Tackling covariate shift with node-based Bayesian neural networks

Trinh, Trung Q

Journal of Machine Learning Research
2022

Trinh , T Q , Heinonen , M , Acerbi , L & Kaski , S 2022 , Tackling covariate shift with node-based Bayesian neural networks . in Proceedings of the 39th International Conference on Machine Learning . Proceedings of Machine Learning Research , vol. 162 , Journal of Machine Learning Research , pp. 21751-21775 , International Conference on Machine Learning , Maryland , United States , 17/07/2022 . < https://proceedings.mlr.press/v162/trinh22a.html >

http://hdl.handle.net/10138/353984

unspecified
publishedVersion

Tackling covariate shift with node-based Bayesian neural networks

Trung Trinh [1] **Markus Heinonen** [1] **Luigi Acerbi** [2] **Samuel Kaski** [1 3]

Abstract

Bayesian neural networks (BNNs) promise improved generalization under covariate shift by providing principled probabilistic representations of epistemic uncertainty. However, weight-based BNNs often struggle with high computational complexity of large-scale architectures and datasets. Node-based BNNs have recently been introduced as scalable alternatives, which induce epistemic uncertainty by multiplying each hidden node with latent random variables, while learning a point-estimate of the weights. In this paper, we interpret these latent noise variables as implicit representations of simple and domain-agnostic data perturbations during training, producing BNNs that perform well under covariate shift due to input corruptions. We observe that the diversity of the implicit corruptions depends on the entropy of the latent variables, and propose a straightforward approach to increase the entropy of these variables during training. We evaluate the method on out-of-distribution image classification benchmarks, and show improved uncertainty estimation of node-based BNNs under covariate shift due to input perturbations. As a side effect, the method also provides robustness against noisy training labels.

1. Introduction

Bayesian neural networks (BNNs) induce epistemic uncertainty over predictions by placing a distribution over the weights (MacKay, 1992; 1995; Hinton & van Camp, 1993; Neal, 1996). However, it is challenging to infer the weight posterior due to the high dimensionality and multi-modality of this distribution (Wenzel et al., 2020; Izmailov et al., 2021b). Alternative BNN methods have been introduced to avoid the complexity of weight-space inference, which include combining multiple maximum-a-posteriori (MAP) solutions (Lakshminarayanan et al., 2017), performing inference in the function-space (Sun et al., 2019), or performing inference in a lower dimensional latent space (Karaletsos et al., 2018; Pradier et al., 2018; Izmailov et al., 2020; Dusenberry et al., 2020).

A recent approach to simplify BNNs is *node stochasticity*, which assigns latent noise variables to hidden nodes of the network (Kingma et al., 2015; Gal & Ghahramani, 2016; Karaletsos et al., 2018; Karaletsos & Bui, 2020; Dusenberry et al., 2020; Nguyen et al., 2021). By restricting inference to the node-based latent variables, node stochasticity greatly reduces the dimension of the posterior, as the number of nodes is orders of magnitude smaller than the number of weights in a neural network (Dusenberry et al., 2020). Within this framework, multiplying each hidden node with its own random variable has been shown to produce great predictive performance, while having dramatically smaller computational complexity compared to weight-space BNNs (Gal & Ghahramani, 2016; Kingma et al., 2015; Dusenberry et al., 2020; Nguyen et al., 2021).

In this paper, we focus on *node-based BNNs*, which represent epistemic uncertainty by inferring the posterior distribution of the multiplicative latent node variables while learning a point-estimate of the weight posterior (Dusenberry et al., 2020; Trinh et al., 2020). We show that node stochasticity simulates a set of implicit corruptions in the data space during training, and by learning in the presence of such corruptions, node-based BNNs achieve natural robustness against some real-world input corruptions. This is an important property because one of the key promises of BNNs is robustness under *covariate shift* (Ovadia et al., 2019; Izmailov et al., 2021b), defined as a change in the distribution of input features at test time with respect to that of the training data. Based on our findings, we derive an entropy regularization approach to improve out-of-distribution generalization for node-based BNNs.

In summary, our contributions are:

1. We demonstrate that node stochasticity simulates data-space corruptions during training. We show that the diversity of these corruptions corresponds to the entropy

[1]Department of Computer Science, Aalto University, Finland [2]Department of Computer Science, University of Helsinki, Finland [3]Department of Computer Science, University of Manchester, UK. Correspondence to: Trung Trinh <trung.trinh@aalto.fi>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

of the latent node variables, and training on more diverse generated corruptions produce node-based BNNs that are robust against a wider range of corruptions.

2. We derive an entropy-regularized variational inference formulation for node-based BNNs.

3. We demonstrate excellent empirical results in predictive uncertainty estimation under covariate shift due to corruptions compared to strong baselines on large-scale image classification tasks.

4. We show that, as a side effect, our approach provides robust learning in the presence of noisy training labels.

Our code is available at https://github.com/AaltoPML/node-BNN-covariate-shift.

2. Background

Neural networks. We define a standard neural network $\mathbf{f}(\mathbf{x})$ with L layers for an input $\mathbf{x}$ as follows:

$$\mathbf{f}^0(\mathbf{x}) = \mathbf{x} \tag{1}$$
$$\mathbf{h}^\ell(\mathbf{x}) = \mathbf{W}^\ell \mathbf{f}^{\ell-1}(\mathbf{x}) + \mathbf{b}^\ell \tag{2}$$
$$\mathbf{f}^\ell(\mathbf{x}) = \sigma^\ell\big(\mathbf{h}^\ell(\mathbf{x})\big), \qquad \forall \ell = 1, \dots, L \tag{3}$$
$$\mathbf{f}(\mathbf{x}) = \mathbf{f}^L(\mathbf{x}), \tag{4}$$

where the parameters $\theta = \{\mathbf{W}^\ell, \mathbf{b}^\ell\}_{\ell=1}^L$ consist of the weights and biases, and the $\{\sigma^\ell\}_{\ell=1}^L$ are the activation functions. For the ℓ-th layer, $\mathbf{h}^\ell$ and $\mathbf{f}^\ell$ are the pre- and post-activations, respectively.

Node-based Bayesian neural networks. Probabilistic neural networks constructed using node stochasticity have been studied by Gal & Ghahramani (2016); Kingma et al. (2015); Louizos & Welling (2017); Karaletsos et al. (2018); Karaletsos & Bui (2020); Dusenberry et al. (2020); Trinh et al. (2020); Nguyen et al. (2021). We focus on inducing node stochasticity by multiplying each hidden node with its own random latent variables, and follow the framework of Dusenberry et al. (2020) for optimization. A node-based BNN $\mathbf{f}_\mathcal{Z}(\mathbf{x})$ is defined as:

$$\mathbf{f}_\mathcal{Z}^0(\mathbf{x}) = \mathbf{x} \tag{5}$$
$$\mathbf{h}_\mathcal{Z}^\ell(\mathbf{x}) = (\mathbf{W}^\ell(\mathbf{f}_\mathcal{Z}^{\ell-1}(\mathbf{x}) \circ \mathbf{z}^\ell) + \mathbf{b}^\ell) \circ \mathbf{s}^\ell \tag{6}$$
$$\mathbf{f}_\mathcal{Z}^\ell(\mathbf{x}) = \sigma^\ell\big(\mathbf{h}_\mathcal{Z}^\ell(\mathbf{x})\big), \qquad \forall \ell = 1, \dots, L \tag{7}$$
$$\mathbf{f}_\mathcal{Z}(\mathbf{x}) = \mathbf{f}_\mathcal{Z}^L(\mathbf{x}), \tag{8}$$

where $\mathbf{z}^\ell$ and $\mathbf{s}^\ell$ are the multiplicative latent random variables of the incoming and outgoing signal of the nodes of the ℓ-th layer, and $\circ$ denotes the Hadamard (element-wise) product. We collect all latent variables to $\mathcal{Z} = \{\mathbf{z}^\ell, \mathbf{s}^\ell\}_{\ell=1}^L$.[1]

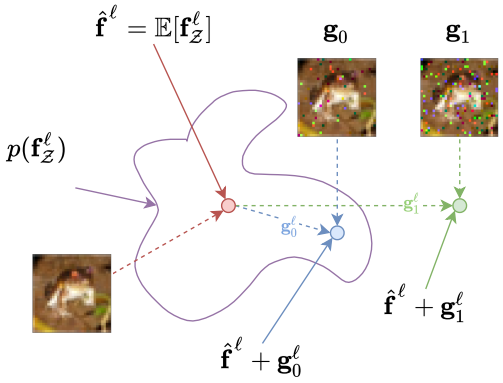


Figure 1. A sketch depicting the connection between the output distribution at the ℓ-th layer induced by node stochasticity (purple) centered on the average output (●), and the output shifts generated by input corruptions (●,●). We expect good performance under mild corruption $\mathbf{g}_0$, as the resulting shift remains inside the high-density region of $p(\mathbf{f}_\mathcal{Z}^\ell)$, and worse results under severe corruption $\mathbf{g}_1$.

To learn the network parameters, we follow Dusenberry et al. (2020) and perform variational inference (Blei et al., 2017) over the weight parameters θ and latent node variables $\mathcal{Z}$. We begin by defining a prior $p(\theta, \mathcal{Z}) = p(\theta)p(\mathcal{Z})$. We set a variational posterior approximation $q_{\hat{\theta},\phi}(\theta, \mathcal{Z}) = q_{\hat{\theta}}(\theta)q_\phi(\mathcal{Z})$, where $q_{\hat{\theta}}(\theta) = \delta(\theta - \hat{\theta})$ is a Dirac delta distribution and $q_\phi(\mathcal{Z})$ is a Gaussian or a mixture of Gaussians distribution. We infer the posterior by minimizing the Kullback-Leibler (KL) divergence between variational approximation q and true posterior $p(\theta, \mathcal{Z}|\mathcal{D})$. This is equivalent to maximizing the evidence lower bound (ELBO):

$$\mathcal{L}(\hat{\theta}, \phi) = \mathbb{E}_{q_\phi(\mathcal{Z})}\Big[\log p(\mathcal{D}|\hat{\theta}, \mathcal{Z})\Big] - \mathrm{KL}\Big[q_\phi(\mathcal{Z}) \,||\, p(\mathcal{Z})\Big] + \log p(\hat{\theta}). \tag{9}$$

In essence, we find a MAP solution for the more numerous weights θ, while inferring the posterior distribution of the latent variables $\mathcal{Z}$. We refer the reader to Appendix A for detailed derivations.

Neural networks under covariate shift. In this paper, we focus on covariate shift from input corruptions, following the setting of Hendrycks & Dietterich (2019). To simulate covariate shift, one can take an input $\mathbf{x}$ assumed to come from the same distribution as the training samples and apply

[1] In this paper, we use a slightly more general definition of node-based BNN with two noise variables per node, and compare it with single-variable variants in Section 5.

an input corruption $\mathbf{g}^0$ to form a shifted version $\mathbf{x}^c$ of $\mathbf{x}$:

$$\mathbf{x}^c = \mathbf{x} + \mathbf{g}^0(\mathbf{x}). \tag{10}$$

For instance, $\mathbf{x}$ could be an image and $\mathbf{g}^0$ can represent the shot noise corruption (Hendrycks & Dietterich, 2019). The input corruption $\mathbf{g}^0(\mathbf{x})$ creates a shift in the output of each layer $\mathbf{g}^\ell(\mathbf{x})$ (see Fig. 1). We can approximate these shifts by first-order Taylor expansion (see Appendix C for full derivation),

$$\overbrace{\mathbf{g}^\ell(\mathbf{x})}^{\text{shift}} = \overbrace{\mathbf{f}^\ell(\mathbf{x}^c)}^{\text{corrupted output}} - \overbrace{\mathbf{f}^\ell(\mathbf{x})}^{\text{clean output}} \tag{11}$$

$$\approx \mathbf{J}_\sigma\big[\mathbf{h}^\ell(\mathbf{x})\big]\big(\mathbf{W}^\ell \mathbf{g}^{\ell-1}(\mathbf{x})\big), \tag{12}$$

where $\mathbf{J}_{\sigma^\ell} = \partial\sigma^\ell/\partial\mathbf{h}^\ell$ denotes the (diagonal) Jacobian of the activation σ^ℓ with respect to $\mathbf{h}^\ell$. While $\mathbf{g}^0$ causes activation shifts in every layer of the network, we focus on the shift in the final output layer $\mathbf{g}^L$. The approximation in Eq. (12) shows that this shift depends on the input $\mathbf{x}$, the network's architecture (e.g., choice of activation functions) and parameters θ. We measure the robustness of a network with respect to a corruption $\mathbf{g}^0(\cdot)$ on the dataset $\mathcal{D} = \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$ by the induced *mean square shift*,

$$\mathrm{MSS}_g = \frac{1}{N}\sum_{n=1}^{N} ||\mathbf{g}^L(\mathbf{x}_n)||_2^2, \tag{13}$$

where MSS_g is the average shift on the data. Ideally, we want MSS_g to be small for the network to still provide nearly correct predictions given corrupted inputs. When the training data and the architecture are fixed, MSS_g depends on the parameters θ. A direct approach to find θ minimizing MSS_g is to apply the input corruption $\mathbf{g}^0$ to each input $\mathbf{x}_n$ during training to teach the network to output the correct label $\mathbf{y}_n$ given $\mathbf{g}^0(\mathbf{x}_n)$. However, this approach is not domain-agnostic and requires defining a list of corruptions beforehand. In the next sections, we discuss the usage of multiplicative latent node variables as an implicit way to simulate covariate shifts during training.

3. Characterizing implicit corruptions

In this section, we demonstrate that multiplicative node variables correspond to implicit input corruptions. We show how to extract and visualize these new corruptions.

