons*, which output zero, do not affect predictions of the model. We provide more details in Appendix K.

Linear Bayesian models. In models that are linear in parameters, such as Bayesian linear regression and Gaussian processes, we are typically saved from the perils of BMA under covariate shift discussed in section 5, because the MAP and the predictive mean under BMA coincide. We provide further details in Appendix L.

An optimization perspective on the covariate shift problem.

In Section 5.1, we argued that SGD is more robust to covariate shift than HMC because the regularizer pushes the weights that correspond to dead pixels towards zero. This effect is mainly obtained through explicit regularization, without which these weights will remain at their initial values. We study the effect of initialization and explicit regularization on the performance of SGD under covariate shift in Appendix M.1. We also show in Appendix M.2 that other stochastic optimizers such as Adam [27] and Adadelta [57] behave similarly to SGD under covariate shift. Finally, we discuss the effect of test data corruptions on the loss landscape in Appendix M.3 and argue that the relative sharpness of low density posterior samples makes these solutions, which are included in a BMA but not MAP, more vulnerable to covariate shift.

Limitations. Due to the intense computational requirements of HMC, our experiments are limited to smaller models and datasets. Our analysis is focused on issues arising in models that are non-linear in their parameters. Moreover, while our proposed priors help improve robustness, they do not entirely resolve the issue. For example, in the *Contrast* dataset of Figure 5 (right panel), the BMA is still underperforming MAP.

Conclusion. Our work has demonstrated, both empirically and theoretically, how linear dependencies in the training data cause Bayesian neural networks to generalize poorly under covariate shift — explaining the important and unexpected findings in Izmailov et al. [22]. While the two priors we introduce achieve some improvement in BNN performance under covariate shift, we are only beginning to explore possible remedies. Our work is intended as a step towards understanding the true properties of Bayesian neural networks, and improving the robustness of Bayesian model averaging under covariate shift.

Acknowledgements

We thank Martin Arjovsky, Behnam Neyshabur, Vaishnavh Nagarajan, Marc Finzi, Polina Kirichenko, Greg Benton and Nate Gruver for helpful discussions. This research is supported with Cloud TPUs from Google's TPU Research Cloud (TRC), and by an Amazon Research Award, NSF I-DISRE 193471, NIH R01DA048764-01A1, NSF IIS-1910266, and NSF 1922658NRT-HDR: FUTURE Foundations, Translation, and Responsibility for Data Science.

References

[1] Maruan Al-Shedivat, Andrew Gordon Wilson, Yunus Saatchi, Zhiting Hu, and Eric P Xing. Learning scalable deep kernels with recurrent structure. *The Journal of Machine Learning Research*, 18(1):2850–2886, 2017.

[2] Martin Arjovsky. Out of distribution generalization in machine learning. *arXiv preprint arXiv:2103.02667*, 2021.

[3] Ben Athiwaratkun, Marc Finzi, Pavel Izmailov, and Andrew Gordon Wilson. There are many consistent explanations of unlabeled data: Why you should average. *arXiv preprint arXiv:1806.05594*, 2018.

[4] Gregory W Benton, Wesley J Maddox, Sanae Lotfi, and Andrew Gordon Wilson. Loss surface simplexes for mode connecting volumes and fast ensembling. *arXiv preprint arXiv:2102.13042*, 2021.

[5] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[6] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[7] Ginevra Carbone, Matthew Wicker, Luca Laurenti, Andrea Patane, Luca Bortolussi, and Guido Sanguinetti. Robustness of bayesian neural networks to gradient-based attacks. *arXiv preprint arXiv:2002.04359*, 2020.

[8] Oscar Chang, Yuling Yao, David Williams-King, and Hod Lipson. Ensemble model patching: A parameter-efficient variational bayesian neural network. *arXiv preprint arXiv:1905.09453*, 2019.

[9] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223. JMLR Workshop and Conference Proceedings, 2011.

[10] Hal Daumé III. Frustratingly easy domain adaptation. *arXiv preprint arXiv:0907.1815*, 2009.

[11] Hal Daume III and Daniel Marcu. Domain adaptation for statistical classifiers. *Journal of artificial Intelligence research*, 26:101–126, 2006.

[12] Michael Dusenberry, Ghassen Jerfel, Yeming Wen, Yian Ma, Jasper Snoek, Katherine Heller, Balaji Lakshminarayanan, and Dustin Tran. Efficient and scalable bayesian neural nets with rank-1 factors. In *International conference on machine learning*, pages 2782–2792, 2020.

[13] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *nature*, 542(7639):115–118, 2017.

[14] Angelos Filos, Sebastian Farquhar, Aidan N Gomez, Tim GJ Rudner, Zachary Kenton, Lewis Smith, Milad Alizadeh, Arnoud de Kroon, and Yarin Gal. A systematic comparison of bayesian deep learning robustness in diabetic retinopathy tasks. *arXiv preprint arXiv:1912.10481*, 2019.

[15] Vincent Fortuin, Adrià Garriga-Alonso, Florian Wenzel, Gunnar Rätsch, Richard Turner, Mark van der Wilk, and Laurence Aitchison. Bayesian neural network priors revisited. *arXiv preprint arXiv:2102.06571*, 2021.

[16] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[17] Varun Gulshan, Lily Peng, Marc Coram, Martin C Stumpe, Derek Wu, Arunachalam Narayanaswamy, Subhashini Venugopalan, Kasumi Widner, Tom Madams, Jorge Cuadros, et al. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *Jama*, 316(22):2402–2410, 2016.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[19] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1): 1–42, 1997.

[21] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. In *Uncertainty in Artificial Intelligence (UAI)*, 2018.

[22] Pavel Izmailov, Sharad Vikram, Matthew D Hoffman, and Andrew Gordon Wilson. What are bayesian neural network posteriors really like? *arXiv preprint arXiv:2104.14421*, 2021.

[23] Jörn-Henrik Jacobsen, Jens Behrmann, Richard Zemel, and Matthias Bethge. Excessive invariance causes adversarial vulnerability. *arXiv preprint arXiv:1811.00401*, 2018.

[24] Norman P Jouppi, Doe Hyun Yoon, George Kurian, Sheng Li, Nishant Patil, James Laudon, Cliff Young, and David Patterson. A domain-specific supercomputer for training deep neural networks. *Communications of the ACM*, 63(7):67–78, 2020.

[25] Alex Kendall and Yarin Gal. What uncertainties do we need in Bayesian deep learning for computer vision? In *Advances in neural information processing systems*, pages 5574–5584, 2017.

[26] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[27] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[28] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The CIFAR-10 dataset. 2014. http://www.cs.toronto.edu/kriz/cifar.html.

[29] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6402–6413, 2017.

[30] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[31] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[32] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913*, 2017.

[33] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[34] David JC MacKay. Probable networks and plausible predictions—a review of practical bayesian methods for supervised neural networks. *Network: computation in neural systems*, 6(3):469–505, 1995.

[35] Wesley J Maddox, Pavel Izmailov, Timur Garipov, Dmitry P Vetrov, and Andrew Gordon Wilson. A simple baseline for bayesian uncertainty in deep learning. *Advances in Neural Information Processing Systems*, 32:13153–13164, 2019.

[36] Rhiannon Michelmore, Matthew Wicker, Luca Laurenti, Luca Cardelli, Yarin Gal, and Marta Kwiatkowska. Uncertainty quantification with statistical guarantees in end-to-end autonomous driving control. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7344–7350. IEEE, 2020.

[37] Norman Mu and Justin Gilmer. Mnist-c: A robustness benchmark for computer vision. *arXiv preprint arXiv:1906.02337*, 2019.

[38] Vaishnavh Nagarajan, Anders Andreassen, and Behnam Neyshabur. Understanding the failure modes of out-of-distribution generalization. *arXiv preprint arXiv:2010.15775*, 2020.

[39] Radford M Neal et al. Mcmc using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

[40] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[41] Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. What is being transferred in transfer learning? *arXiv preprint arXiv:2008.11687*, 2020.

[42] Jorge Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980.

[43] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D Sculley, Sebastian Nowozin, Joshua V Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. *arXiv preprint arXiv:1906.02530*, 2019.

[44] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[45] C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for Machine Learning*. The MIT Press, 2006.

[46] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[47] Abhijit Guha Roy, Jie Ren, Shekoofeh Azizi, Aaron Loh, Vivek Natarajan, Basil Mustafa, Nick Pawlowski, Jan Freyberg, Yuan Liu, Zach Beaver, et al. Does your dermatology classifier know what it doesn't know? detecting the long-tail of unseen conditions. *arXiv preprint arXiv:2104.03829*, 2021.

[48] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.

[49] Amos Storkey. When training and test sets are different: characterizing learning transfer. *Dataset shift in machine learning*, 30:3–28, 2009.

[50] Amos J Storkey and Masashi Sugiyama. Mixture regression for covariate shift. *Advances in neural information processing systems*, 19:1337, 2007.

[51] Masashi Sugiyama, Benjamin Blankertz, Matthias Krauledat, Guido Dornhege, and Klaus-Robert Müller. Importance-weighted cross-validation for covariate shift. In *Joint Pattern Recognition Symposium*, pages 354–363. Springer, 2006.

[52] Masashi Sugiyama, Matthias Krauledat, and Klaus-Robert Müller. Covariate shift adaptation by importance weighted cross validation. *Journal of Machine Learning Research*, 8(5), 2007.

[53] Andrew Trask, Felix Hill, Scott Reed, Jack Rae, Chris Dyer, and Phil Blunsom. Neural arithmetic logic units. *arXiv preprint arXiv:1808.00508*, 2018.

[54] Florian Wenzel, Kevin Roth, Bastiaan S Veeling, Jakub Świątkowski, Linh Tran, Stephan Mandt, Jasper Snoek, Tim Salimans, Rodolphe Jenatton, and Sebastian Nowozin. How good is the Bayes posterior in deep neural networks really? *arXiv preprint arXiv:2002.02405*, 2020.