3.1. Relating input corruptions and multiplicative nodes

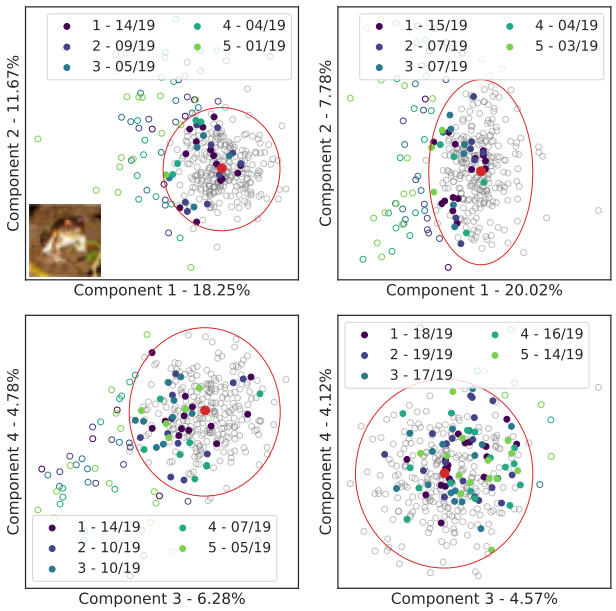
The node-based BNN of Eqs. (5)-(8) induces the *predictive posterior* $p(\mathbf{f}_{\mathcal{Z}}^\ell(\mathbf{x}))$ over the ℓ-th layer outputs by marginalizing over the variational latent *parameter posterior* $q(\mathcal{Z}_{\leq\ell})$. Optimization of the variational objective in Eq. (9) enforces the model to achieve low loss on the training data despite each layer output being corrupted by noise from $q(\mathcal{Z})$, represented by the expected log likelihood term of the ELBO. Let $\hat{\mathbf{f}}^\ell(\mathbf{x})$ denote the mean predictive posterior,

$$\hat{\mathbf{f}}^\ell(\mathbf{x}) = \mathbb{E}_{q(\mathcal{Z})}\big[\mathbf{f}_{\mathcal{Z}}^\ell(\mathbf{x})\big], \quad \forall\ell = 1, \ldots, L, \tag{14}$$

and where we denote the final output $\hat{\mathbf{f}}(\mathbf{x}) = \hat{\mathbf{f}}^L(\mathbf{x})$. If the shifted output $\hat{\mathbf{f}}^\ell(\mathbf{x} + \mathbf{g}^0(\mathbf{x})) = \hat{\mathbf{f}}^\ell(\mathbf{x}) + \mathbf{g}^\ell(\mathbf{x})$ caused by corrupting a training sample $\mathbf{x}$ using $\mathbf{g}^0$ lies within the predictive distribution of $\mathbf{f}_{\mathcal{Z}}^\ell(\mathbf{x})$ (blue dot in Fig. 1), then the model can map this corrupted version of $\mathbf{x}$ to its correct label. This implies robustness against the space of implicit corruptions generated by $q(\mathcal{Z})$, which indirectly leads to robustness against real corruptions. However, standard variational inference will converge to a posterior whose entropy is calibrated for the variability in the training data, but does not necessarily account for corruptions caused by covariate shifts. Thus, the posterior might cover the corruption with low severity $\mathbf{g}_0$ (blue dot in Fig. 1), but not the one with higher severity $\mathbf{g}_1$ (green dot in Fig. 1). To promote predictive distributions that are more robust to perturbations, we propose to increase the entropy of $p(\mathbf{f}_{\mathcal{Z}}^\ell(\mathbf{x}))$ by increasing the entropy of the variational posterior $q(\mathcal{Z})$.

Empirical demonstration. To illustrate our intuition, we present an example with two node-based BNNs, one with high entropy and one with lower entropy. We use the ALL-CNN-C architecture of Springenberg et al. (2014) and CIFAR10 (Krizhevsky et al., 2009). We initialize the standard deviations of $q(\mathcal{Z})$ for the low-entropy model using the half-normal $\mathcal{N}^+(0.16, 0.02)$, while we use $\mathcal{N}^+(0.32, 0.02)$ for the high-entropy model. For brevity, we refer to the former model as $\mathcal{M}_{16}$ and the latter model as $\mathcal{M}_{32}$. In the left plot of Fig. 3, we show that, after training, $\mathcal{M}_{32}$ retains higher variational posterior entropy than $\mathcal{M}_{16}$ due to having higher initial standard deviations for $q(\mathcal{Z})$.[2] We use principal component analysis (PCA) to visualize the samples from the output distribution $p(\mathbf{f}_{\mathcal{Z}}^\ell(\mathbf{x}))$ of the ℓ-th layer with respect to one input image $\mathbf{x}$, as well as the output $\{\hat{\mathbf{f}}^\ell(\mathbf{x} + \mathbf{g}_i(\mathbf{x}))\}_{i=1}^{95}$ under the real image corruptions $\{\mathbf{g}_i\}_{i=1}^{95}$ from Hendrycks & Dietterich (2019). There are 19 corruption types with 5 levels of severity, totalling 95 corruption functions. Fig. 2 shows the activations of the last layer, projected into a two-dimensional subspace with PCA for visualization. From this figure, we can see that there is more overlap between samples from $p(\mathbf{f}_{\mathcal{Z}}^\ell(\mathbf{x}))$ and the shifted outputs $\{\hat{\mathbf{f}}^\ell(\mathbf{x} + \mathbf{g}_i(\mathbf{x}))\}_{i=1}^{95}$ for $\mathcal{M}_{32}$ in Fig. 2b than for $\mathcal{M}_{16}$ in Fig. 2a. This indicates that during training the posterior of $\mathcal{M}_{32}$ is able to simulate a larger number of implicit corruptions bearing resemblance to the real-world corruptions than the posterior of $\mathcal{M}_{16}$, leading to better neg-

[2] Obtaining high-entropy models by starting with high-entropy initializations is a simple heuristic for the purpose of this example. We introduce a principled approach in Section 4.



(a) $\mathcal{M}_{16}$ (b) $\mathcal{M}_{32}$

Figure 2. PCA plots of the last layer's outputs of models **(a)** $\mathcal{M}_{16}$ and **(b)** $\mathcal{M}_{32}$ with respect to one sample from CIFAR-10 (included in the top left panel). Grey circles are samples from the output distribution induced by $q(\mathcal{Z})$, while the red ellipse shows their 99 percentile. The red circle denotes the expected output $\hat{\mathbf{f}}^\ell(\mathbf{x}) = \mathbb{E}_{q(\mathcal{Z})}[\mathbf{f}^\ell_{\mathcal{Z}}(\mathbf{x})]$ of the test point. Other colored circles represents the expected output $\hat{\mathbf{f}}^\ell$ of the 19 corrupted versions of the test point under 5 levels of severity Hendrycks & Dietterich (2019). Most of the mild corruptions reside inside the predictive posterior of both models (filled color circles). By contrast, only the higher-entropy $\mathcal{M}_{32}$ model encapsulates a large fraction of the severe corruptions – empirically demonstrating the intuition sketched in Fig. 1 and described in Section 3.1.

ative log-likelihood (NLL) accross all level of corruptions as well as on the clean test set in Fig. 3. This example supports our intuition that increasing the entropy of the latent variables $\mathcal{Z}$ allows them to simulate more diverse implicit corruptions, thereby boosting the model's robustness against a wider range of input corruptions.

Why latent variables at every layer? In principle, we could have introduced latent variables only to the first layer of the network, as the shift simulated in the first layer will propagate to subsequent layers. However, modern NNs contain asymmetric activation functions such as ReLU or Softplus, which can attenuate the signal of the shift in the later layers. Thus, the latent variables in every layer (after the first one) maintain the strength of the shift throughout the network during the forward pass. Moreover, by using latent variables at every layer – as opposed to only the first layer – we can simulate a more diverse set of input corruptions, since we can map each sample $\mathcal{Z}$ from $q(\mathcal{Z})$ to an input

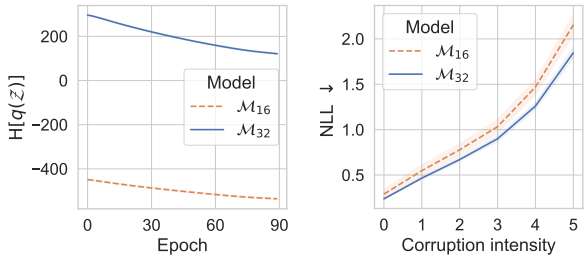
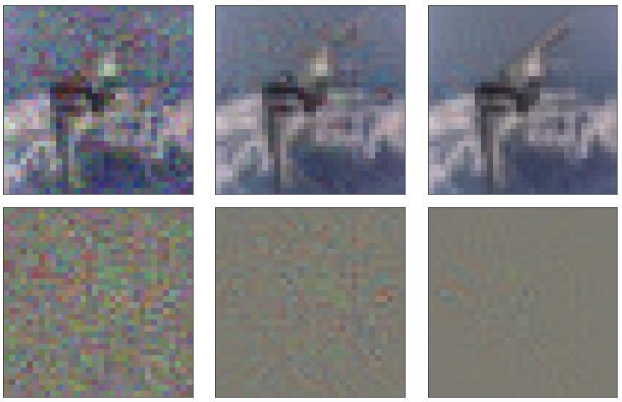


Figure 3. **(Left)** Example evolution of $\mathbb{H}[q(\mathcal{Z})]$ during training, which shows that the variational entropy decreases over time. **(Right)** Performance of two models under different corruption levels (level 0 indicates no corruption). The model with higher entropy $\mathcal{M}_{32}$ performs better than the one with lower entropy $\mathcal{M}_{16}$ across all corruption levels. For each result in both plots, we report the mean and standard deviation over 25 runs. The error bars in the left plot are too small to be seen.



(a) $\lambda = 0.03$ (b) $\lambda = 0.10$ (c) $\lambda = 0.30$

Figure 4. Implicit corruptions generated from model $\mathcal{M}_{32}$ with respect to one image by minimizing the loss in Eq. (16) under varying λ. Top row are the resulting images from the corruptions below. We can see that λ controls the severity of the generated corruptions.

corruption as shown in the following section.

3.2. Visualizing the implicit corruptions

Next, we show how to find the explicit image corruptions that correspond to the stochasticity of the predictive posterior. Let $\mathcal{Z}$ be a sample drawn from $q(\mathcal{Z})$. If we assume that $\mathcal{Z}$ corresponds to an input corruption $\mathbf{g}(\mathbf{x})$:

$$\mathbf{f}_{\mathcal{Z}}(\mathbf{x}) = \hat{\mathbf{f}}(\mathbf{x} + \mathbf{g}(\mathbf{x})) \tag{15}$$

then we can approximately solve for $\mathbf{g}(\mathbf{x}) = \mathbf{x}^c - \mathbf{x}$ by finding $\mathbf{x}^c$ that minimizes

$$\mathcal{L}(\mathbf{x}^c) = \frac{1}{2}\left|\left|\mathbf{f}_{\mathcal{Z}}(\mathbf{x}) - \hat{\mathbf{f}}(\mathbf{x}^c)\right|\right|_2^2 + \frac{\lambda}{2}||\mathbf{g}(\mathbf{x})||_2^2 \tag{16}$$

using gradient descent. The second term with a coefficent $\lambda \geq 0$ regularizes the norm of $\mathbf{g}(\mathbf{x})$. This approach is simi-

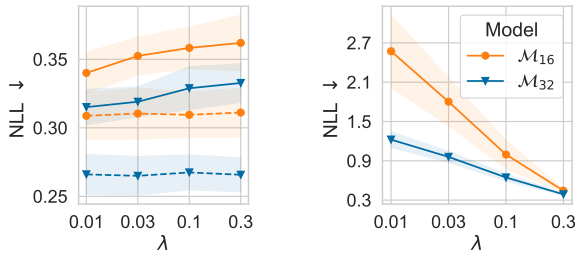


Figure 5. Negative log-likelihood (NLL) on 1024 test images of CIFAR-10 corrupted by the implicit corruptions generated by $\mathcal{M}_{16}$ and $\mathcal{M}_{32}$, whose intensities are controlled by λ in Eq. (16). For each result, we report the mean and standard deviation over 10 runs. **(Left)** Each model is tested on the corruptions that it generated. Dashed lines are results on the clean images for reference. Each model is resistant to its own corruptions, as evidenced by the slight decrease in performance under different λ. **(Right)** Each model is tested on the corruptions produced by the other. The model with higher entropy $\mathcal{M}_{32}$ is more robust against the corruptions of the one with lower entropy $\mathcal{M}_{16}$ than the reverse, which further supports the notion that higher entropy provides better robustness against input corruptions.

lar to the method of finding adversarial examples of Goodfellow et al. (2014). Fig. 4 visualizes the corruptions generated by $\mathcal{M}_{32}$ on a test image of CIFAR10 under different λ. We can see that λ controls the severity of the corruptions, with smaller λ corresponding to higher severity.

Is a model robust against its own corruptions? We use both models $\mathcal{M}_{16}$ and $\mathcal{M}_{32}$ to generate corruptions on a subset of 1024 test images of CIFAR10. We generate 8 corruptions per test image. The left plot of Fig. 5 shows that each model is robust against its own implicit corruptions even when the corruption is severe, as evidenced by the small performance degradation under different λ. By comparing the right plot to the left plot, we can see that each model is less resistant to the corruptions generated by the other model than its own corruptions. Crucially, however, the performance of $\mathcal{M}_{32}$ under the corruptions generated by $\mathcal{M}_{16}$ is better than the reverse. This example thus suggests that while each model is resistant to its own corruptions, the model with higher entropy shows better robustness against the corruptions created by the other model.[3]

4. Maximizing variational entropy

The previous sections motivated the usage of variational posteriors with high entropy from the perspective of simulating a diverse set of input corruptions. In this section, we discuss a simple method to increase the variational entropy.

[3]We note that this a proof of concept and more experiments are needed to verify if these results hold true in general.

4.1. The augmented ELBO

Our goal is to find posterior approximations that have high entropy. In the previous section, we considered a heuristic approach of initializing $q(\mathcal{Z})$ with high entropy (Fig. 3). However, if the initial entropy of $q(\mathcal{Z})$ is too high, training will converge slowly due to high variance in the gradients.

Here we consider the approach of augmenting the original ELBO in Eq. (9) with an extra γ-weighted entropy term, adapting Mandt et al. (2016). The augmented γ-ELBO is

$$\mathcal{L}_\gamma(\hat{\theta}, \phi) = \mathcal{L}(\hat{\theta}, \phi) + \gamma \mathbb{H}\big[q_\phi(\mathcal{Z})\big] \tag{17}$$

$$= \underbrace{\mathbb{E}_{q_\phi(\mathcal{Z})}\Big[\log p(\mathcal{D}|\hat{\theta}, \mathcal{Z})\Big]}_{\text{expected log-likelihood}} - \underbrace{\mathbb{H}\big[q_\phi(\mathcal{Z}), p(\mathcal{Z})\big]}_{\text{cross-entropy}} \tag{18}$$

$$+ \underbrace{(\gamma+1)\mathbb{H}\big[q_\phi(\mathcal{Z})\big]}_{\text{variational entropy}} + \underbrace{\log p(\hat{\theta})}_{\text{weight prior}}, \tag{19}$$

where we decompose the KL into its cross-entropy and entropy terms. $\gamma \geq 0$ controls the amount of extra entropy, with $\gamma = 0$ reducing to the classic ELBO in Eq. (9). We can interpret the terms in Eq. (19) as follows: the first term fits the variational parameters to the dataset; the second and fourth terms regularize ϕ and $\hat{\theta}$ respectively; the third term increases the entropy of the variational posterior.