[55] Matthew Wicker, Luca Laurenti, Andrea Patane, Zhuotong Chen, Zheng Zhang, and Marta Kwiatkowska. Bayesian inference with certifiable adversarial robustness. In *International Conference on Artificial Intelligence and Statistics*, pages 2431–2439. PMLR, 2021.

[56] Andrew Gordon Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. *arXiv preprint arXiv:2002.08791*, 2020.

[57] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Appendix Outline

This appendix is organized as follows.

- In Appendix A, we provide details on hyper-parameters, datasets and architectures used in our experiments.
- In Appendix B, we examine BNN performance under covariate shift for a variety of different standard priors with different hyper-parameter settings.
- In Appendix C, we study the effect of spurious correlations on BMA performance using the shift-MNIST dataset.
- In Appendix D, we study how temperature scaling impacts BMA performance under covariate shift.
- In Appendix E, we provide proofs of our propositions from section 5.
- In Appendix F, we visualize how different corruptions introduce noise along different principal components of the data, and relate this to BMA performance on these corruptions.
- In Appendix G, we explain why approximate inference methods SWAG and MC Dropout do not suffer the same performance degradation under covariate shift as HMC.
- In Appendix H, we analyze a more general family of priors that includes the *EmpCov* prior from section 6.
- In Appendix I, we introduce the sum filter prior for improving BNN robustness to non-zero-mean noise.
- In Appendix J, we provide an example of a model architecture where the BMA will be impacted by nonlinear dependencies in the training data.
- In Appendix K, we examine how BNNs can be impacted by linear dependencies beyond the first layer using the example of dead neurons.
- In Appendix L we prove that linear dependencies do not hurt BMAs of linear models under covariate shift.
- In Appendix M, we examine covariate shift from an optimization perspective.
- Lastly, in Appendix N, we provide details on licensing.

A Hyper-parameters and details of experiments

A.1 Prior definitions

Here we define the prior families used in the main text and the appendix, and the corresponding hyper-parameters.

Gaussian priors. We consider iid Gaussian priors of the form $\mathcal{N}(0, \alpha^2 I)$, where α^2 is the prior variance. Gaussian priors are the default choice in Bayesian neural networks [e.g. 15, 22, 56].

Laplace priors. We consider priors of the form $\text{Laplace}(\alpha) : \frac{1}{2\alpha} \exp(-\|x\|_1/\alpha)$, where $\|\cdot\|_1$ is the ℓ_1-norm.

Student-t priors. In Appendix B, we consider iid Student-t priors of the form $\text{Student-}t(\nu, \alpha^2) : \frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})\sqrt{\nu\pi}}(1 + \frac{w^2}{\nu\alpha^2})^{-\frac{\nu+1}{2}}$, where ν represents the degrees of freedom and α^2 is the prior variance.

Exp-norm priors. In Appendix B, we consider the prior family of the form $\text{ExpNorm}(p, \alpha^2) : \exp(-\|w\|^p/2\alpha^2)$. Notice that for $p = 2$, we get the Gaussian prior family. By varying p we can construct more heavy-tailed ($p < 2$) or less heavy-tailed ($p > 2$) priors.

A.2 Hyper-parameters and details

HMC hyper-parameters. The hyper-parameters for HMC are the step size, trajectory length and any hyper-parameters of the prior. Following Izmailov et al. [22], we set the trajectory length $\tau = \frac{\pi\sigma_{prior}}{2}$ where σ_{prior} is the standard deviation of the prior. We choose the step size to ensure that the accept rates are high; for most of our MLP runs we do 10^4 leapfrog steps per sample, while for CNN we do $5 \cdot 10^3$ leapfrog steps per sample.

Data Splits. For all CIFAR-10 and MNIST experiments, we use the standard data splits: 50000 training samples for CIFAR-10, 60000 training samples for MNIST, and 10000 test samples for both. For all data corruption experiments, we evaluate on the corrupted 10000 test samples. For domain shift experiments, we evaluate on 26032 SVHN test samples for MNIST to SVHN and 7200 STL-10 test samples for CIFAR-10 to STL-10. In all cases, we normalize the inputs using train data statistics, and do not use any data augmentation.

Neural network architectures. Due to computational constraints, we use smaller neural network architectures for our experiments. All architectures use ReLU activations. For MLP experiments, we use a fully-connected network with 2 hidden layers of 256 neurons each. For CNN experiments, we use a network with 2 convolutional layers followed by 3 fully-connected layers. Both convolutional layers have 5×5 filters, a stride of 1, and use 2×2 average pooling with stride 2. The first layer has 6 filters and uses padding, while the second layer has 16 filters and does not use padding. The fully connected layers have 400, 120, and 84 hidden units.

MAP and deep ensemble hyper-parameters. We use the SGD optimizer with momentum 0.9, cosine learning rate schedule and weight decay 100 to approximate the MAP solution. In Appendix M we study the effect of using other optimizers and weight decay values. On MNIST, we run SGD for 100 epochs, and on CIFAR we run for 300 epochs. For the deep ensemble baselines, we train 10 MAP models independently and ensemble their predictions.

Prior hyper-parameters. To select prior hyper-parameters we perform a grid search, and report results for the optimal hyperparameters in order to compare the best versions of different models and priors. We report the prior hyper-parameters used in our main evaluation in Table 1. In Appendix B we provide detailed results for various priors with different hyper-parameter choices.

Table 1: Prior hyper-parameters

Hyper-parameter	MNIST MLP	MNIST CNN	CIFAR-10 CNN
BNN, Gaussian prior; α^2	$\frac{1}{100}$	$\frac{1}{100}$	$\frac{1}{100}$
BNN, Laplace prior; α	$\sqrt{\frac{1}{6}}$	$\sqrt{\frac{1}{6}}$	$\sqrt{\frac{1}{200}}$

Tempering hyper-parameters. For the tempering experiments, we use a Gaussian prior with variance $\alpha^2 = \frac{1}{100}$ on MNIST and $\alpha^2 = \frac{1}{3}$ on CIFAR-10. We set the posterior temperature to 10^{-2}. We provide additional results for other prior variance and temperature combinations in Appendix D.

Compute. We ran all the MNIST experiments on TPU-V3-8 devices, and all CIFAR experiments on 8 NVIDIA Tesla-V100 devices. A single HMC chain with 100 iterations

on these devices takes roughly 1.5 hours for MNIST MLP, 2 hours for CIFAR CNN and 3 hours for MNIST CNN. As a rough upper-bound, we ran on the order of 100 different HMC chains, each taking 2 hours on average, resulting in 200 hours on our devices, or roughly 1600 GPU-hours (where we equate 1 hour on TPU-V3-8 to 8 GPU-hours).

B Additional results on BNN robustness

B.1 Error-bars and additional metrics

We report the accuracy, log-likelihood and expected calibration error (ECE) for deep ensembles, MAP solutions and BMA variations in Figure 6. We report the results for different corruption intensities (1, 3, 5) and provide error-bars computed over 3 independent runs. Across the board, *EmpCov* priors provide the best performance among BNN variations on all three metrics.

B.2 Detailed results for different priors

In this section, we evaluate BNNs with several prior families and provide results for different choices of hyper-parameters. The priors are defined in subsection A.1.

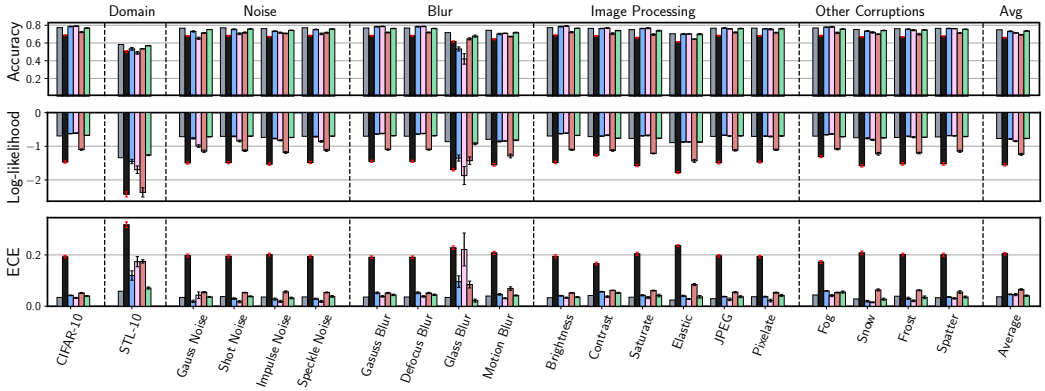
We report the results using CNNs and MLPs on MNIST in Figure 7. None of the considered priors completely close the gap to MAP under all corruptions. Gaussian priors show the worst results, losing to MAP on all MNIST-C corruptions and Gaussian noise, at all prior standard deviations. Laplace priors show similar results to Gaussian priors under Gaussian noise, but beat MAP on the *stripe* corruption in MNIST-C. Student-t priors show better results, matching or outperforming MAP on all affine transformations, but still underpeform significantly under *Gaussian noise*, *Brightness* and *Fog* corruptions. Finally, exp-norm priors can match MAP on Shot noise and also outperform MAP on *stripe*, but lose on other corruptions. The in-domain performance with exp-norm priors is also lower compared to the other priors considered.

To sum up, none of the priors considered is able to resolve the poor robustness of BNNs under covariate shift. In particular, all priors provide poor performance under *Gaussian noise*, *Brightness* and *Fog* corruptions.