4.2. Tempered posterior inference

One could also arrive at Eq. (19) by minimizing the KL divergence between the approximate posterior $q_\phi(\hat{\theta}, \mathcal{Z})$ and the tempered posterior $p_\gamma(\theta, \mathcal{Z}|\mathcal{D})$ (Mandt et al., 2016):

$$p_\gamma(\theta, \mathcal{Z}|\mathcal{D}) = \frac{p(\mathcal{D}|\theta, \mathcal{Z})^\tau p(\mathcal{Z}, \theta)^\tau}{p_\gamma(\mathcal{D})} \tag{20}$$

$$p_\gamma(\mathcal{D}) = \int_\theta \int_\mathcal{Z} p(\mathcal{D}|\theta, \mathcal{Z})^\tau p(\mathcal{Z}, \theta)^\tau d\mathcal{Z} d\theta, \tag{21}$$

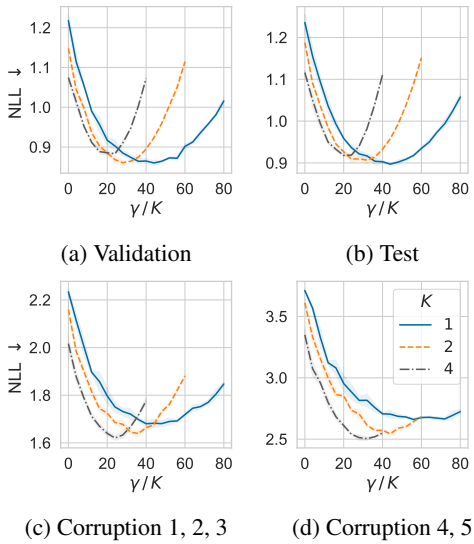
where the temperature $\tau = 1/(\gamma+1)$. The tempered posterior variational approximation

$$\arg\min_{\hat{\theta}, \phi} \ \frac{1}{\tau} \mathrm{KL}\Big[q_\phi(\hat{\theta}, \mathcal{Z}) \ || \ p_\gamma(\theta, \mathcal{Z}|\mathcal{D})\Big] \tag{22}$$

is equivalent to tempered ELBO maximization

$$\arg\max_{\hat{\theta}, \phi} \ \mathcal{L}_\gamma(\hat{\theta}, \phi) - \log p_\gamma(\mathcal{D})^{\frac{1}{\tau}}. \tag{23}$$

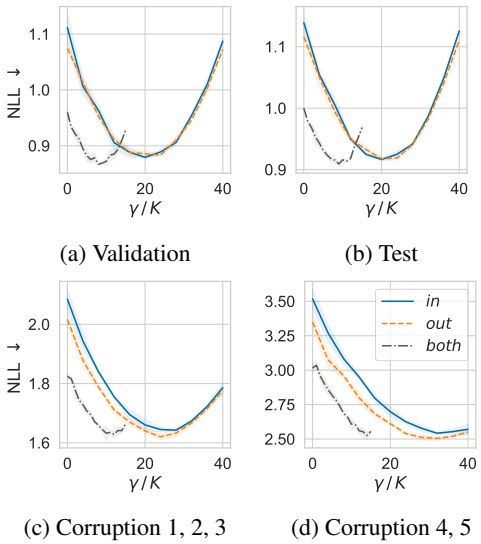
We refer the reader to Appendix B for detailed derivations. The entropy-regularized γ-ELBO thus corresponds to the family of tempered variational inference, and with positive $\gamma > 0$, to 'hot' posteriors (Wenzel et al., 2020). In the next section, we will demonstrate empirically the benefits of such hot posteriors in node-based BNNs.



(a) Validation (b) Test

(c) Corruption 1, 2, 3 (d) Corruption 4, 5

Figure 6. Results of (VGG16 / CIFAR100 / out) with different K. The results in (c) are averaged over the first three levels of corruption, and those in (d) are averaged over the last two levels. Notice that we rescale γ by K in the x-axis to provide better visualization, as we find that larger K requires higher optimal γ. We report the mean and standard deviation over 5 runs for each result. Overall, more components provide better optimal performance on OOD data. Higher γ provides better OOD performance as the cost of ID performance.



(a) Validation (b) Test

(c) Corruption 1, 2, 3 (d) Corruption 4, 5

Figure 7. Results of VGG16 on CIFAR100 with different latent variable structures. Here we use $K = 4$ components. We report the mean and standard deviation over 5 runs for each result. Overall, using either only the latent input variables or latent output variables requires higher optimal γ than using both. Using only the latent output variables produces better results than the latent input variables on OOD data, despite similar ID performance.

5. Experiments

In this section, we present experimental results of node-based BNNs on image classification tasks. For the datasets, we use CIFAR (Krizhevsky et al., 2009) and TINYIMAGENET (Le & Yang, 2015), which have corrupted versions of the test set provided by Hendrycks & Dietterich (2019). We use VGG16 (Simonyan & Zisserman, 2014), RESNET18 (He et al., 2016a) and PREACTRESNET18 (He et al., 2016b) for the architectures. We test three structures of latent variables: *in*, where we only use the input latent variables $\{\mathbf{z}^\ell\}_{\ell=1}^L$; *out*, where we only use the output latent variables $\{\mathbf{s}^\ell\}_{\ell=1}^L$; and *both*, where we use both $\{\mathbf{z}^\ell\}_{\ell=1}^L$ and $\{\mathbf{s}^\ell\}_{\ell=1}^L$. We use $K \in \{1, 2, 4\}$ Gaussian component(s) in the variational posterior. For each result, we report the mean and standard deviation over multiple runs.

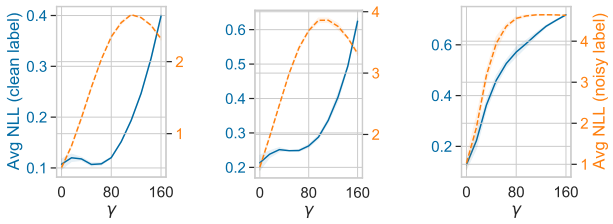
5.1. Effects of γ on covariate shift

In this section, we study the changes in performance of the model trained with the γ-ELBO objective as we increase γ. We perform experiments with VGG16 on CIFAR100, and use the corrupted test set of CIFAR100 provided by Hendrycks & Dietterich (2019). In Fig. 6, we show the *out* model's behaviour under a different number of Gaussian components K. In Fig. 7, we show the results of a model with $K = 4$ components under the different latent variable structures *in*, *out*, and *both*.

These figures show that performance across different test sets improves as γ increases up until a threshold and then degrades afterward. The optimal γ for each set of test images correlates with the severity of the corruptions, where more severe corruptions can be handled by enforcing more diverse set of implicit corruptions during training. However, learning on a more diverse implicit corruptions requires higher capacity, and reduces the learning capacity needed to obtain good performance on the in-distribution (ID) data. The entropy coefficient γ thus controls the induced trade-off between ID performance and out-of-distribution (OOD) robustness.

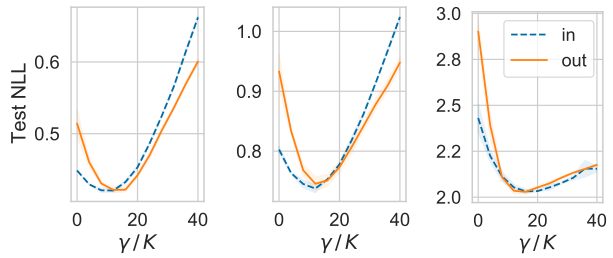
Fig. 6 shows that for ID data, the optimal performance of the model (at optimal γ) remains similar under different K. On OOD data, however, higher K consistently produces better results as γ varies. The optimal γ is higher for variational distributions with more components. This finding is likely because with more mixture components, the variational posterior can approximate the true posterior more accurately, and thus it can better expand into the high-density region of the true posterior as its entropy increases.

Fig. 7 shows the optimal performance on ID data is quite similar between different latent architectures. On OOD, the optimal performance of using both input and output latent variables is similar to using only output latent variables,



(a) 20% (b) 40% (c) 80%

Figure 8. Results of RESNET18 on two subsets of CIFAR10 training samples with clean and noisy labels. Here we use $K = 4$ components and only the latent output variables. We denote the percentage of training samples with corrupted labels under each subfigure. We report the mean and standard deviation over 5 runs for each result. As γ increases, the NLL of noisy labels increases much faster than that of clean labels even when the majority of labels are wrong (c), indicating that higher γ prevents the model from memorizing random labels.



(a) 20% (b) 40% (c) 80%

Figure 9. Results of RESNET18 on clean CIFAR10 test sets under different percentages of noise in training labels. We report the mean and standard deviation over 5 runs for each result. As high γ prevents learning from noisy labels as demonstrated in Fig. 8, it leads to improved performance on clean test sets.

while using only input latent variables produces slightly worse optimal performance. The optimal γ is lower when the model uses both types of latent variables $(\mathbf{z}, \mathbf{s})$, because the entropy of the product of two latent variables increases rapidly as we increase the entropy of both latent variables.

We also observe these patterns in other architectures and datasets (see Appendix I). In summary, from our experimental results we find that using only output latent variables with a sufficient number of components (e.g., $K = 4$) achieves excellent results for node-based BNNs in our benchmark.

5.2. Effects of γ on robustness against noisy labels

Learning wrong labels amounts to memorizing random patterns, which requires more capacity from the model than learning generalizable patterns (Arpit et al., 2017). We hypothesize that if we corrupt wrongly labelled training samples with sufficiently diverse implicit corruptions, we overwhelm the neural network making it unable to memorize these spurious patterns during training. To test this intuition, we follow the experiment in Jiang et al. (2018), where we take a percentage of training samples in CIFAR10 and corrupt their labels. We thus split the training set into two parts: $\mathcal{D}_1$ containing only samples with correct labels, and $\mathcal{D}_2$ including those with wrong labels. We then track the final NLL of $\mathcal{D}_1$ and $\mathcal{D}_2$ under different γ, and visualize the results in Fig. 8. This figure shows that as γ increases, the NLL of $\mathcal{D}_2$ (noisy labels) increases much faster than that of $\mathcal{D}_1$ (clean labels), indicating that the network fails to learn random patterns under simulated corruptions. As a consequence, the model generalizes better on the test set, as shown in Fig. 9.

5.3. Benchmark results

Figs. 10 and 11 present the results of node-based BNNs and baselines on CIFAR10/CIFAR100 and TINYIMAGENET. We choose SWAG (Maddox et al., 2019), cSG-HMC (Zhang et al., 2020) and ASAM (Kwon et al., 2021) as our baselines. These are strong baselines, as both SWAG and cSG-HMC have demonstrated state-of-the-art uncertainty estimation, while ASAM produce better MAP models than stochastic gradient descent by actively seeking wide loss valleys. We repeated each experiment 25 times with different random seeds. For each method, we also consider its ensemble version where we combine 5 models from different runs when making predictions. For the ensemble versions, each experiment is repeated 5 times. We use 30 Monte Carlo samples for node-based BNNs, SWAG, cSG-HMC and their ensemble versions to estimate the posterior predictive distribution. We use standard performance metrics of expected calibration error (ECE) (Naeini et al., 2015), NLL and predictive error. We use RESNET18 for CIFAR10/CIFAR100 and PREACTRESNET18 for TINYIMAGENET. We also include the result of VGG16 on CIFAR10/CIFAR100 in Appendix G. For evaluation, we use the corrupted test images provided by Hendrycks & Dietterich (2019).

On CIFAR100, node-based BNNs outperform the baselines in NLL and error, however SWAG performs best on CIFAR10. Interestingly, in CIFAR100, node-based BNNs and their ensembles have worse ECE than the baselines on ID data, however as the test images become increasingly corrupted, the ECEs of the baselines degrade rapidly while the ECE of node-based BNNs remains below a threshold. Similar behaviors are observed on TINYIMAGENET, with the node-based BNNs produce the lowest NLL and error while not experiencing ECE degradation under corruptions.

6. Related works

Multiplicative latent node variables in BNNs. There have been several earlier works that utilize multiplicative latent node variables, either as a primary source of pre-

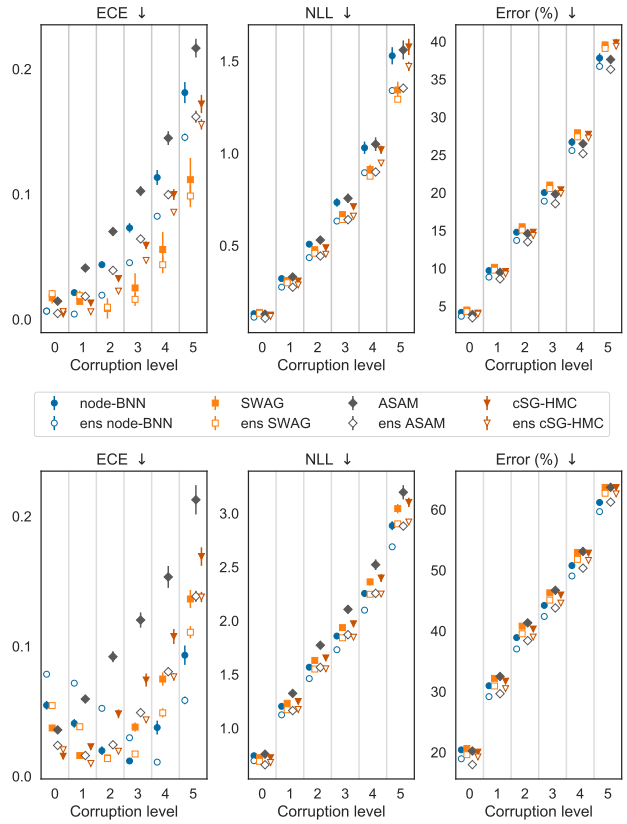


Figure 10. Results of RESNET18 on CIFAR10 **(top)** and CIFAR100 **(bottom)**. We use $K = 4$ and only the latent output variables for node-based BNNs. We plot ECE, NLL and error for different corruption levels, where level 0 indicates no corruption. We report the average performance over 19 corruption types for level 1 to 5. We denote the ensemble of a method using the shorthand *ens* in front of the name. Each result is the average over 25 runs for *non-ens* versions and 5 runs for *ens* versions. The error bars represent the standard deviations across different runs. Node-based BNNs and their ensembles (blue) perform best across all metrics on OOD data of CIFAR100, while having competitive results on CIFAR10. We include a larger version of this plot in Appendix G.

dictive uncertainty such as MC-Dropout (Gal & Ghahramani, 2016), Variational Dropout (Kingma et al., 2015), Rank-1 BNNs (Dusenberry et al., 2020) and Structured Dropout (Nguyen et al., 2021); or to improve the flexibility of the mean-field Gaussian posterior in variational inference (Louizos & Welling, 2017). Here we study the contribution of these latent variables to robustness under covariate shift.

BNNs under covariate shift. Previous works have evaluated the predictive uncertainty of BNNs under covariate shift (Ovadia et al., 2019; Izmailov et al., 2021b), with the recent work by Izmailov et al. (2021b) showing that standard BNNs with high-fidelity posteriors perform worse than MAP solutions under covariate shift. Izmailov et al.

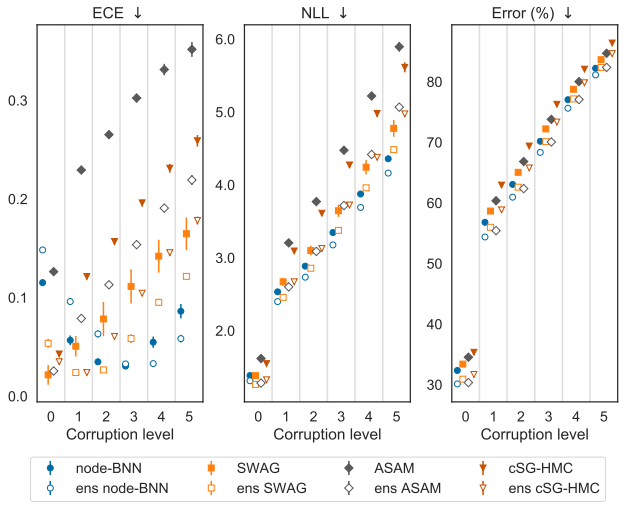


Figure 11. Results of PREACTRESNET18 on TINYIMAGENET. We use $K = 4$ and only the latent output variables for node-based BNNs. We plot ECE, NLL and error for different corruption levels, where level 0 indicates no corruption. We report the average performance over 19 corruption types for level 1 to 5. We denote the ensemble of a method using the shorthand *ens* in front of the name. Each result is the average over 25 runs for *non-ens* versions and 5 runs for *ens* versions. The error bars represent the standard deviations across different runs. Node-based BNNs and their ensembles (blue) perform best accross all metrics on OOD data, while having competitive performance on ID data. We include a larger version of this plot in Appendix G.

(2021a) attributed this phenomenon to the absence of posterior contraction on the null-space of the data manifold. This problem is avoided in node-based BNNs as they still maintain a point-estimate for the weights.

Dropout as data augmentation. Similar to our study, a previous work by Bouthillier et al. (2015) studied Dropout from the data augmentation perspective. Here we study latent variables with more flexible posterior (mixture of Gaussians) and focus on simulating input corruptions for OOD robustness.

Adversarial robustness via feature perturbations. Data-space perturbations have been investigated as a means to defend neural networks against adversarial attacks (Li et al., 2018; Jeddi et al., 2020; Vadera et al., 2020).

Tempered posteriors. Tempered posteriors have been used in variational inference to obtain better variational posterior approximations (Mandt et al., 2016). A recent study put the focus on the cold posterior effect of weight-based BNNs (Wenzel et al., 2020). We have shown that our approach of regularizing the variational entropy is equivalent to performing variational inference with a hot posterior

as the target distribution. Tempered posteriors have also been studied in Bayesian statistics as a means to defend against model misspecification (Grünwald, 2012; Miller & Dunson, 2019; Alquier & Ridgway, 2020; Medina et al., 2021). Covariate shift is a form of model misspecification, as model mismatch arises from using a model trained under different assumptions about the statistics of the data.

7. Conclusion

We analyzed node-based BNNs from the perspective of using latent node variables for simulating input corruptions. We showed that by regularizing the entropy of the latent variables, we increase the diversity of the implicit corruptions, and thus improve performance of node-based BNNs under covariate shift. Across CIFAR10, CIFAR100 and TINYIMAGENET, entropy regularized node-based BNNs produce excellent results in uncertainty metrics on OOD data.

In this study, we focused on variational inference, leaving the study of implicit corruptions under other approximate inference methods as future work. Furthermore, our work shows the benefits of hot posteriors and argues for an inherent trade-off between ID and OOD performance in node-based BNNs. It is an interesting future direction to study these questions in weight-based BNNs. Finally, our work presented entropy as a surprisingly useful summary statistic that can partially explain the complex connection between the variational posterior and corruption robustness. One important research direction is to develop more informative statistics that can better encapsulate this connection.

Acknowledgement

This work was supported by the Academy of Finland (Flagship programme: Finnish Center for Artificial Intelligence FCAI and grants no. 292334, 294238, 319264, 328400) and UKRI Turing AI World-Leading Researcher Fellowship, EP/W002973/1. We acknowledge the computational resources provided by Aalto Science-IT project and CSC–IT Center for Science, Finland.

References

Alquier, P. and Ridgway, J. Concentration of tempered posteriors and of their variational approximations. *The Annals of Statistics*, 48(3):1475–1497, 2020.

Arpit, D., Jastrzębski, S., Ballas, N., Krueger, D., Bengio, E., Kanwal, M. S., Maharaj, T., Fischer, A., Courville, A., Bengio, Y., and Lacoste-Julien, S. A closer look at memorization in deep networks. In *ICML*, pp. 233–242. PMLR, 2017.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

Bouthillier, X., Konda, K., Vincent, P., and Memisevic, R. Dropout as data augmentation. *arXiv preprint arXiv:1506.08700*, 2015.

Dusenberry, M., Jerfel, G., Wen, Y., Ma, Y., Snoek, J., Heller, K., Lakshminarayanan, B., and Tran, D. Efficient and scalable Bayesian neural nets with rank-1 factors. In *ICML*, pp. 2782–2792, 2020.

Gal, Y. and Ghahramani, Z. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *ICML*, 2016.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Grünwald, P. The Safe Bayesian. In *International Conference on Algorithmic Learning Theory*, pp. 169–183. Springer, 2012.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016a.

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

Hendrycks, D. and Dietterich, T. Benchmarking neural network robustness to common corruptions and perturbations. *Proceedings of the International Conference on Learning Representations*, 2019.

Hinton, G. E. and van Camp, D. Keeping the neural networks simple by minimizing the description length of the weights. In *COLT*, pp. 5–13, 1993.

Izmailov, P., Maddox, W. J., Kirichenko, P., Garipov, T., Vetrov, D., and Wilson, A. G. Subspace inference for Bayesian deep learning. In *UAI*, pp. 1169–1179, 2020.

Izmailov, P., Nicholson, P., Lotfi, S., and Wilson, A. G. Dangers of Bayesian model averaging under covariate shift. *arXiv preprint arXiv:2106.11905*, 2021a.

Izmailov, P., Vikram, S., Hoffman, M. D., and Wilson, A. G. What are Bayesian neural network posteriors really like? *arXiv preprint arXiv:2104.14421*, 2021b.

Jebara, T. and Kondor, R. Bhattacharyya and expected likelihood kernels. In *Learning theory and kernel machines*, pp. 57–71. Springer, 2003.

Jebara, T., Kondor, R., and Howard, A. Probability product kernels. *The Journal of Machine Learning Research*, 5: 819–844, 2004.

Jeddi, A., Shafiee, M. J., Karg, M., Scharfenberger, C., and Wong, A. Learn2Perturb: an end-to-end feature perturbation learning to improve adversarial robustness. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1241–1250, 2020.

Jiang, L., Zhou, Z., Leung, T., Li, L.-J., and Fei-Fei, L. MentorNet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In *ICML*, 2018.

Karaletsos, T. and Bui, T. D. Hierarchical Gaussian process priors for Bayesian neural network weights. *arXiv preprint arXiv:2002.04033*, 2020.

Karaletsos, T., Dayan, P., and Ghahramani, Z. Probabilistic meta-representations of neural networks. *arXiv preprint arXiv:1810.00555*, 2018.

Kingma, D. P., Salimans, T., and Welling, M. Variational dropout and the local reparameterization trick. In *NIPS*, pp. 2575–2583, 2015.

Kolchinsky, A. and Tracey, B. D. Estimating mixture entropy with pairwise distances. *Entropy*, 19(7), 2017. ISSN 1099-4300. doi: 10.3390/e19070361. URL https://www.mdpi.com/1099-4300/19/7/361.

Krizhevsky, A., Nair, V., and Hinton, G. CIFAR-10 and CIFAR-100 datasets. *URl: https://www. cs. toronto. edu/kriz/cifar. html*, 6(1):1, 2009.

Kwon, J., Kim, J., Park, H., and Choi, I. K. ASAM: Adaptive sharpness-aware minimization for scale-invariant learning of deep neural networks. *arXiv preprint arXiv:2102.11600*, 2021.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. Simple and scalable predictive uncertainty estimation using deep ensembles. In *NIPS*, pp. 6405–6416, 2017.

Le, Y. and Yang, X. S. Tiny ImageNet visual recognition challenge. 2015.

Li, B., Chen, C., Wang, W., and Carin, L. Certified adversarial robustness with additive noise. *arXiv preprint arXiv:1809.03113*, 2018.

Louizos, C. and Welling, M. Multiplicative normalizing flows for variational Bayesian neural networks. In *International Conference on Machine Learning*, pp. 2218–2227. PMLR, 2017.

MacKay, D. J. C. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3): 448–472, May 1992. ISSN 0899–7667.

MacKay, D. J. C. 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.

Maddox, W. J., Izmailov, P., Garipov, T., Vetrov, D. P., and Wilson, A. G. A simple baseline for Bayesian uncertainty in deep learning. In *Advances in Neural Information Processing Systems*, pp. 13153–13164, 2019.

Mandt, S., McInerney, J., Abrol, F., Ranganath, R., and Blei, D. Variational tempering. In *Artificial Intelligence and Statistics*, pp. 704–712. PMLR, 2016.

Medina, M. A., Olea, J. L. M., Rush, C., and Velez, A. On the robustness to misspecification of α-posteriors and their variational approximations. *arXiv preprint arXiv:2104.08324*, 2021.

Miller, J. W. and Dunson, D. B. Robust Bayesian Inference via Coarsening. *Journal of the American Statistical Association*, 114(527):1113–1125, July 2019. ISSN 0162-1459. doi: 10.1080/01621459.2018.1469995. URL https://doi.org/10.1080/01621459.2018.1469995. Publisher: Taylor & Francis.

Naeini, M. P., Cooper, G. F., and Hauskrecht, M. Obtaining well calibrated probabilities using Bayesian binning. In *AAAI*, 2015.

Neal, R. M. *Bayesian Learning for Neural Networks*. Lecture Notes in Statistics. Springer-Verlag, New York, 1996. ISBN 978-0-387-94724-2.

Nguyen, S., Nguyen, D., Nguyen, K., Than, K., Bui, H., and Ho, N. Structured dropout variational inference for Bayesian neural networks. In *NeurIPS*, 2021.

Ovadia, Y., Fertig, E., Ren, J., Nado, Z., Sculley, D., Nowozin, S., Dillon, J. V., Lakshminarayanan, B., and Snoek, J. Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift. *arXiv preprint arXiv:1906.02530*, 2019.

Pradier, M. F., Pan, W., Yao, J., Ghosh, S., and Doshi-Velez, F. Projected BNNs: Avoiding weight-space pathologies by learning latent representations of neural network weights. *arXiv preprint arXiv:1811.07006*, 2018.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

Sun, S., Zhang, G., Shi, J., and Grosse, R. Functional variational Bayesian neural networks. *arXiv preprint arXiv:1903.05779*, 2019.

Trinh, T., Kaski, S., and Heinonen, M. Scalable Bayesian neural networks by layer-wise input augmentation. *arXiv preprint arXiv:2010.13498*, 2020.

Vadera, M. P., Shukla, S. N., Jalaian, B., and Marlin, B. M. Assessing the adversarial robustness of Monte Carlo and distillation methods for deep Bayesian neural network classification. *arXiv preprint arXiv:2002.02842*, 2020.

Wenzel, F., Roth, K., Veeling, B. S., Świątkowski, J., Tran, L., Mandt, S., Snoek, J., Salimans, T., Jenatton, R., and Nowozin, S. How good is the Bayes posterior in deep neural networks really? *arXiv preprint arXiv:2002.02405*, 2020.

Zhang, R., Li, C., Zhang, J., Chen, C., and Wilson, A. G. Cyclical stochastic gradient MCMC for Bayesian deep learning. *International Conference on Learning Representations*, 2020.