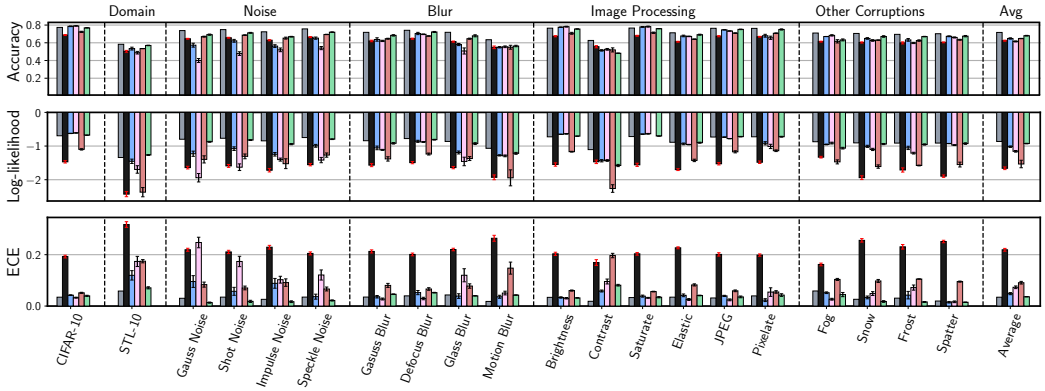
C Bayesian neural networks and spurious correlations

For corrupted data, models experience worse performance due to additional noisy features being introduced. However, it's also possible that the reverse can occur, and a seemingly highly predictive feature in the training data will not be present in the test data. This distinct category of covariate shift is often called spurious correlation. To test performance with spurious correlations, we use the Shift-MNIST dataset [23], where we introduce spurious features via modifying the training data so that a set of ten pixels in the image perfectly correlates with class labels.

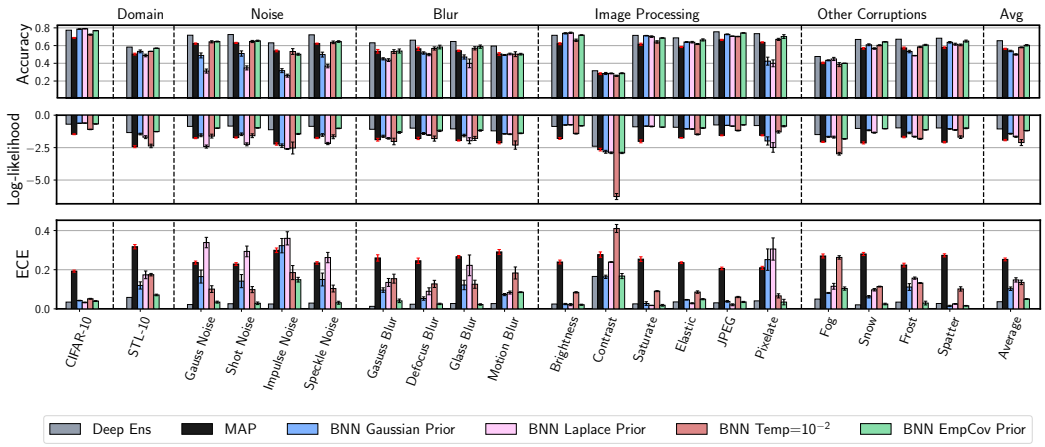
Table 2 shows the results for deep ensembles, MAP and BNNs with Gaussian and *EmpCov* priors on the Shift-MNIST dataset. We see worse accuracy for CNN architectures, demonstrating how more complex architectures can more easily over-fit the spurious correlations. BNNs with Gaussian prior perform better than MAP on both MLP and CNN, but significantly worse than deep ensembles for CNNs. Notice that the *EmpCov* prior does not improve performance here for either architecture, highlighting the difference between spurious correlations and other forms of covariate shift. In particular, the largest principal components of the Shift-MNIST training dataset place large magnitude weights on the spurious features, and so using the *EmpCov* prior results in samples with larger weights for



(a) Corruption Intensity 1

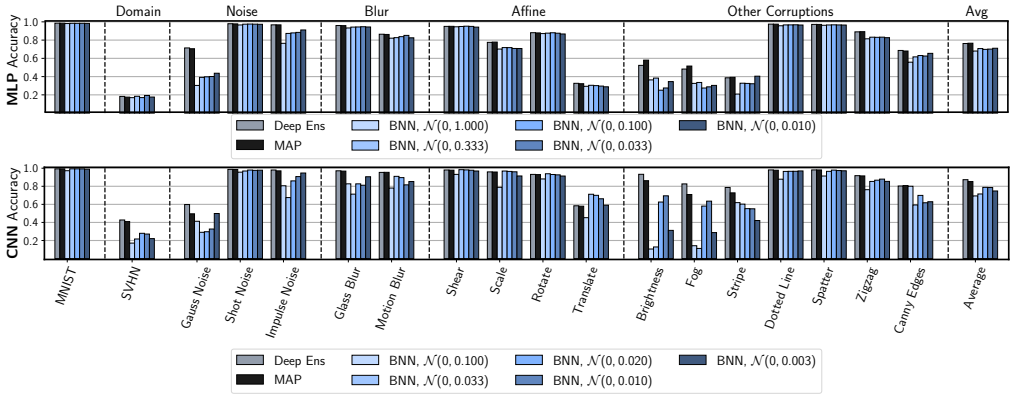


(b) Corruption Intensity 3

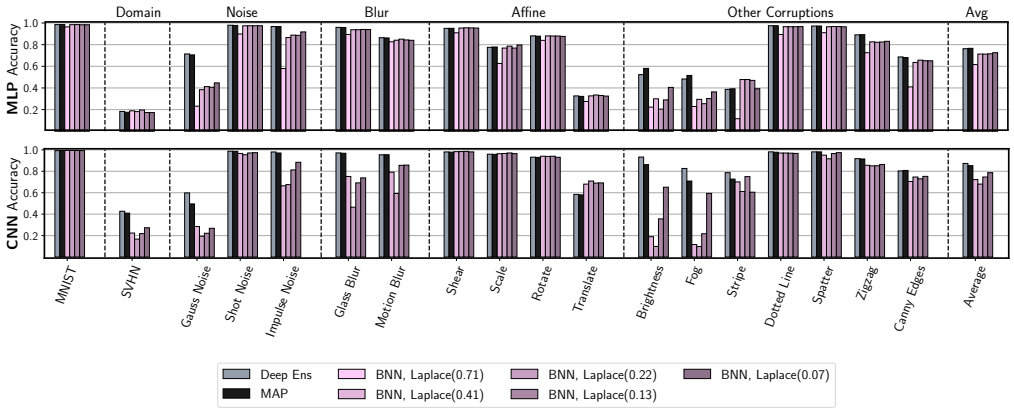


(c) Corruption Intensity 5

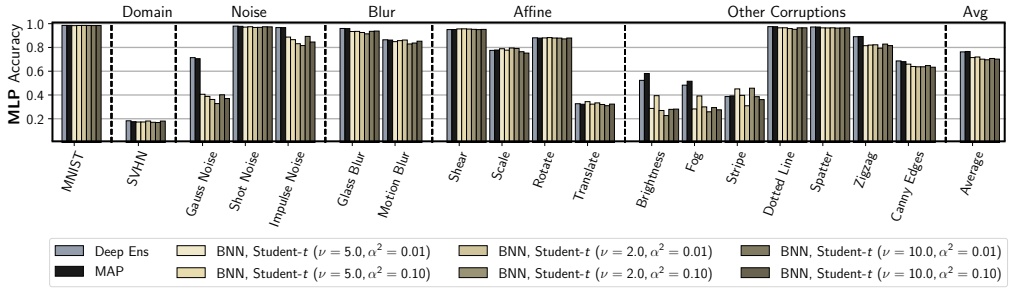
Figure 6: **Detailed results on CIFAR-10.** Accuracy, log-likelihood and log-likelihood for deep ensembles, MAP solution, and BNN variants under covariate shift on CIFAR-10. We report the performance at corruption intensity levels 1, 3 and 5 (corruption intensity does not affect the CIFAR-10 and STL-10 columns in the plots). For all methods except deep ensembles we report the mean and standard deviation (via error-bars) over 3 random seeds. *EmpCov* priors provide the only BNN variation that consistently performs on par with deep ensembles in terms of log-likelihood and ECE. Tempered posteriors improve the accuracy on some of the corruptions, but significantly hurt in-domain performance.



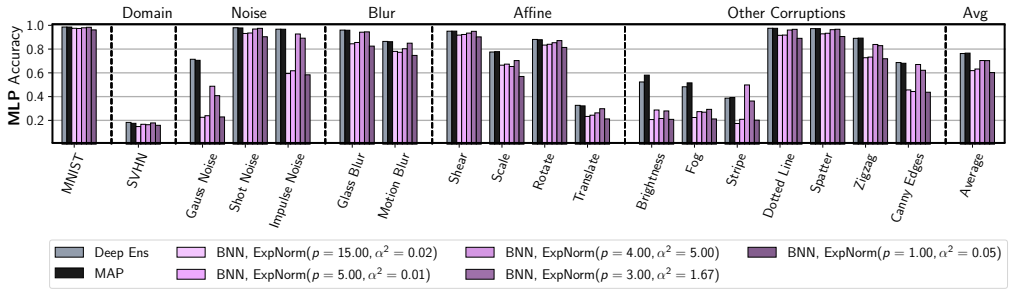
(a) Gaussian priors



(b) Laplace priors



(c) Student-t priors



(d) ExpNorm priors

Figure 7: **Priors on MNIST.** We report the performance of different prior families under covariate shift on MNIST. For Gaussian and Laplace prior families we report the results using both the MLP and CNN architectures; for Student-t and ExpNorm we only report the results for MLP. None of the priors can match the MAP performance across the board, with particularly poor results under *Gaussian noise*, *Brightness* and *Fog* corruptions.

Table 2: **Spurious correlations.** Accuracy and log-likelihood of MAP, deep ensembles and BNNs with Gaussian and *EmpCov* priors on the Shift-MNIST dataset.