A. Original ELBO derivation

Here we provide a detail derivation of the ELBO in Eq. (9). We assume a prior $p(\theta, \mathcal{Z}) = p(\theta)p(\mathcal{Z})$ for the parameters θ and latent variables $\mathcal{Z}$, and we assume a variational posterior $q_{\phi,\hat{\theta}}(\theta, \mathcal{Z}) = \delta(\theta - \hat{\theta})q_\phi(\mathcal{Z})$ where $\delta(.)$ is a Dirac delta distribution. We arrive at the ELBO in Eq. (9) by minimizing the KL divergence between the variational approximation and the true posterior with respect to the variational parameters $(\hat{\theta}, \phi)$:

$$\arg\min_{\phi,\hat{\theta}} \mathrm{KL}\left[q_{\phi,\hat{\theta}}(\theta, \mathcal{Z})||p(\theta, \mathcal{Z}|\mathcal{D})\right] \tag{24}$$

$$= \arg\min_{\phi,\hat{\theta}} \mathbb{E}_{q_{\phi,\hat{\theta}}(\theta,\mathcal{Z})}\left[\log q_{\phi,\hat{\theta}}(\theta, \mathcal{Z}) - \log p(\mathcal{D}|\theta, \mathcal{Z}) - \log p(\theta, \mathcal{Z}) + \log p(\mathcal{D})\right] \tag{25}$$

$$= \arg\min_{\phi,\hat{\theta}} \mathbb{E}_{q_\phi(\mathcal{Z})}\left[-\log p(\mathcal{D}|\hat{\theta}, \mathcal{Z})\right] + \mathrm{KL}\left[q_\phi(\mathcal{Z})||p(\mathcal{Z})\right] - \log p(\hat{\theta}) + \log p(\mathcal{D}) \tag{26}$$

$$= \arg\min_{\phi,\hat{\theta}} -\mathcal{L}(\hat{\theta}, \phi) \tag{27}$$

B. Tempered ELBO derivation

Here we show a connection between the tempered posterior with temperature $\tau = 1/(\gamma + 1)$ in Eq. (20) and the augmented ELBO in Section 4.1:

$$\arg\min_{\phi,\hat{\theta}} \frac{1}{\tau}\mathrm{KL}\left[q_{\phi,\hat{\theta}}(\theta, \mathcal{Z}) \,||\, p_\gamma(\theta, \mathcal{Z}|\mathcal{D})\right] \tag{28}$$

$$= \arg\min_{\phi,\hat{\theta}} \frac{1}{\tau}\mathbb{E}_{q_{\phi,\hat{\theta}}(\theta,\mathcal{Z})}\left[\log q_{\phi,\hat{\theta}}(\theta, \mathcal{Z}) - \tau \log p(\mathcal{D}|\theta, \mathcal{Z}) - \tau \log p(\theta, \mathcal{Z}) + \log p_\gamma(\mathcal{D})\right] \tag{29}$$

$$= \arg\min_{\phi,\hat{\theta}} \mathbb{E}_{q_{\phi,\hat{\theta}}(z,\theta)}\left[\frac{1}{\tau}\log q_{\phi,\hat{\theta}}(\theta, \mathcal{Z}) - \log p(\mathcal{D}|\theta, \mathcal{Z}) - \log p(\theta) - \log p(\mathcal{Z})\right] + \frac{1}{\tau}\log p_\gamma(\mathcal{D}) \tag{30}$$

$$= \arg\min_{\phi,\hat{\theta}} -\mathbb{E}_{q_\phi(\mathcal{Z})}\left[\log p(\mathcal{D}|\hat{\theta}, \mathcal{Z})\right] + \mathrm{KL}\left[q_\phi(\mathcal{Z})||p(\mathcal{Z})\right] - \gamma\mathbb{H}\left[q_\phi(\mathcal{Z})\right] - \log p(\hat{\theta}) + \frac{1}{\tau}\log p_\gamma(\mathcal{D}) \tag{31}$$

$$= \arg\min_{\phi,\hat{\theta}} -\mathcal{L}_\gamma(\hat{\theta}, \phi) + \log p_\gamma(\mathcal{D})^{\frac{1}{\tau}} \tag{32}$$

C. Derivation of layer-wise activation shifts due to input corruptions

Here we explain in detail the approximation of layer-wise activation shifts in Eq. (12). To simulate covariate shift, one can take an input $\mathbf{x}$ assumed to come from the same distribution as the training samples and apply a corruption $\mathbf{g}^0$ to form a shifted version $\mathbf{x}^c$ of $\mathbf{x}$:

$$\mathbf{x}^c \triangleq \mathbf{x} + \mathbf{g}^0(\mathbf{x}) \tag{33}$$

For instance, $\mathbf{x}$ could be an image and $\mathbf{g}^0$ can represent the shot noise corruption as seen in Hendrycks & Dietterich (2019). The corruption $\mathbf{g}^0(\mathbf{x})$ creates a shift in the activation of the first layer $\mathbf{f}^1$ which can be approximated using the first-order Taylor expansion:

$$\mathbf{g}^1(\mathbf{x}) = \mathbf{f}^1(\mathbf{x}^c) - \mathbf{f}^1(\mathbf{x}) \tag{34}$$

$$= \sigma\left(\mathbf{W}^1(\mathbf{x} + \mathbf{g}^0(\mathbf{x})) + \mathbf{b}^1\right) - \sigma\left(\mathbf{W}^1\mathbf{x} + \mathbf{b}^1\right) \tag{35}$$

$$\approx \mathbf{J}_\sigma\left[\mathbf{h}^1(\mathbf{x})\right]\left(\mathbf{W}^1\mathbf{g}^0(\mathbf{x})\right) \tag{36}$$

where $\mathbf{J}_\sigma = \partial\sigma/\partial\mathbf{h}$ denotes the Jacobian of the activation σ with respect to pre-activation outputs $\mathbf{h}$. Similarly, the approximation of the activation shift in the second layer is:

$$\mathbf{g}^2(\mathbf{x}) = \mathbf{f}^2(\mathbf{x}^c) - \mathbf{f}^2(\mathbf{x}) \tag{37}$$

$$= \sigma\left(\mathbf{W}^2\mathbf{f}^1(\mathbf{x}^c) + \mathbf{b}^2\right) - \sigma\left(\mathbf{W}^2\mathbf{f}^1(\mathbf{x}) + \mathbf{b}^2\right) \tag{38}$$

$$= \sigma\left(\mathbf{W}^2(\mathbf{f}^1(\mathbf{x}) + \mathbf{g}^1(\mathbf{x})) + \mathbf{b}^2\right) - \sigma\left(\mathbf{W}^2\mathbf{f}^1(\mathbf{x}) + \mathbf{b}^2\right) \tag{39}$$

$$\approx \mathbf{J}_\sigma\left[\mathbf{h}^2(\mathbf{x})\right]\left(\mathbf{W}^2\mathbf{g}^1(\mathbf{x})\right) \tag{40}$$

Table 1. The ALL-CNN-C architecture

ALL-CNN-C
Input 32×32 RGB images
3×3 conv. with 96 output filters, ReLU 3×3 conv. with 96 output filters, ReLU
3×3 conv. with 96 output filters and stride $r = 2$, ReLU
3×3 conv. with 192 output filters, ReLU 3×3 conv. with 192 output filters, ReLU
3×3 conv. with 192 output filters and stride $r = 2$, ReLU
3×3 conv. with 192 output filters, ReLU 1×1 conv. with 10 output filters, ReLU
Global average pooling 10-way softmax

Generally, one can approximate the shift in the output of the ℓ-th layer caused by $\mathbf{g}(\mathbf{x})$ as:

$$\mathbf{g}^{\ell}(\mathbf{x}) = \mathbf{f}^{\ell}(\mathbf{x}^{c}) - \mathbf{f}^{\ell}(\mathbf{x}) \approx \mathbf{J}_{\sigma}\left[\mathbf{h}^{\ell}(\mathbf{x})\right]\left(\mathbf{W}^{\ell}\mathbf{g}^{\ell-1}(\mathbf{x})\right) \tag{41}$$

D. Details on small-scale experiments

For the small-scale experiments in Section 3, we use the ALL-CNN-C architecture from Springenberg et al. (2014). We describe this architecture in Table 1. We train the model for 90 epochs, and only use the output latent variables and a posterior with 1 Gaussian component for this experiment

E. Additional visualization of outputs at each layer

In Section 3, we provide a PCA visualization of the outputs from the last layer of a node-based ALL-CNN-C BNN on one sample of CIFAR10. Here we also provide the same visualizations for the first two and the last two layers of the network. We use the same input image as Fig. 2.

F. Additional details on the experiments and hyperparameters

F.1. Approximation for the KL divergence with mixture variational posterior

We use a mixture of Gaussians (MoG) distribution with K equally-weighted components to provide a flexible approximation of the true posterior in the latent space:

$$q(\mathcal{Z}) = \frac{1}{K}\sum_{k=1}^{K} q_k(\mathcal{Z}) \tag{42}$$

$$q_k(\mathcal{Z}) = \prod_{\ell=1}^{L} q_{k,\ell}(\mathcal{Z}^{\ell}) \tag{43}$$

$$q_{k,\ell}(\mathcal{Z}^{\ell}) = \mathcal{N}(\boldsymbol{\mu}_{k,\ell}, \operatorname{diag}\boldsymbol{\sigma}^2_{k,\ell}). \tag{44}$$

where L is the number of layers. We use a Gaussian prior with global scalar variance s^2 for the latent prior,

$$p(\mathcal{Z}) = \mathcal{N}(\mathbf{1}, s^2 I). \tag{45}$$

The KL divergence decomposes into cross-entropy and entropy terms,

$$\mathrm{KL}[q(\mathcal{Z}) \,||\, p(\mathcal{Z})] = \mathbb{H}[q, p] - \mathbb{H}[q] = \frac{1}{K}\sum_{k=1}^{K} \mathbb{H}[q_k, p] - \mathbb{H}[q], \tag{46}$$

where the cross-entropy reduces into tractable terms $\mathbb{H}[q_k, p]$ for Gaussians. The mixture entropy $\mathbb{H}[q]$ remains intractable, but admits a lower bound (Kolchinsky & Tracey, 2017),

$$\mathbb{H}[q] \geq \frac{1}{K}\sum_{k=1}^{K} \mathbb{H}[q_k] - \frac{1}{K}\sum_{k=1}^{K} \log\left(\frac{1}{K}\sum_{r=1}^{K} \mathrm{BC}(q_k, q_r)\right) \triangleq \widehat{\mathbb{H}}[q] \tag{47}$$

where

$$\mathrm{BC}(q, q') = \int \sqrt{q(\mathbf{z})}\sqrt{q'(\mathbf{z})} d\mathbf{z} \qquad \leq 1 \tag{48}$$

is the Bhattacharyya kernel of overlap between two distributions (Jebara & Kondor, 2003; Jebara et al., 2004), and has a closed form solution for a pair of Gaussians q, q'. The Bhattacharyya kernel has the convenient normalization property $\mathrm{BC}(q, q) = 1$. The lower bound considers unary and pairwise component entropies.

F.2. Experimental details and hyperparameters

We actually maximizes the following objective to train the node-based BNNs on large-scale experiments:

$$\mathcal{L}_{\gamma,\beta}(\hat{\theta}, \phi) = \mathbb{E}_{q_\phi(\mathcal{Z})}\Big[\log p(\mathcal{D}|\hat{\theta}, \mathcal{Z})\Big] + \log p(\hat{\theta}) + \beta\left(-\mathbb{H}\big[q_\phi(\mathcal{Z}), p(\mathcal{Z})\big] + (\gamma+1)\widehat{\mathbb{H}}\big[q_\phi(\mathcal{Z})\big]\right) \tag{49}$$

which is the augmented ELBO in Eq. (19) with additional coefficient β for the cross-entropy and variational entropy term. We also replace the intractable mixture entropy $\mathbb{H}[q]$ with its tractable lower bound $\widehat{\mathbb{H}}[q]$ presented in Eq. (47). During training, we will anneal β from 0 to 1. We found this to have ease optimization and produce better final results. For all experiments, we estimate the expected log-likelihood in the loss function using 4 samples.

For all the experiments on CIFAR10/CIFAR100, we run each experiment for 300 epochs, where we increase β from 0 to 1 for the first 200 epochs. We use SGD as our optimizer, and we use a weight decay of 0.0005 for the parameters θ. We use a batch size of 128. For all the experiments on TINYIMAGENET, we run each experiment for 150 epochs, where we increase β from 0 to 1 for the first 100 epochs. We use a batch size of 256. Bellow, we use λ_1 and λ_2 to denote the learning rate of the parameters θ and ϕ respectively.

For VGG16, we set the initial learning rate $\lambda_1 = \lambda_2 = 0.05$, and we decrease λ_1 linearly from 0.05 to 0.0005 from epoch 150 to epoch 270, while keeping λ_2 fixed throughout training. We initialize the standard deviations with $\mathcal{N}^+(0.30, 0.02)$ and set the standard deviation of the prior to 0.30.

For RESNET18, we set the initial learning rate $\lambda_1 = \lambda_2 = 0.10$, and we decrease λ_1 linearly from 0.10 to 0.001 from epoch 150 to epoch 270, while keeping λ_2 fixed throughout training. We initialize the standard deviations with $\mathcal{N}^+(0.40, 0.02)$ and set the standard deviation of the prior to 0.40.

For PREACTRESNET18, we set the initial learning rate $\lambda_1 = \lambda_2 = 0.10$, and we decrease λ_1 linearly from 0.10 to 0.001 from epoch 75 to epoch 135, while keeping λ_2 fixed throughout training. We initialize the standard deviations with $\mathcal{N}^+(0.30, 0.02)$ and set the standard deviation of the prior to 0.30.

F.3. Runtime

We report the average running times of different methods in Table 2. We used similar number of epochs for all methods in each experiment. All experiment were performed on one Tesla V100 GPU. Overall, node BNNs took 4 times longer to train than SWAG since we use 4 Monte Carlo samples per training sample to estimate the expected log-likelihood in the γ-ELBO. ASAM took 2 times longer to train than SWAG since they require two forward-backward passes per minibatch.

G. Additional benchmark results

Here we include the benchmark results of VGG16 on CIFAR10 and CIFAR100 in Fig. 13. We also include Fig. 14 and Fig. 15 as larger versions of Fig. 10 and Fig. 11.

Table 2. Average running times of different methods measured in seconds. All experiments were performed on one Tesla V100 GPU.

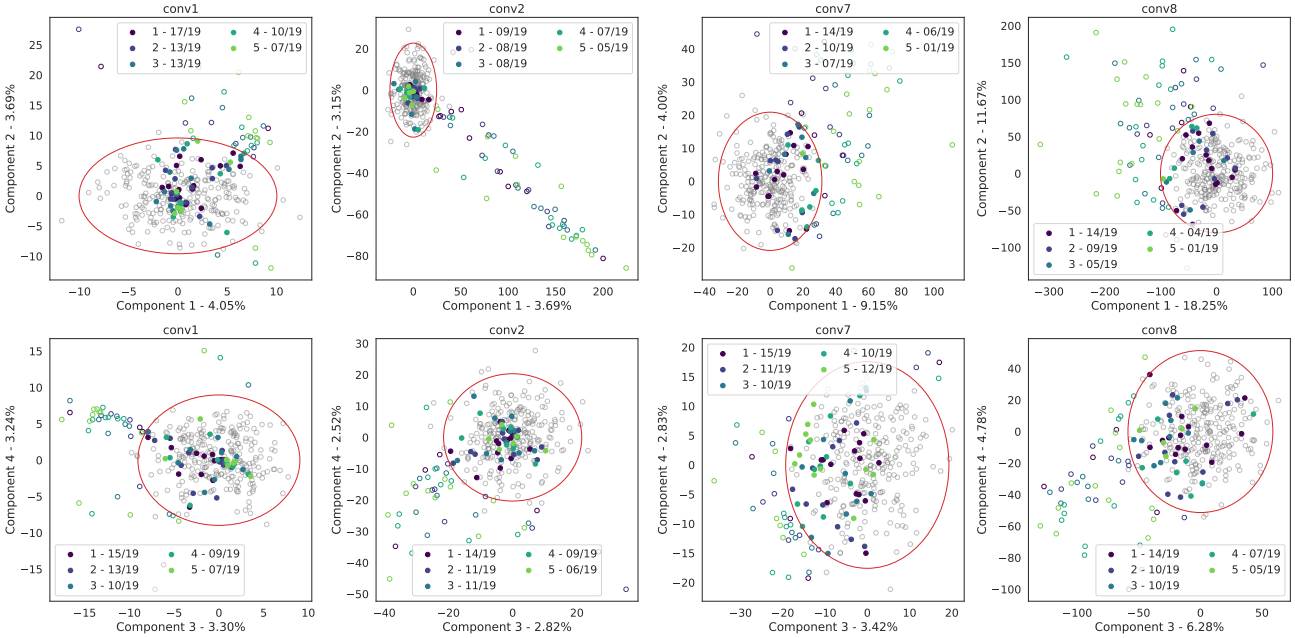
Model	Dataset	Node-BNN	SWAG	ASAM
VGG16	CIFAR100	13274	3384	6870
	CIFAR10	12941	3251	6539
ResNet18	CIFAR100	18093	4528	9086
	CIFAR10	17733	4474	8921
PreActResNet18	TinyImagenet	54892	13830	26564

H. The evolution of variational entropy during training

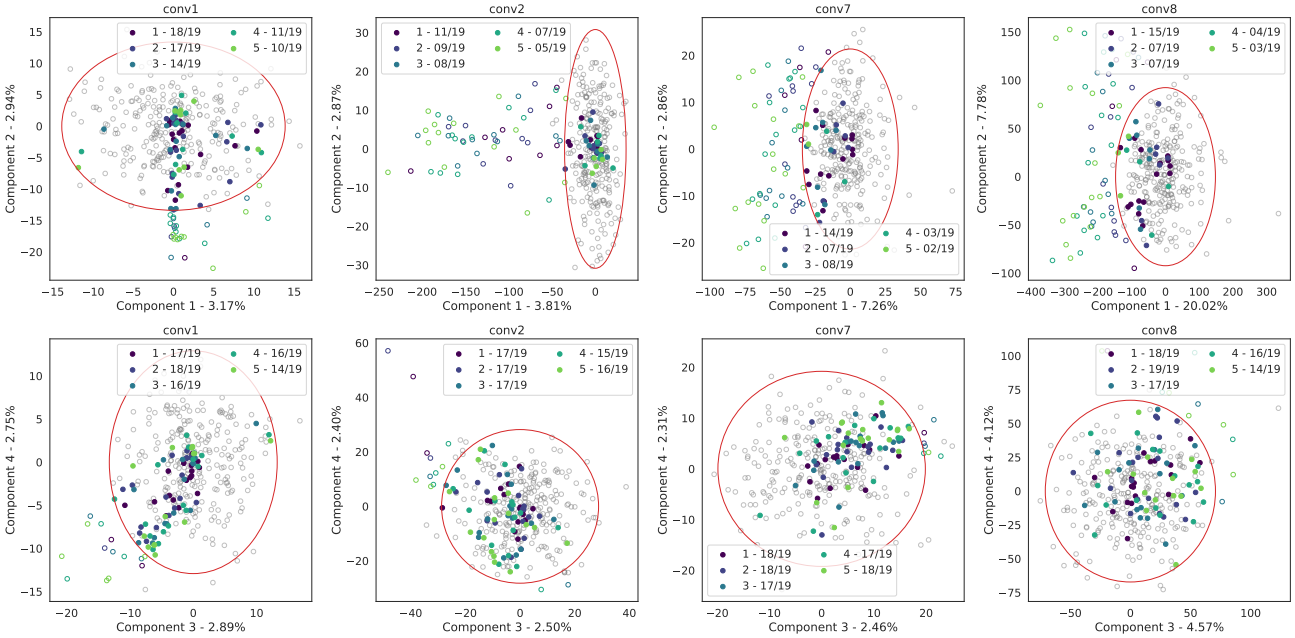
We visualize the progression of the variational entropy when trained using the original ELBO (without the γ-entropy term) under different settings in Figs. 16-19. We can observe the typical behaviour of variational inference that it tends to reduce the entropy of the variational posterior over time.

I. Additional results on the effect of γ on performance of node-based BNNs

Here we include Figs. 20-24 to show the effect of γ on performance of node-based BNNs under different architectures and datasets.



(a) The outputs of the first two and last two layer in $\mathcal{M}_{16}$. $q(\mathcal{Z})$ is a single Gaussian with the standard deviations initialized from a half normal $\mathcal{N}^+(0.16, 0.02)$



(b) The outputs of the first two and last two layer in $\mathcal{M}_{32}$ whose posterior $q(\mathcal{Z})$ is a single Gaussian with the standard deviations initialized from a half normal $\mathcal{N}^+(0.32, 0.02)$.

Figure 12. PCA plots of the outputs for the first two and last two layers on a node-based ALL-CNN-C BNN with respect to one image from CIFAR10. Grey unfilled circle are samples from the output distribution induced by the latent variables, while the red ellipse is the 99 percentile of this distribution. The color circle represents the expected output $\hat{\mathbf{f}}^\ell$ under input corruptions, where we fill the circle if it lies within the ellipse. Each axis label is the component index and its explained variance ratio. In the legend, we denote the severity of the corruptions and the ratio between number of points lie within the 99 percentile of the output distribution and the total number of corruption types. We use the corruptions from (Hendrycks & Dietterich, 2019) containing 5 levels of severity and 19 types. For the model with larger $H[q(\mathcal{Z})]$ in 12b, the number of points lie within the ellipse is higher than the model with smaller $H[q(\mathcal{Z})]$ in 12a.

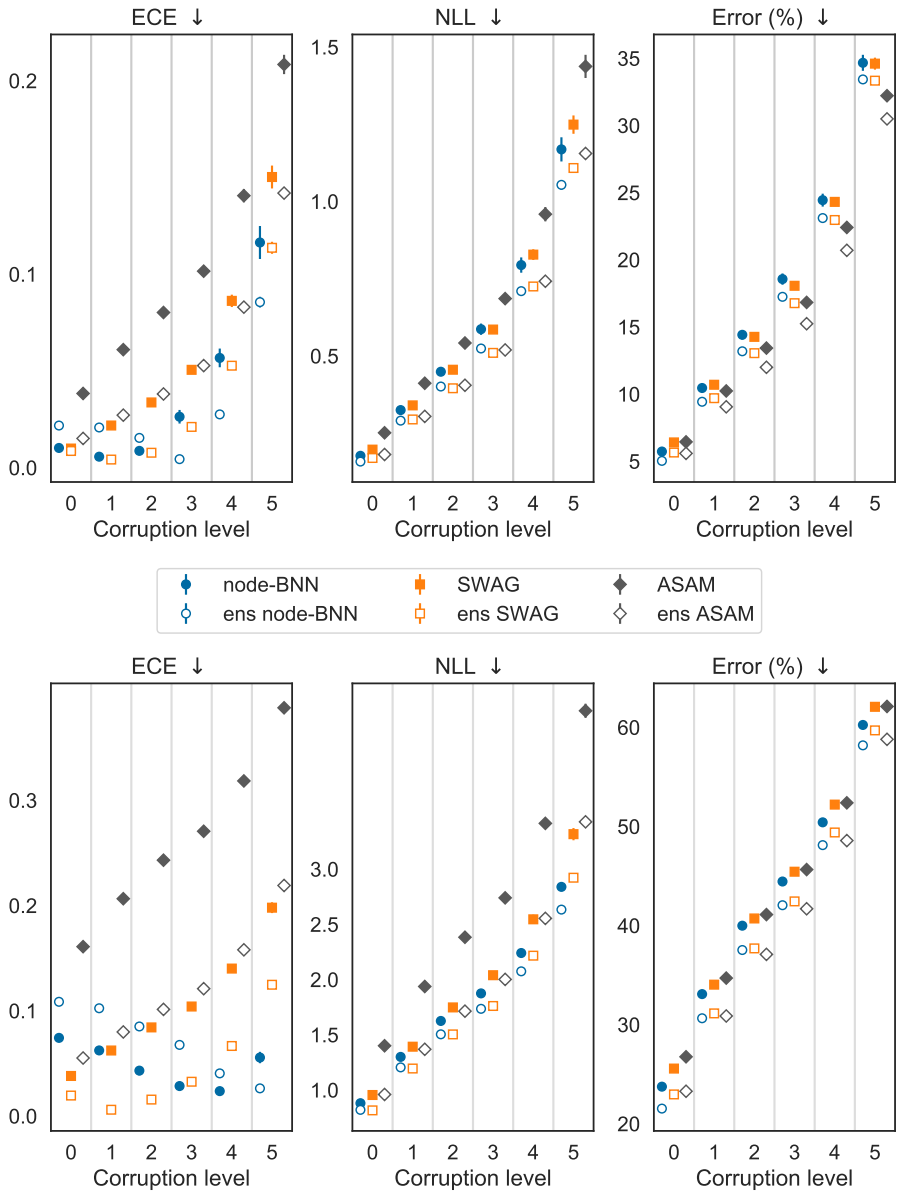


Figure 13. Results of VGG16 on CIFAR10 **(top)** and CIFAR100 **(bottom)**. We use $K = 4$ and only the latent output variables for node-based BNNs. We plot ECE, NLL and error for different corruption levels, where level 0 indicates no corruption. We report the average performance over 19 corruption types for level 1 to 5. We denote the ensemble of a method using the shorthand *ens* in front of the name. Each result is the average over 25 runs for non-ens versions and 5 runs for ens versions. The error bars represent the standard deviations across different runs. Node-based BNNs and their ensembles (blue) perform best in term of ECE and NLL on OOD data, while having similar accuracy to other methods.

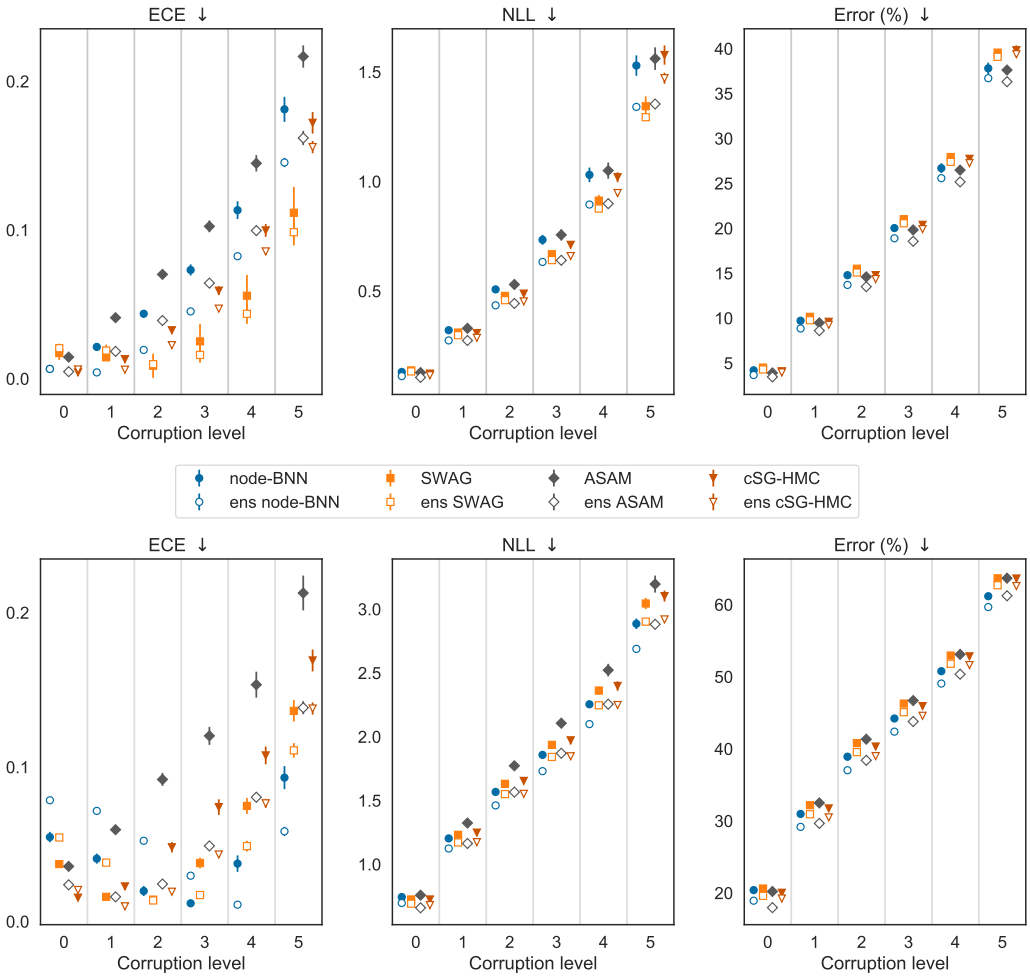


Figure 14. Results of RESNET18 on CIFAR10 **(top)** and CIFAR100 **(bottom)**. We use $K = 4$ and only the latent output variables for node-based BNNs. We plot ECE, NLL and error for different corruption levels, where level 0 indicates no corruption. We report the average performance over 19 corruption types for level 1 to 5. We denote the ensemble of a method using the shorthand *ens* in front of the name. Each result is the average over 25 runs for *non-ens* versions and 5 runs for *ens* versions. The error bars represent the standard deviations across different runs. Node-based BNNs and their ensembles (blue) perform best across all metrics on OOD data of CIFAR100, while having competitive results on CIFAR10.

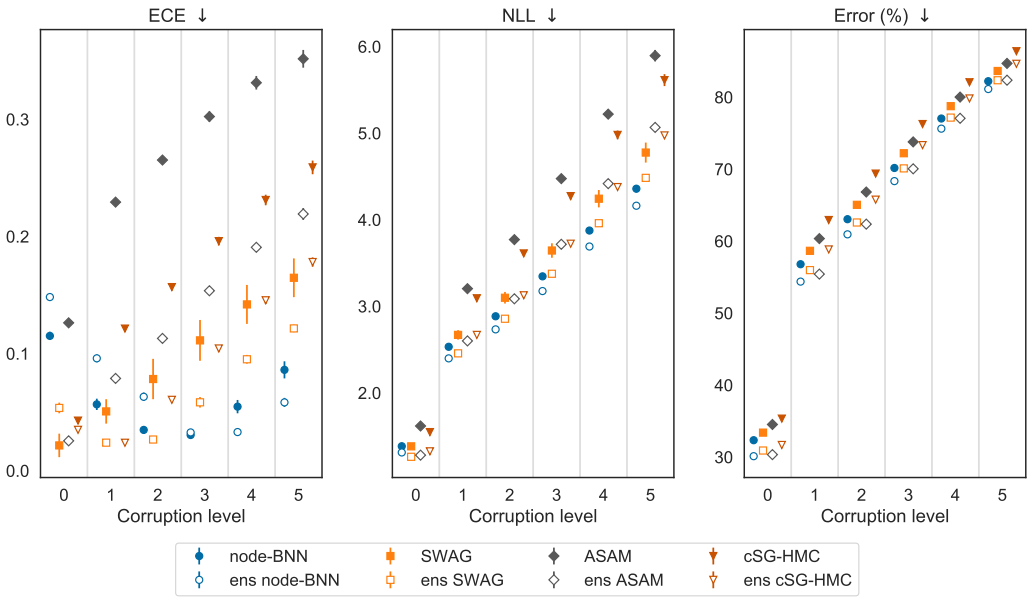


Figure 15. Results of PREACTRESNET18 on TINYIMAGENET. We use $K = 4$ and only the latent output variables for node-based BNNs. We plot ECE, NLL and error for different corruption levels, where level 0 indicates no corruption. We report the average performance over 19 corruption types for level 1 to 5. We denote the ensemble of a method using the shorthand *ens* in front of the name. Each result is the average over 25 runs for *non-ens* versions and 5 runs for *ens* versions. The error bars represent the standard deviations across different runs. Node-based BNNs and their ensembles (blue) perform best accross all metrics on OOD data, while having competitive performance on ID data.