Model	MLP Accuracy	MLP LL	CNN Accuracy	CNN LL
MAP	88.70%	-0.527	48.63%	-2.206
Deep Ensemble	88.73%	-0.527	72.99%	-1.041
BNN Gaussian Prior	90.83%	-0.598	64.27%	-1.326
BNN EmpCov Prior	86.95%	-1.146	64.41%	-1.450

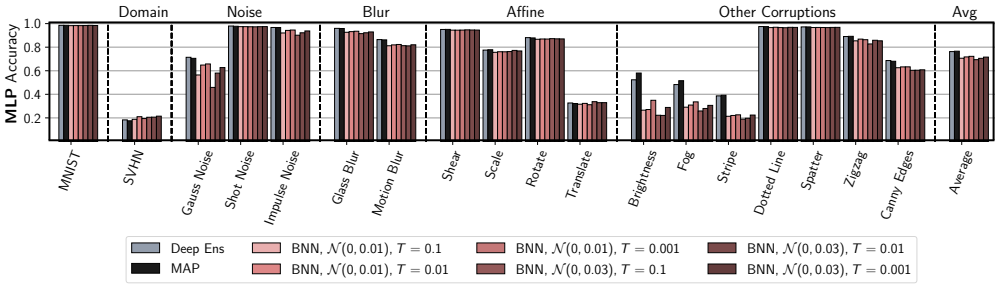


Figure 8: **Temperature ablation.** We report the performance of BNNs with Gaussian priors and tempered posteriors for different temperatures and prior scales. Low temperatures ($T = 10^{-2}$, 10^{-3}) can provide a significant improvement on the *noise* corruptions, but do not improve the results significantly under other corruptions.

the activated (spurious) pixels. When those same pixels are not activated in the test set, such samples will have a larger shift in their predictions.

An in-depth analysis of BNNs in the presence of spurious correlations remains an exciting direction for further research.

D Tempered Posteriors

In this section we explore the effect of posterior tempering on the performance of the MLP on MNIST. In particular, following Wenzel et al. [54] we consider the cold posteriors:

$$p_T(W|D) \propto (p(D|W)p(W))^{1/T}, \tag{4}$$

where $T \leq 1$. In Figure 8 we report the results for BNNs with Gaussian priors with variances 0.01 and 0.03 and posterior temperatures $T \in \{10^{-1}, 10^{-2}, 10^{-3}\}$. As observed by Izmailov et al. [22], lower temperatures $(10^{-2}, 10^{-3})$ improve performance under the *Gaussian noise* corruption; however, low temperatures do not help with other corruptions significantly.

E Proofs of the theoretical results

For convenience, in this section we assume that a constant value of 1 is appended to the input features instead of explicitly modeling a bias vector b. We assume that the output $f(x, W)$ of the network with parameters W on an input x is given by

$$f(x, W) = \psi(\phi(\dots\phi(\phi(xW^1)W^2 + b^2))W^l + b^l), \tag{5}$$

where ϕ are non-linearities of the intermediate layers (e.g. ReLU) and ψ is the final link function (e.g. softmax).

We will also assume that the likelihood is a function $\ell(\cdot, \cdot)$ that only depends on the output of the network and the target label:

$$p(y|x, W) = \ell(y, f(x, W)). \tag{6}$$

For example, in classification $\ell(y, f(x, W)) = f(x, W)[y]$, the component of the output of the softmax layer corresponding to the class label y. Finally, we assume that the likelihood factorizes over the inputs:

$$p(\mathcal{D}|W) = \prod_{x,y \in \mathcal{D}} p(y|x, W) \tag{7}$$

for any collection of datapoints $\mathcal{D}$.

E.1 Proof of Lemma 1

We restate the Lemma:

Lemma 1 *Suppose that the input feature x_k^i is equal to zero for all the examples x_k in the training dataset D. Suppose the prior distribution over the parameters $p(W)$ factorizes as $p(W) = p(w_{ij}^1) \cdot p(W \setminus w_{ij}^1)$ for some neuron j in the first layer, where $W \setminus w_{ij}^1$ represents all the parameters W of the network except w_{ij}^1. Then, the posterior distribution $p(W|D)$ will also factorize and the marginal posterior over the parameter w_{ij}^1 will coincide with the prior:*

$$p(W|D) = p(W \setminus w_{ij}^1|D) \cdot p(w_{ij}^1). \tag{8}$$

Consequently, the MAP solution will set the weight w_{ij}^1 to the value with maximum prior density.

Proof. Let us denote the input vector x without the input feature i by x^{-i}, and the matrix W^1 without the row i by W_{-i}^1. We can rewrite Equation 5 as follows:

$$f(x, W) = \psi(\phi(\dots \phi(\phi(x^{-i} W_{-i}^1 + \underbrace{x^i W_i^1}_{=0}) W^2 + b^2)) W^l + b^l). \tag{9}$$

As for all the training inputs x_k the feature x_k^i is equal to 0, the vector $x^i W_i^1$ is equal to zero and can be dropped:

$$f(x_k, W) = \psi(\phi(\dots \phi(\phi(x_k^{-i} W_{-i}^1) W^2 + b^2)) W^l + b^l)) =: f'(x_k, W_{-i}), \tag{10}$$

where W_{-i} denotes the vector of parameters W without W_i^1, and we defined a new function f' that does not depend on W_i and is equivalent to f on the training data. Consequently, according to Equation 6 and Equation 7, we can write

$$p(D|W) = \prod_{k=1}^{n} \ell(y_k, f(x_k, W)) = \prod_{k=1}^{n} \ell(y_k, f'(x_k, W_{-i})). \tag{11}$$

In other words, the likelihood does not depend on W_i^1 and in particular w_{ij}^1 for any j.

Let us write down the posterior over the parameters using the factorization of the prior:

$$p(W|D) = \frac{\overbrace{p(D|W \setminus w_{ij}^1) p(W \setminus w_{ij}^1)}^{\text{does not depend on } w_{ij}^1} p(w_{ij}^1)}{Z}, \tag{12}$$

where Z is a normalizing constant that does not depend on W. Hence, the posterior factorizes as a product of two distributions: $p(D|W \setminus w_{ij}^1) p(W \setminus w_{ij}^1)/Z$ over $W \setminus w_{ij}^1$ and $p(w_{ij}^1)$. The

marginal posterior over w_{ij}^1 thus coincides with the prior and is independent of the other parameters.

Maximizing the factorized posterior Equation 12 to find the MAP solution, we set the w_{ij}^1 to the maximum of its marginal posterior, as it is independent of the other parameters. ■

E.2 Formal statement and proof of Proposition 1

First, let us prove the following result for the MAP solution.

Proposition 1' *Consider the following assumptions:*

(a) The input feature x_k^i is equal to zero for all the examples x_k in the training dataset D.

(b) The prior over the parameters factorizes as $p(W) = p(W_{-i}) \cdot p(W_i^1)$, where W_{-i} is the vector of all parameters except for W_i^1, the row i of the weight matrix W^1 of the first layer.

(c) The prior distribution $p(W_i^1)$ has maximum density at 0.

Consider an input $x(c) = [x^1, \ldots, x^{i-1}, c, x^{i+1}, \ldots, x^m]$. Then, the prediction with the MAP model W_{MAP} does not depend on c: $f(x(c), W_{MAP}) = f(x(0), W_{MAP})$.

Proof. Analogous to the proof of Lemma 1, we can show that under the assumptions (a), (b) the posterior over the parameters factorizes as

$$p(W|D) = p(W^{-i}|D)p(W_i^1). \tag{13}$$

Then, the MAP solution will set the weights W_i^1 to the point of maximum density, which is 0 under assumption (c). Consequently, based on Equation 9, we can see that the output of the MAP model will not depend on $x^i = c$. ■

Next, we provide results for the Bayesian model average. We define *positive-homogeneous* activations as functions ϕ that satisfy $\phi(c \cdot x) = c \cdot \phi(x)$ for any positive scalar c and any x. For example, ReLU and Leaky ReLU activations are positive-homogeneous.

We will call a vector z of class logits (inputs to softmax) ϵ-*separable* if the largest component z_i is larger than all the other components by at least ϵ:

$$z_i - z_j > \epsilon \quad \forall j \neq i. \tag{14}$$

We can prove the following general proposition.

Proposition 1" *We will need the following assumptions:*

(d) The support of the prior over the parameters W_{-i} is bounded: $\|W_{-i}\| < B$.

(e) The activations ϕ are positive-homogeneous and have a Lipschitz constant bounded by L_ϕ.

Consider an input $x(c) = [x^1, \ldots, x^{i-1}, c, x^{i+1}, \ldots, x^m]$. Then, we can prove the following conclusions

(2) Suppose the link function ψ is identity. Suppose also that the expectation $\mathbb{E}[\phi(\ldots\phi(\phi(W_i^1)W^2)\ldots)W^l]$ over W sampled from the posterior is non-zero. Then the predictive mean under BMA (see Equation 2) on the input $x(c)$ depends on c.

(3) Suppose the link function ψ is softmax. Then, for sufficiently large $c > 0$ the predicted class $\hat{y}(c) = \arg\max_y f(x(c), W)[y]$ does not depend on $x(c)$ for any sample W from the posterior such that $z = \phi(\ldots\phi(\phi(W_i^1)W^2)\ldots)W^l$ is ϵ-separable.