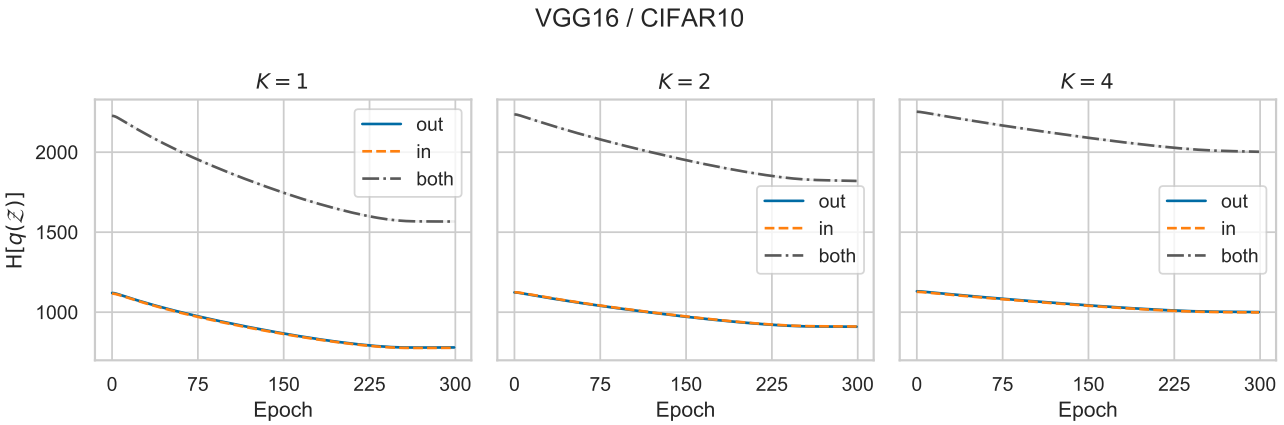


Figure 16. The evolution of entropy during training for VGG16 / CIFAR10 when trained using the original ELBO. Each result is averaged over 5 runs. Each error bar represents one standard deviation but it is too small to be seen.

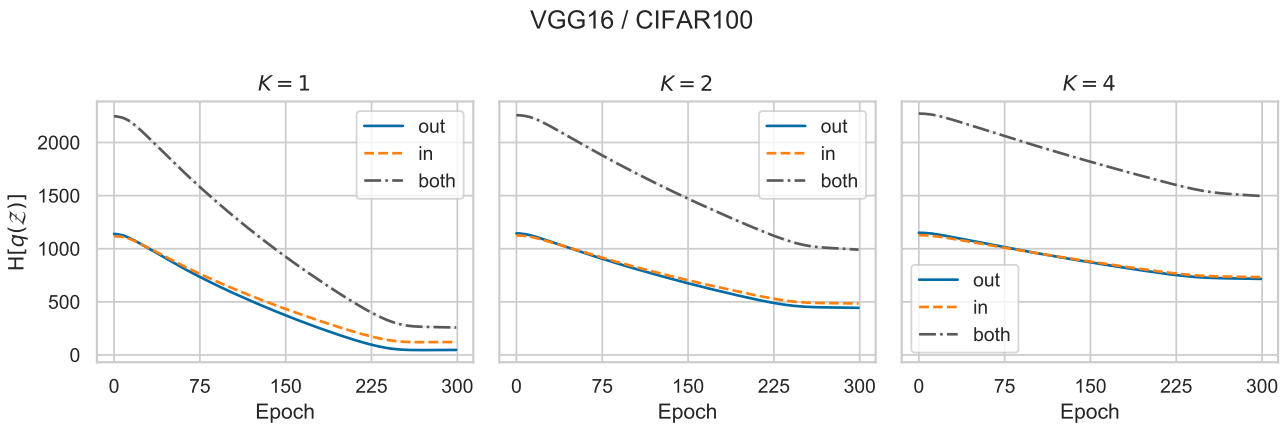


Figure 17. The evolution of entropy during training for VGG16 / CIFAR100 when trained using the original ELBO. Each result is averaged over 5 runs. Each error bar represents one standard deviation but it is too small to be seen.

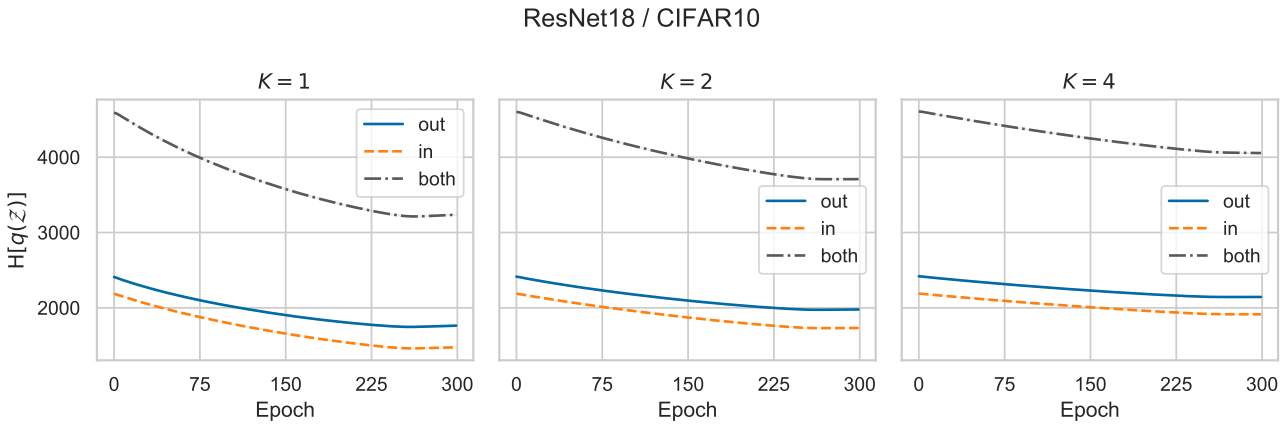


Figure 18. The evolution of entropy during training for RESNET18 / CIFAR10 when trained using the original ELBO. Each result is averaged over 5 runs. Each error bar represents one standard deviation but it is too small to be seen.

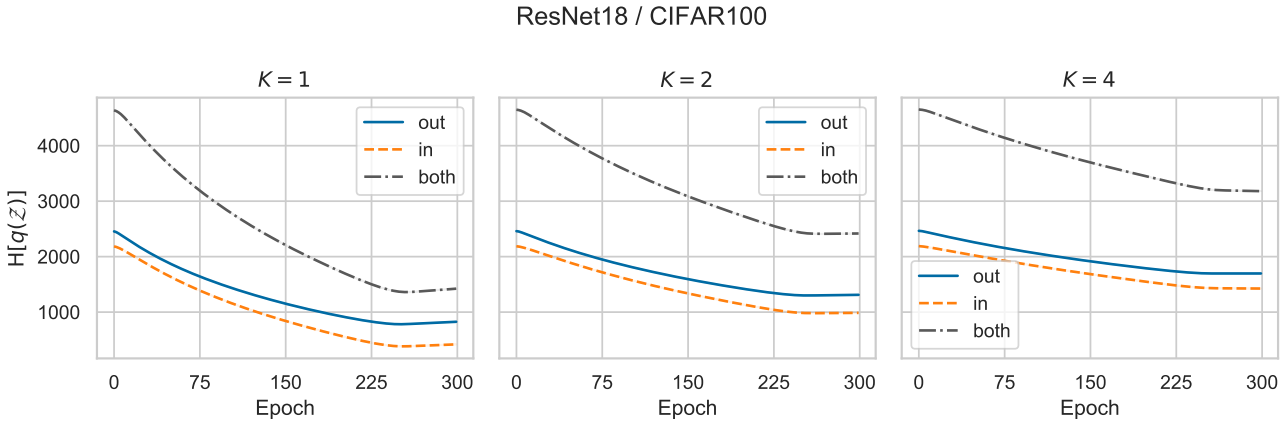


Figure 19. The evolution of entropy during training for RESNET18 / CIFAR100 when trained using the original ELBO. Each result is averaged over 5 runs. Each error bar represents one standard deviation but it is too small to be seen.

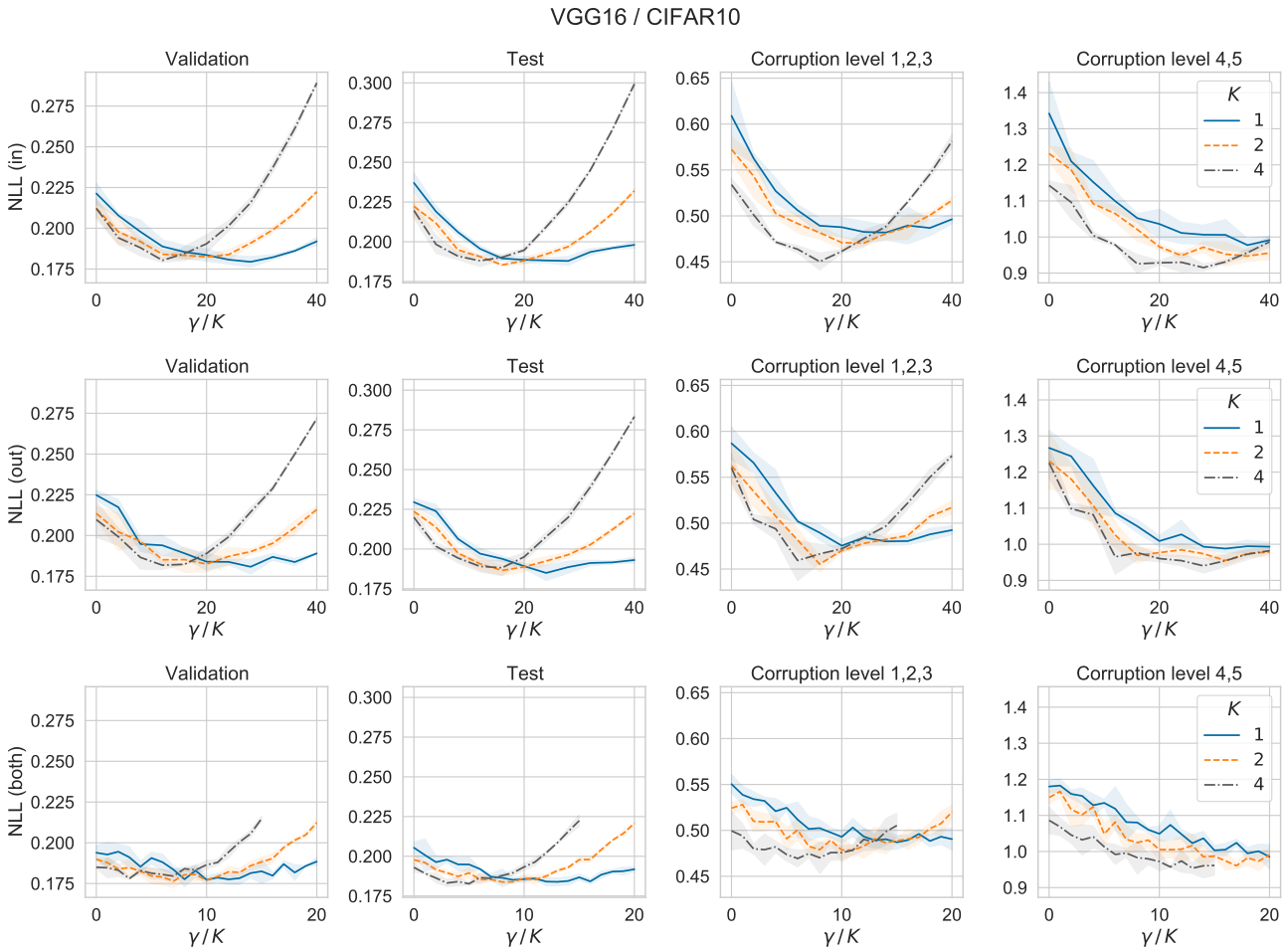


Figure 20. Results of VGG16 on CIFAR10 under different γ value. K is the number of components. Each row corresponds a different latent variable structure. We report the mean and standard deviation over 5 runs for each result.