Proof. We can rewrite Equation 5 as follows:

$$\begin{aligned} f(x(c), W) = \psi(\phi(\dots\phi(\phi(x^{-i}W^1_{-i} + cW^1_i)W^2 + b^2))W^l + b^l) = \\ \psi(\phi(\dots\phi(c\cdot\phi([x^{-i}W^1_{-i}]/c + W^1_i)W^2 + b^2))W^l + b^l) = \\ \psi(c\cdot(\phi(\dots\phi(\phi([x^{-i}W^1_{-i}]/c + W^1_i)W^2 + b^2/c))W^l + b^l/c)). \end{aligned} \tag{15}$$

Now, under our assumptions the prior and hence the posterior over the weights W_{-i} is bounded. As in finite-dimensional Euclidean spaces all norms are equivalent, in particular we imply that (1) the spectral norms $\|W^t\|_2 < L_W$ are bounded for all layers $t = 2, \dots, l$ by a constant L_W, and (2) the Frobenious norms $\|\cdot\|$ of the bias parameters b_t and the weights W^1_{-i} are all bounded by a constant B. We will also assume that the norm of the vector x^{-i} is bounded by the same constant: $\|x^{-i}\| \leq B$.

Consider the difference

$$\begin{aligned} \left\| \left(\phi\left(\dots\phi\left(\phi\left(\frac{x^{-i}W^1_{-i}}{c} + W^1_i\right)W^2 + \frac{b^2}{c}\right)\right)W^l + \frac{b^l}{c}\right) - \right. \\ \left. \phi\left(\dots\phi\left(\phi\left(\frac{x^{-i}W^1_{-i}}{c} + W^1_i\right)W^2 + \frac{b^2}{c}\right)\right)W^l \right\| \leq \frac{B}{c}. \end{aligned} \tag{16}$$

Indeed, by the $\|b^l\| \leq B$. Next, for an arbitrary z we can bound

$$\left\| \phi\left(z + \frac{b^{l-1}}{c}\right)W^l - \phi(z)W^l \right\| \leq L_W \cdot L_\phi \cdot \frac{B}{c}, \tag{17}$$

where we used the fact that ϕ is Lipschitz with L_ϕ and the Lipschitz constant for matrix multiplication by W^l coincides with the spectral norm of W^l which is bounded by L_W.

Using the bound in Equation 17, we have

$$\begin{aligned} \left\| \left(\phi\left(\phi\left(\dots\phi\left(\phi\left(\frac{x^{-i}W^1_{-i}}{c} + W^1_i\right)W^2 + \frac{b^2}{c}\right)W^{l-1} + \frac{b^{l-1}}{c}\right)\dots\right)W^l + \frac{b^l}{c}\right) - \right. \\ \left. \phi\left(\phi\left(\dots\phi\left(\phi\left(\frac{x^{-i}W^1_{-i}}{c} + W^1_i\right)W^2 + \frac{b^2}{c}\right)\cdots\right)W^{l-1}\right)W^l \right\| \leq \frac{B}{c} + L_W \cdot L_\phi \cdot \frac{B}{c}. \end{aligned} \tag{18}$$

Applying the same argument to all layers of the network (including the first layer where $\frac{x^{-i}W^1_{-i}}{c}$ plays the role analogous to $\frac{b^{l-1}}{c}$ in Equation 17), we get

$$\begin{aligned} \left\| \left(\phi\left(\dots\phi\left(\phi\left(\frac{x^{-i}W^1_{-i}}{c} + W^1_i\right)W^2 + \frac{b^2}{c}\right)\dots\right)W^l + \frac{b^l}{c}\right) \right. \\ \left. -\phi\left(\dots\phi\left(\phi\left(W^1_i\right)W^2\right)\dots\right)W^l \right\| \\ \leq \frac{B}{c}(1 + L_W \cdot L_\phi + L^2_W \cdot L^2_\phi + \dots + L^{l-1}_W \cdot L^{l-1}_\phi). \end{aligned} \tag{19}$$

Choosing c to be sufficiently large, we can make the bound in Equation 19 arbitrarily tight.

Conclusion (2) Suppose ψ is the identity. Then, we can write

$$f(x(c), W) = c\cdot\phi\left(\dots\phi\left(\phi\left(W^1_i\right)W^2\right)\dots\right)W^l + \Delta, \tag{20}$$

where Δ is bounded: $\|\Delta\| \leq B(1 + L_W \cdot L_\phi + L^2_W \cdot L^2_\phi + \dots + L^{l-1}_W \cdot L^{l-1}_\phi)$. Consider the predictive mean under BMA,

$$\mathbb{E}_W f(x(c), W) = c\cdot\underbrace{\mathbb{E}_W\phi\left(\dots\phi\left(\phi\left(W^1_i\right)W^2\right)\dots\right)W^l}_{\neq 0} + \underbrace{\mathbb{E}_W\Delta}_{\text{Bounded}}, \tag{21}$$

where the first term is linear in c and the second term is bounded uniformly for all c. Finally, we assumed that the expectation $\mathbb{E}_W \phi\left(\ldots \phi\left(\phi\left(W_i^1\right) W^2\right)\ldots\right) W^l \neq 0$, so for large values of c the first term in Equation 21 will dominate, so the output depends on c.

Conclusion (3) Now, consider the softmax link function ψ. Note that for the softmax we have $\arg\max_y \psi(c \cdot z)[y] = \arg\max_y z[y]$. In other words, multiplying the logits (inputs to the softmax) by a positive constant c does not change the predicted class. So, we have

$$\hat{y}(c, W) = \arg\max_y f(x(c), W)[y] = \arg\max_y \left(\phi\left(\ldots\phi\left(\phi\left(\frac{x^{-i}W_{-i}^1}{c} + W_i^1\right)W^2 + \frac{b^2}{c}\right)\right)W^l + \frac{b^l}{c}\right)[y]. \tag{22}$$

Notice that $z_W = \phi(\ldots\phi(\phi(W_i^1)W^2)\ldots)W^l$ does not depend on the input $x(c)$ in any way. Furthermore, if z_W is ϵ-separable, with class y_W corresponding to the largest component of z_W, then by taking

$$c > \frac{B(1 + L_W \cdot L_\phi + L_W^2 \cdot L_\phi^2 + \ldots + L_W^{l-1} \cdot L_\phi^{l-1})}{\epsilon}, \tag{23}$$

we can guarantee that the predicted class for $f(x(c), W)$ will be y_W according to Equation 19. ■

E.3 General linear dependencies, Proposition 2

We will prove the following proposition, reducing the case of general linear dependencies to the case when an input feature is constant.

Suppose that the prior over the weights W^1 in the first layer is an i.i.d. Gaussian distribution $\mathcal{N}(0, \alpha^2)$, independent of the other parameters in the model. Suppose all the inputs $x_1 \ldots x_n$ in the training dataset D lie in a subspace of the input space: $x_i^T c = 0$ for all $i = 1, \ldots, n$ and some constant vector c such that $\sum_{i=1}^m c_i^2 = 1$.

Let us introduce a new basis $v_1, \ldots, v_m$ in the input space, such that the vector c is the first basis vector. We can do so e.g. by starting with the collection of vectors $\{c, e_2, \ldots, e_m\}$, where e_i are the standard basis vectors in the feature space, and using the Gram–Schmidt process to orthogonalize the vectors. We will use V to denote the matrix with vectors $v_1, \ldots, v_m$ as columns. Due to orthogonality, we have $VV^T = I$.

We can rewrite our model from Equation 5 as

$$f(x, W) = \psi(\phi(\ldots\phi(\phi(\underbrace{xV}_{\bar{x}}\underbrace{V^T W^1}_{\bar{W}^1})W^2 + b^2))W^l + b^l). \tag{24}$$

We can thus re-parameterize the first layer of the model by using transformed inputs $\bar{x} = xV$, and transformed weights $\bar{W}^1 = V^T W^1$. Notice that this re-parameterized model is equivalent to the original model, and doing inference in the re-parameterized model is equivalent to doing inference in the original model.

The induced prior over the weights $\bar{W}^1$ is $\mathcal{N}(0, \alpha^2 I)$, as we simply rotated the basis. Furthermore, the input $\bar{x}_k^1 = x_k^T v_1 = 0$ for all training inputs k. Thus, with the re-parameterized model we are in the setting of Lemma 1 and Propositions 1', 1".

In particular, the posterior over the parameters $\bar{W}_1^1 = v_1^T W^1$ will coincide with the prior $\mathcal{N}(0, \alpha^2 I)$ (Lemma 1). The MAP solution will ignore the feature combination $\bar{x}^1 = x^T v_1$, while the BMA predictions will depend on it (Propositions 1', 1").

E.4 Convolutional layers, Proposition 3

Suppose that the convolutional filters in the first layer are of size $K \times K \times C$, where C is the number of input channels. Let us consider the set $\hat{D}$ of size N of all the patches of size $K \times K \times C$ extracted from the training images in D after applying the same padding as in the first convolutional layer. Let us also denote the set of patches extracted from a fixed input image by D_x.

A convolutional layer applied to x can be thought of as a fully-connected layer applied to all patches in D_x individually, and with results concatenated:

$$conv(w, x) = \left\{ \left(i, j, \sum_{a=i}^{i+k} \sum_{b=j}^{j+k} \sum_{c=1}^{C} x_{a,b,c} \cdot W^1_{a,b,c} \right) \right\}, \tag{25}$$

where $x_{a,b,c}$ is the intensity of the image at location (a, b) in channel c, $W^1_{a,b,c}$ is the corresponding weight in the convolutional filter, and the tuples (i, j, v) for all i, j represent the intensities at location (i, j) in the output image.

In complete analogy with Lemma 1 and Propositions 1', 1", we can show that if all the patches in the dataset $\hat{D}$ are linearly dependent, then we can re-parameterize the convolutional layer so that one of the convolutional weights will always be multiplied by 0 and will not affect the likelihood of the data. The MAP solution will set this weight to zero, while the BMA will sample this weight from the prior, and it will affect predictions.

F How corruptions break linear dependence in the data

In Figure 9, we visualize the projections of the original and corrupted MNIST data on the PCA components extracted from the MNIST train set and the set of all 5×5 patches of the MNIST train set. As we have seen in section 5, the former are important for the MLP robustness, while the latter are important for CNNs.