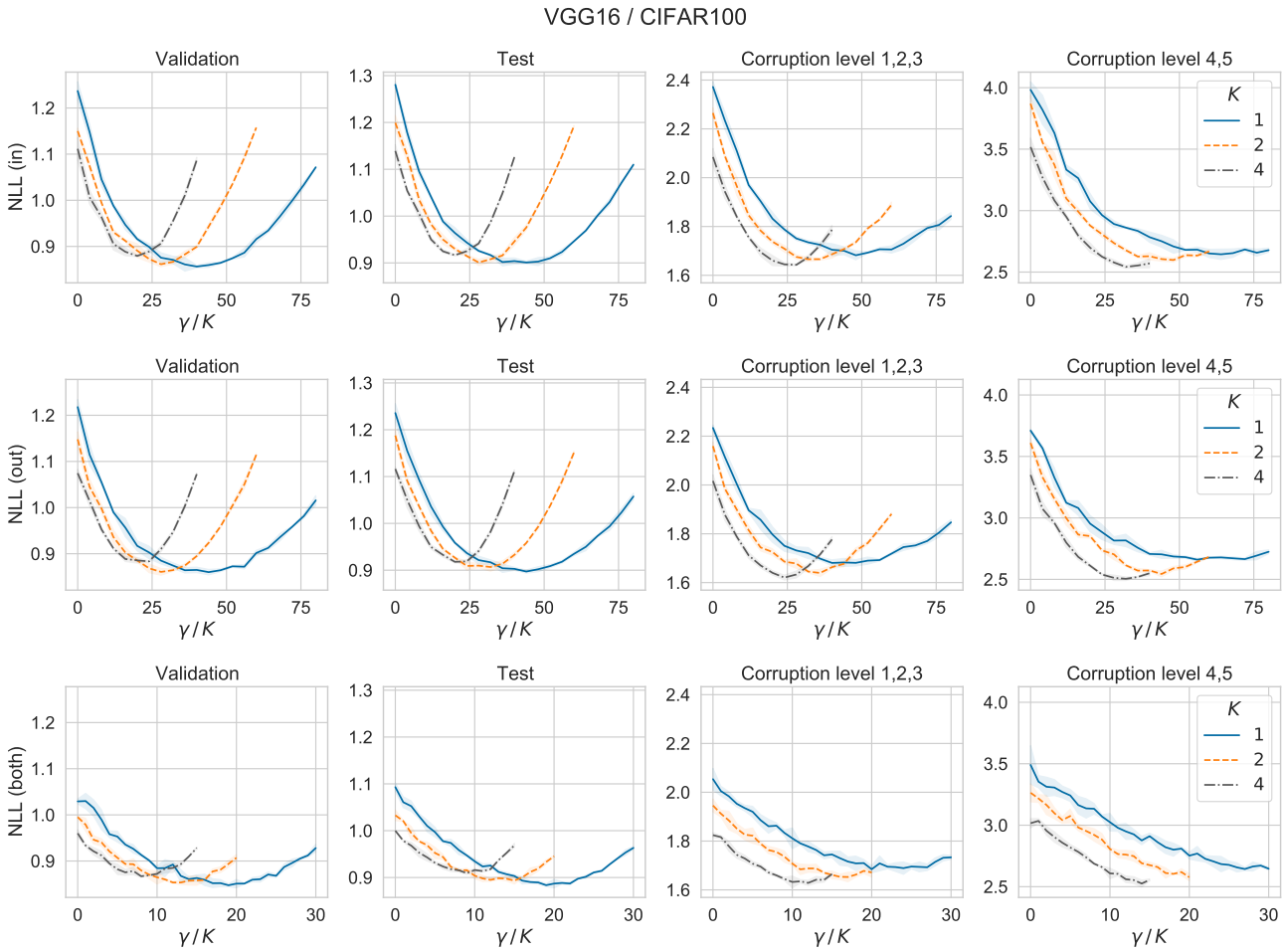


Figure 21. Results of VGG16 on CIFAR100 under different γ value. K is the number of components. Each row corresponds a different latent variable structure. We report the mean and standard deviation over 5 runs for each result.

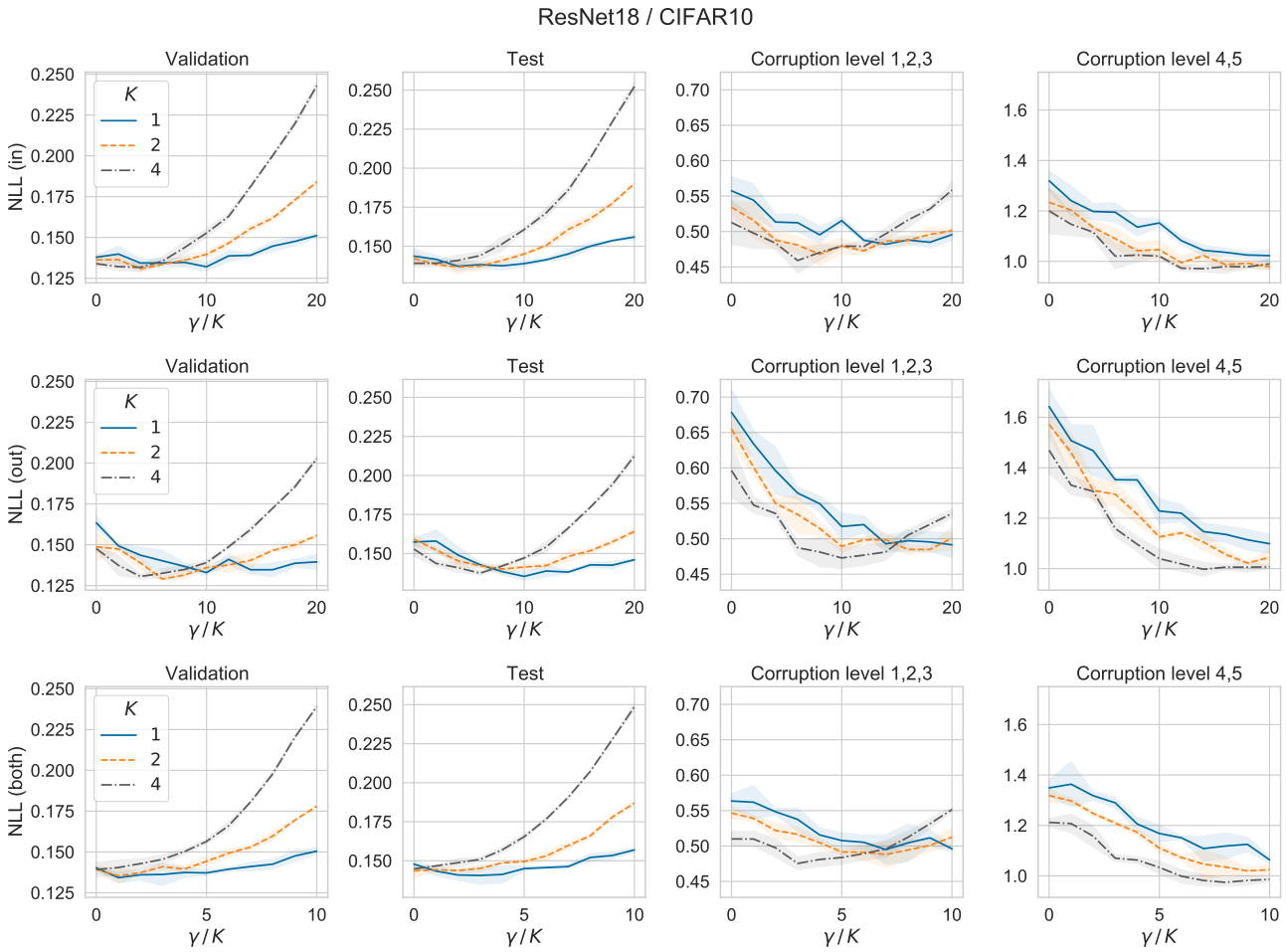


Figure 22. Results of RESNET18 on CIFAR10 under different γ value. K is the number of components. Each row corresponds a different latent variable structure. We report the mean and standard deviation over 5 runs for each result.

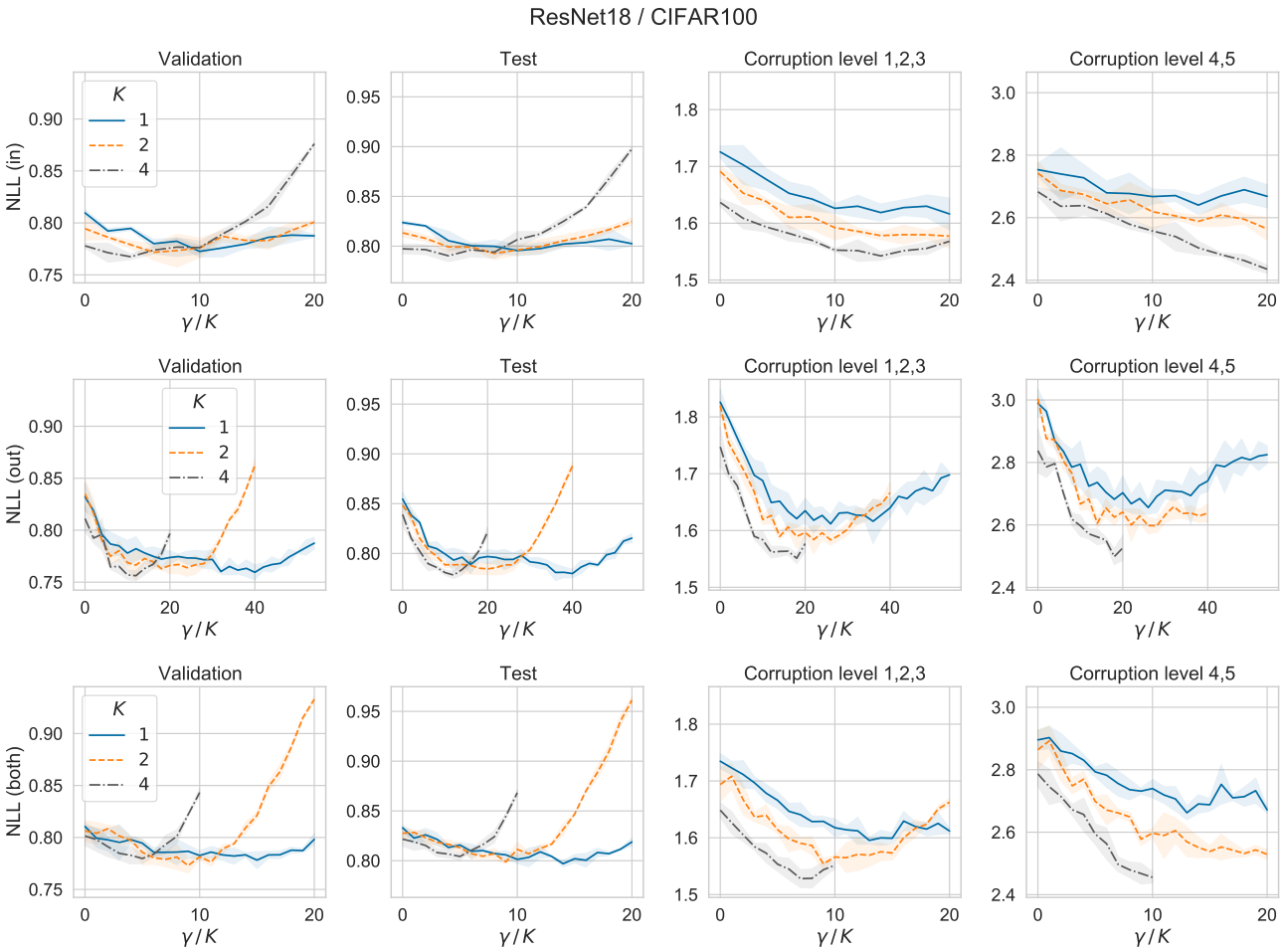


Figure 23. Results of RESNET18 on CIFAR100 under different γ value. K is the number of components. Each row corresponds a different latent variable structure. We report the mean and standard deviation over 5 runs for each result.

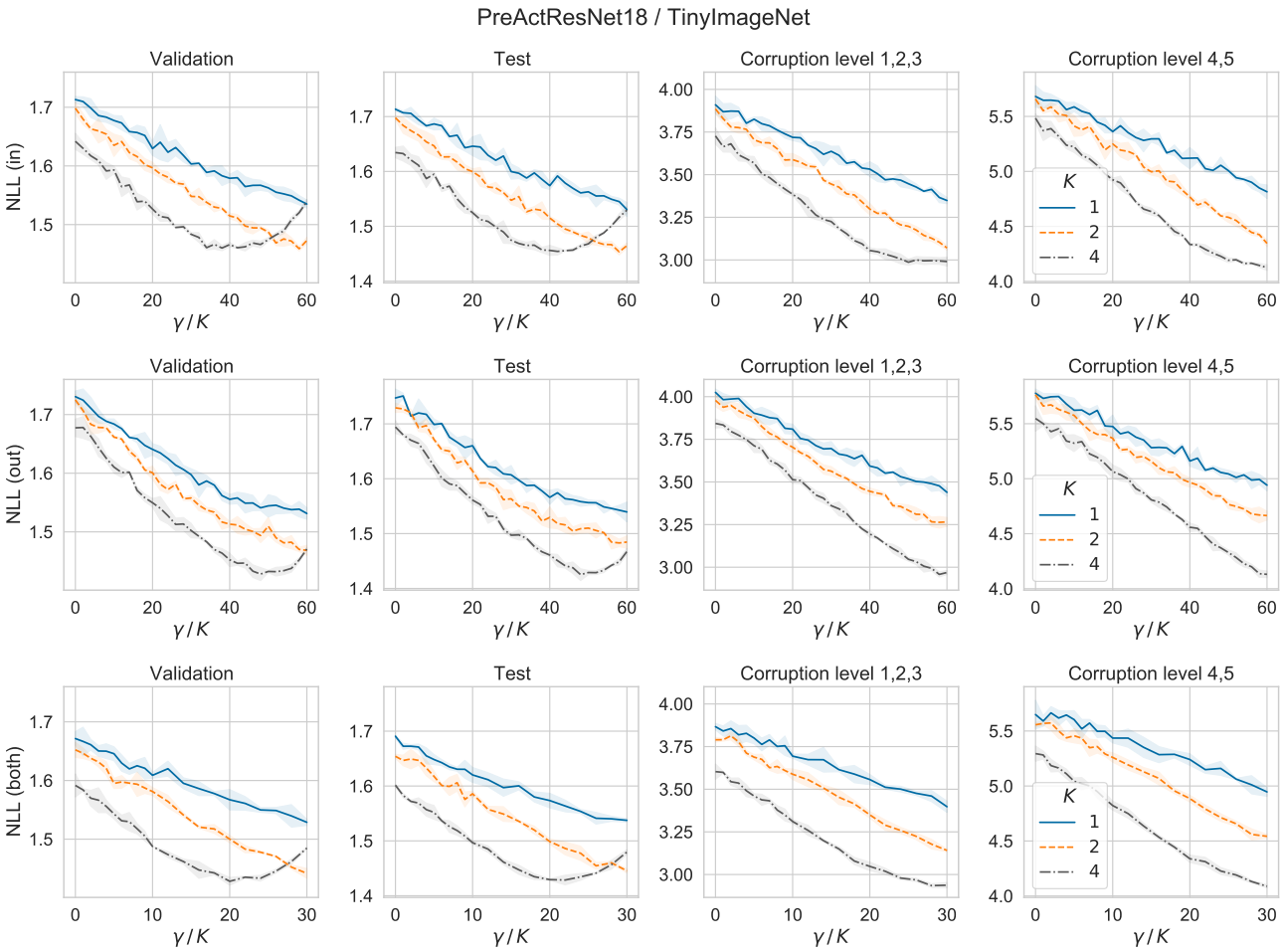


Figure 24. Results of PREACTRESNET18 on TINYIMAGENET under different γ value. K is the number of components. Each row corresponds a different latent variable structure. We report the mean and standard deviation over 5 runs for each result.