Certain corruptions increase variance along the lower PC directions more than others. For example, the *Translate* corruption does not alter the principal components of the 5×5 patches in the images, and so a convolutional BNN with a Gaussian prior is very robust to this corruption. In contrast, Gaussian noise increases variance similarly along all directions, breaking any linear dependencies present in the training data and resulting in much worse BNN performance.

G Analyzing other approximate inference methods

In this section, we provide additional discussion on why popular approximate inference methods SWAG and MC Dropout do not exhibit the same poor performance under covariate shift.

G.1 SWAG

SWA-Gaussian (SWAG) [35] approximates the posterior distribution as a multivariate Gaussian with the SWA solution [21] as its mean. To construct the covariance matrix of this posterior, either the second moment (SWAG-Diagonal) or the sample covariance matrix of the SGD iterates is used. For any linear dependencies in the training data, the corresponding combinations of weights become closer to zero in later SGD iterates due to weight decay. Since SWAG only uses the last K iterates in constructing its posterior, the resulting posterior will likely have very low variance in the directions of any linear dependencies. Furthermore, because SGD is often initialized at low magnitude weights, even the earlier iterates will likely have weights close to zero in these directions.

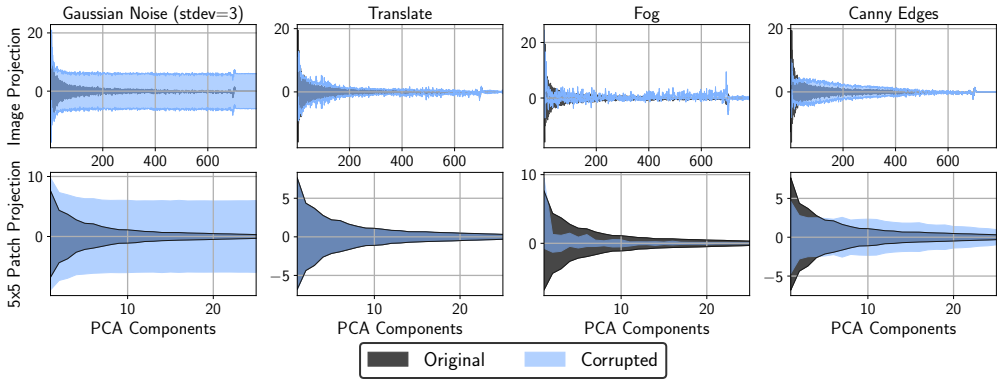


Figure 9: **Corruptions and linear dependence. Top**: The distribution (mean $\pm$ 2 std) of MNIST and MNIST-C images and **bottom**: 5×5 patches extracted from these images projected onto the corresponding principal components of the training data images and patches. *Gaussian noise* corruption breaks linear dependencies in both cases, while *Translate* does not change the projection distribution for the 5×5 patches.

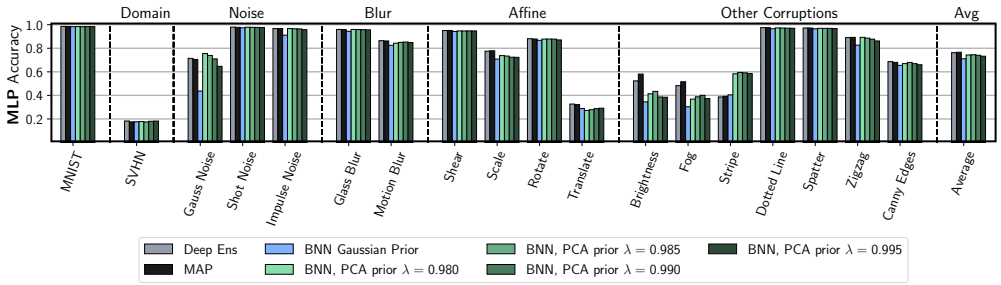


Figure 10: **General PCA priors.** Performance of the PCA priors introduced in Appendix H for various decay rates λ. PCA priors generally improve performance significantly under *Gaussian noise* and *Stripe*. Lower decay rates λ provide better results under *Gaussian noise*.

G.2 MC Dropout

MC Dropout applies dropout at both train and test time, thus allowing computation of model uncertainty from a single network by treating stochastic forward passes through the network as posterior samples. The full model learned at train time is still an approximate MAP solution, and thus will be minimally affected by linear dependencies in the data being broken at test time. As for the test-time dropout, we can conclude that if the expected output of the network is not affected by linear dependencies being broken, then any subset of that network (containing a subset of the network's hidden units) would be similarly unaffected. Additionally, if dropping an input breaks a linear dependency from the training data, the network (as an approximate MAP solution) is robust to such a shift.

H General PCA Priors

In section 6 we introduced the *EmpCov* prior, which improves robustness to covariate shift by aligning with the training dataset's principal components. Following the notation used in section 6, we can define a more general family of PCA priors as

$$p(w^1) = \mathcal{N}(0, \alpha V \text{diag}(s) V^T + \epsilon I), \quad s_i = f(i) \tag{26}$$

where for an architecture with n_w first layer weights, s is a length n_w vector, diag(s) is the $n_w \times n_w$ diagonal matrix with s as its diagonal, and V is an $n_w \times n_w$ matrix such that the i^{th} column of V is the i^{th} eigenvector of Σ.

The *EmpCov* prior is the PCA prior where $f(i)$ returns the i^{th} eigenvalue (explained variance) of Σ. However, there might be cases where we do not want to directly use the empirical covariance, and instead use an alternate f. For example, in a dataset of digits written on a variety of different wallpapers, the eigenvalues for principal components corresponding to the wallpaper pattern could be much higher than those corresponding to the digit. If the task is to identify the digit, using *EmpCov* might be too restrictive on digit-related features relative to wallpaper-related features.

We examine alternative PCA priors where $f(i) = \lambda^i$ for different decay rates λ. We evaluate BNNs with these priors on MNIST-C, and find that the choice of decay rate can significantly alter the performance on various corruptions. Using priors with faster decay rates (smaller λ) can provide noticeable improvement on Gaussian Noise and Zigzag corruptions, while the opposite occurs in corruptions like Translate and Fog. Connecting this result back to Appendix F and Figure 9, we see that the corruptions where faster decay rates improve performance are often the ones which add more noise along the smallest principal components.

I Effect of non-zero mean corruptions

As we have seen in various experiments (e.g. Figure 2, Figure 7), convolutional Bayesian neural networks are particularly susceptible to the *brightness* and *fog* corruptions on MNIST-C. Both of these corruptions are not zero-mean: they shift the average value of the input features by 1.44 and 0.89 standard deviations respectively. In order to understand why non-zero mean corruptions can be problematic, let us consider a simplified corruption that applies a constant shift c to all the pixels in the input image. Ignoring the boundary effects, the convolutional layers are linear in their input. Denoting the output of the convolution with a filter w on an input x as $conv(w, x)$, and an image with all pixels equal to 1 as $\mathbb{1}$ we can write

$$conv(w, x + c \cdot \mathbb{1}) = conv(w, x) + c \cdot conv(w, \mathbb{1}) = conv(w, x) + \mathbb{1} \cdot c \cdot \sum_{a,b} w_{a,b}, \tag{27}$$

where the last term represents an image of the same size as the output of the $conv(\cdot, \cdot)$ but with all pixels equal to the sum of the weights in the convolutional filter w multiplied by c. So, if the input of the convolution is shifted by a constant value c, the output will be shifted by a constant value $c \cdot \sum_{a,b} w_{a,b}$.

As the convolutional layer is typically followed by an activation such as ReLU, the shift in the output of the convolution can significantly hinder the performance of the network. For example, suppose $c \cdot \sum_{a,b} w_{a,b}$ is a negative value such that all the output pixels in $conv(w, x) + \mathbb{1} \cdot c \cdot \sum_{a,b} w_{a,b}$ are negative. In this case, the output of the ReLU activation applied after the convolutional filter will be 0 at all output locations, making it impossible to use the learned features to make predictions.

In the next section, we propose a prior that reduces the sum $\sum_{a,b} w_{a,b}$ of the filter weights, and show that it significantly improves robustness to multiple corruptions, including *fog* and *brightness*.

I.1 SumFilter prior

As we've discussed, if the sum of filter weights for CNNs is zero, then corrupting the input by adding a constant has no effect on our predictions. We use this insight to propose a novel

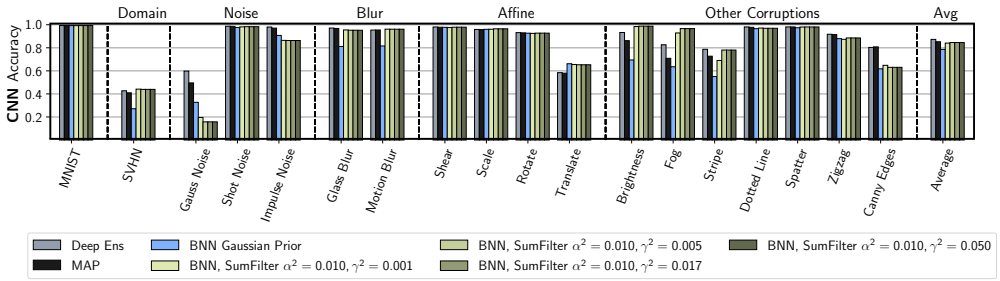


Figure 11: **SumFilter priors.** Performance of BNNs with the *SumFilter* priors introduced in subsection I.1 for the CNN architecture on MNIST. *SumFilter* priors do not improve the performance under *Gaussian noise* unlike *EmpCov priors*, but provide a significant improvement on the *Brightness* and *Fog* corruptions.

prior that constrains the sum of the filter weights. More specifically, we place a Gaussian prior on the parameters and Laplace prior on the sum of the weights:

$$p(w) \sim \mathcal{N}\left(w|0, \alpha^2 I\right) \times \text{Laplace}\Big(\sum_{\text{filter}} w|0, \gamma^2\Big). \tag{28}$$

For our experiments, we only place the additional Laplace prior on the sum of weights in first layer filters. An alternative version could place the prior over filter sums in subsequent layers, which may be useful for deeper networks.

I.2 Experiments

Figure 11 shows that this prior substantially improves the performance of a convolutional BNN on MNIST-C. The BNN with a filter sum prior yields a better or comparable performance to MAP for all MNIST corruptions, with the exception of *Canny Edges* and *Impulse Noise*. We also implemented this prior for MLPs, but found that it only improved BNN performance on two corruptions, fog and brightness. Overall, this prior addresses a more specific issue than *EmpCov*, and we would not expect it to be applicable to as many forms of covariate shift.

J Example: Bayesian NALU under covariate shift

The Neural Arithmetic Logic Unit (NALU) [53] is an architecture which can learn arithmetic functions that extrapolate to values outside those observed during training. A portion of the unit is of the form $\prod_{j=1}^{m} |x_j|^{w_j}$, and in this section we examine a simplified form of this unit in order to demonstrate an instance where nonlinear dependencies hurt BMA under covariate shift.

Let's consider the NALU-inspired architecture with input features $x^1, \ldots, x^m$ that takes the form $f(x, w) = \prod_{j=1}^{m} (x^j)^{w_j}$. Suppose the prior over the weights $w = [w_1, \ldots, w_m]$ is an i.i.d. Gaussian distribution $\mathcal{N}(0, \alpha^2)$. Suppose all inputs $x_1, \ldots x_n$ in training dataset $\mathcal{D}$ lie in a subspace of the input space: $\prod_{j=1}^{m} (x_i^j)^{p_j} = 1$ for all $i = 1, \ldots, n$ and some constant vector p such that $\sum_{j=1}^{m} p_j^2 = 1$. Following the same approach as subsection E.3, we can introduce a new basis $v_1, \ldots, v_m$ in the input space such that $v_1 = p$. We can similarly re-parameterize the model using the weights rotated into this new basis, $\bar{w} = w^T v_1, \ldots, w^T v_m$, and it follows

that $w_i = \bar{w}_1 \cdot v_1^i + \cdots + \bar{w}_m \cdot v_m^i$ for all $i = 1, \ldots, n$. Using the corresponding transformed inputs $\bar{x}^i = \prod_{j=1}^m (x^j)^{v_i^j}$ for all $i = 1, \ldots, n$, we can rewrite our model as follows:

$$f(x, w) = f(\bar{x}, \bar{w}) = (\prod_{j=2}^{m} (\bar{x}^j)^{\bar{w}_j}) \cdot \underbrace{(\bar{x}^1)^{\bar{w}_1}}_{=1}. \tag{29}$$

Since $f(\bar{x}, \bar{w})$ does not depend on $\bar{w}_1$ for all $\bar{x} \in \bar{\mathcal{D}}$, we can follow the same reasoning from subsection E.1 to conclude that the marginal posterior over $\bar{w}_1$ coincides with the induced prior. Since $\bar{w}$ is the result of simply rotating w into a new basis, it also follows that the induced prior over $\bar{w}$ is $\mathcal{N}(0, \alpha^2 I)$, and that the posterior can be factorized as $p(\bar{w}|\bar{\mathcal{D}}) = p(\bar{w} \setminus \bar{w}_1|\bar{\mathcal{D}}) \cdot p(\bar{w}_1)$.

Consider a test input $\bar{x}_k(c) = [c, \bar{x}_k^2, \ldots, \bar{x}_k^m]$. The predictive mean under BMA will be:

$$\mathbb{E}_{\bar{w}} f(\bar{x}_k(c), \bar{w}) = \mathbb{E}_{\bar{w} \setminus \bar{w}_1} \prod_{j=2}^{m} (\bar{x}_k^j)^{\bar{w}_j} \cdot \mathbb{E}_{\bar{w}_1} c^{\bar{w}_1}. \tag{30}$$

Thus the predictive mean depends upon c, and so the BMA will not be robust to the nonlinear dependency being broken at test time. In comparison, the MAP solution would set $\bar{w}_1 = 0$, and its predictions would not be affected by c.

While the dependency described in this section may not necessarily be common in real datasets, we highlight this example to demonstrate how a nonlinear dependency can still hurt BMA robustness. This further demonstrates how the BMA issue we've identified does not only involve *linear* dependencies, but rather involves dependencies which have some relationship to the model architecture.

K Dead neurons

Neural network models can often contain *dead neurons*: hidden units which output zero for all inputs in the training set. This behaviour occurs in classical training when a neuron is knocked off the training data manifold, resulting in zero non-regularized gradients for the corresponding weights and thus an inability to train the neuron using the gradient signal from the non-regularized loss. However, we can envision scenarios where a significant portion of the BNN posterior distribution contains models with dead neurons, such as when using very deep, overparameterized architectures.

Let us consider the posterior distribution over the parameters W conditioned on the parameters $W^1, W^2, b^2, \ldots, W^k, b^k$ of the first k layers, where we use the notation of Appendix E. Suppose for the parameters $W^1, W^2, b^2, \ldots, W^k, b^k$ the k-th layer contains a dead neuron, i.e. an output that is 0 for all the inputs x_j in the training dataset D. Then, consider the sub-network containing layers $k+1, \ldots, l$. For this sub-network, the output of a dead neuron in the k-th layer is an input that is 0 for all training inputs. We can then apply the same reasoning as we did in subsection E.1, subsection E.2 to show that there will exist a direction in the parameters W^k of the $k+1$-st layer, such that along this direction the posterior *conditioned* on the parameters $W^1, W^2, b^2, \ldots, W^k, b^k$ coincides with the prior (under the assumption that the prior over the parameters W^k is iid and independent of the other parameters). If a test input is corrupted in a way that activates the dead neuron, the predictive distribution of the BMA *conditioned* on the parameters $W^1, W^2, b^2, \ldots, W^k, b^k$ will change.

L Bayesian linear regression under covariate shift

We examine the case of Bayesian linear regression under covariate shift. Let us define the following Bayesian linear regression model:

$$y = w^\top \phi(x, z) + \epsilon(x) \tag{31}$$
$$\epsilon \sim \mathcal{N}(0, \sigma^2) \tag{32}$$

where $w \in \mathbb{R}^d$ are linear weights and z are the deterministic parameters of the basis function ϕ. We consider the dataset $D = \{(x_i, y_i)\}_{i=1}^n$ and define $y := (y_1, \ldots, y_N)^\top$, $X := (x_1, \ldots, x_n)^\top$, and $\Phi := (\phi(x_1, z), \ldots, \phi(x_n, z))^\top$

The likelihood function is given by:

$$p(y|X, w, \sigma^2) = \prod_{i=1}^{n} \mathcal{N}(y_i|w^\top \phi(x_i, z), \sigma^2). \tag{33}$$

Let us choose a conjugate prior on the weights:

$$p(w) = \mathcal{N}(w|\mu_0, \Sigma_0), \tag{34}$$

The posterior distribution is given by:

$$\begin{aligned} p(w|\mathcal{D}) &\propto \mathcal{N}(w|\mu_0, \Sigma_0) \times \prod_{i=1}^{n} \mathcal{N}(y_i|w^\top \phi(x_i, z), \sigma^2) \\ &= \mathcal{N}(w|\mu, \Sigma), \\ \mu &= \Sigma \left(\Sigma_0^{-1}\mu_0 + \frac{1}{\sigma^2}\Phi^\top y \right), \\ \Sigma^{-1} &= \Sigma_0^{-1} + \frac{1}{\sigma^2}\Phi^\top \Phi. \end{aligned}$$

The MAP solution is therefore equal to the mean,

$$w_{\text{MAP}} = \Sigma \left(\Sigma_0^{-1}\mu_0 + \frac{1}{\sigma^2}\Phi^\top y \right) = \left(\Sigma_0^{-1} + \frac{1}{\sigma^2}\Phi^\top \Phi \right)^{-1} \left(\Sigma_0^{-1}\mu_0 + \frac{1}{\sigma^2}\Phi^\top y \right) \tag{35}$$

Thus, we see that the BMA and MAP predictions coincide in Bayesian linear regression, and both will have equivalent performance under covariate shift in terms of accuracy.

What happens away from the data distribution? If the data distribution spans the entire input space, than the posterior will contract in every direction in the weight space. However, if the data lies in a linear (or affine, if we are using a Gaussian prior) subspace of the input space, there will be directions in the parameter space for which the posterior would coincide with the prior. Now, if a test input does not lie in the same subspace, the predictions on that input would be affected by the shift vector according to the prior. Specifically, if the input x is shifted from the subspace containing the data by a vector v orthogonal to the subspace, then the predictions between x and its projection to the subspace would differ by $w^T v$, where $w \sim \mathcal{N}(\mu_0, \Sigma_0)$, which is itself $\mathcal{N}(\mu_0^T v, v^T \Sigma_0 v)$. Assuming the prior is zero-mean, the mean of the prediction would not be affected by the shift, but the uncertainty will be highly affected. The MAP solution on the other hand does not model uncertainty.

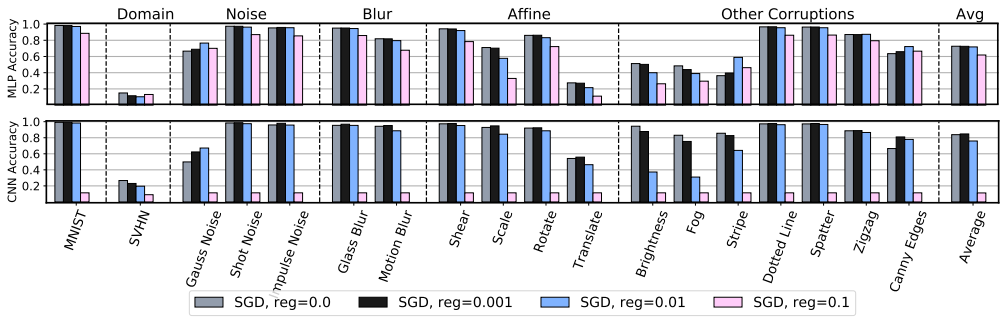


Figure 12: **Effect of regularization on SGD's performance on corrupted MNIST.** Accuracy for the following values of the regularization parameter: 0.0, 0.001, 0.01, and 0.1. **Top**: Fully-connected network; **bottom**: Convolutional neural network. Regularization helps improve the performance on some corruptions, such as Gaussian noise, but its absence does not affect SGD's robustness under covariate shift because the weights are initialized at small values.

M An optimization perspective on the covariate shift problem.

In this section, we examine SGD's robustness to covariate shift from an optimization perspective.

M.1 Effect of regularization and initialization on SGD's robustness to covariate shift

In Section K, we discussed how SGD pushes the weights that correspond to dead neurons, a generalization of the dead pixels analysis, towards zero thanks to the regularization term. In this section, we study the effect of regularization and initialization on SGD's robustness under covariate shift.

Regularization
To study the effect of regularization on SGD's robustness under covarite shift, we hold all hyperparameters fixed and we change the value of the regularization parameter. Figure 12 shows the outcome. Most initialization schemes for neural networks initialize the weights with values close to zero, hence we expect SGD not perform as poorly on out-of-distribution data as HMC on these networks even without regularization. Therefore, we see that SGD without regularization (reg = 0.0) is still competitive with reasonably regularized SGD.

Initialization
The default initialization scheme for fully-connected layers in Pytorch for example is the He initialization [18]. We use a uniform initialization $\mathcal{U}(-b, b)$ and study the effect of varying b on the performance of SGD under covarite shift for a fully-connected neural network. Figure 13 shows our empirical results, where smaller weights result in better generalization on most of the corruptions.

M.2 Other stochastic optimizers

In addition to SGD, we examine the performance of Adam [27], Adadelta [57], L-BFGS [42, 33] on corrupted MNIST. Figure 14 shows the results for all 4 algorithms on the MNIST dataset under covariate shift, for both fully-connected and convolutional neural networks. We see that SGD, Adam and Adadelta have comparable performance for convolutional neural networks, whereas SGD has an edge over both algorithms on MLP. L-BFGS provides a comparatively

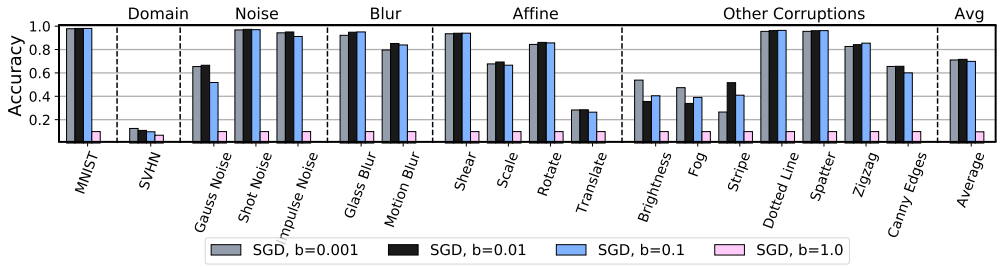


Figure 13: **Effect of initialization on SGD's performance on corrupted MNIST with MLP.** The weights are initialized using a uniform distribution $\mathcal{U}(-b, b)$, and we consider the following values for b: 0.001, 0.01, 0.1, and 1.0. All experiments were run without regularization. For most corruptions, initializing the weights at smaller values leads to better robustness to covariate shift.

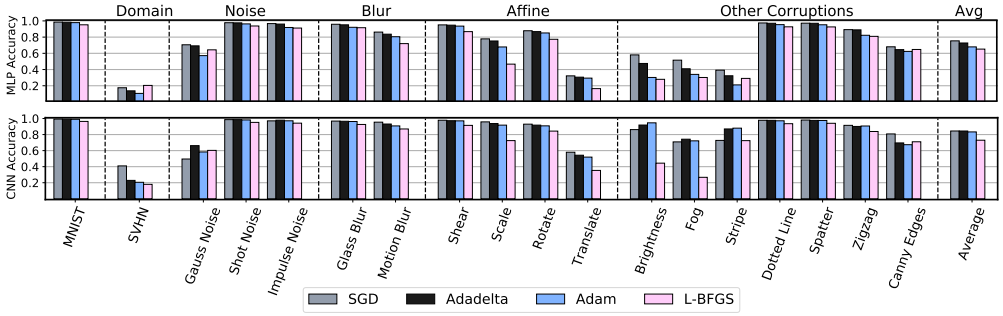


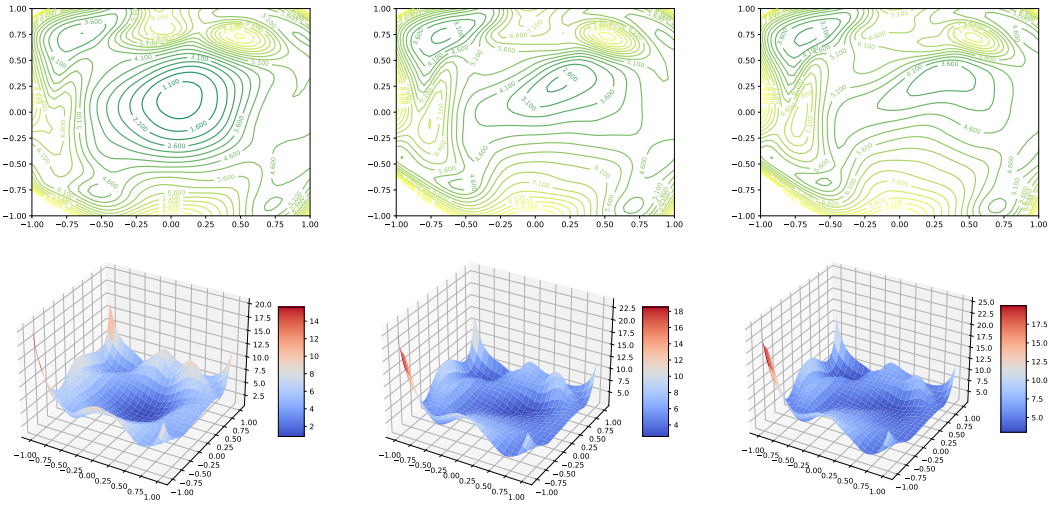
Figure 14: **Robustness on MNIST for different stochastic optimizers.** Accuracy for SGD, Adadelta, Adam and L-BFGS on MNIST under covariate shift. **Top**: Fully-connected network; **bottom**: Convolutional neural network. Adam and Adadelta provide competitive performance with SGD for most corruptions. However, SGD is better on the MLP architecture for some corruptions whereas Adam and Adadelta are better on the *same* corruptions with the CNN architecture.

poor performance and we hypothesise that it is due to the lack of regularization. Naive regularization of the objective function does not improve the performance of L-BFGS.

M.3 Loss surface analysis

There have been several works that tried to characterize the geometric properties of the loss landscape and describe its connection to the generalization performance of neural networks. In particular, it is widely believed that flat minima are able to provide better generalization [20, 26]. Intuitively, the test distribution introduces a horizontal shift in the loss landscape which makes minima that lie in flat regions of the loss surface perform well for both train and test datasets. From the other side, it is well-known that SGD produces flat minima. Hence, we would like to understand the type of distortions that corruptions in the corrupted CIFAR-10 dataset introduce in the loss surface, and evaluate the potential advantage of flat minima in this context.

In the same fashion as Li et al. [32], we visualize the effect of the Gaussian noise corruption on the loss surface for different intensity levels as shown in Figure 15. These plots are produced for two random directions of the parameter space for a ResNet-56 network. We observe that high levels of intensity make the loss surface more flat, but result in a worse test loss overall. We can see visually that the mode in the central flat region, that we denote w_0, is less affected by the corruption than a solution picked at random. Figure 16 shows the



(a) Corruption intensity: 1 (b) Corruption intensity: 3 (c) Corruption intensity: 5

Figure 15: 2D (**top**) and 3D **bottom** visualizations of the loss surface of a ResNet-56 network on corrupted CIFAR-10 with different intensity levels of the Gaussian noise corruption.

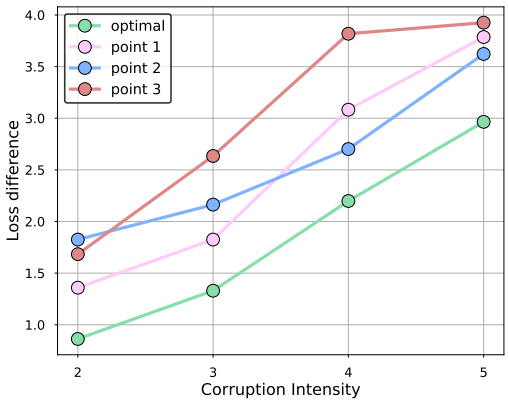


Figure 16: Loss difference between different random solutions, including the mode found through standard SGD training (*optimal* in the legend), and the new mode for each corruption intensity.

loss difference between different solutions including w_0, that we call *optimal*, and the new mode for each corruption intensity. We can see that the mode is indeed less affected by the corruptions than other randomly selected solutions of the same loss region.

N Licensing

The MNIST dataset is made available under the terms of the Creative Commons Attribution-Share Alike 3.0 license. The CIFAR-10 dataset is made available under the MIT license. Our code is a fork of the Google Research repository at https://github.com/google-research/google-research, which has source files released under the Apache 2.0 license.