Variational Nested Dropout (Appendix)

Yufei Cui[1], Yu Mao[1], Ziquan Liu[1], Qiao Li[2], Antoni B. Chan[1], Xue Liu[3], Tei-Wei Kuo[4], Chun Jason Xue[1]
yufeicui92@gmail.com, yfcui@ibingli.com
[1]Department of Computer Science, City University of Hong Kong
[2]School of Informatics, Xiamen University
[3]School of Computer Science, McGill University
[4]Department of Computer Science and Information Engineering, National Taiwan University

A Derivation and Proofs

A.1 Derivation of Properties

Property 1. If $\mathbf{c} \sim \text{Gumbel\_softmax}(\tau, \beta, \epsilon_z)$[1], then $z_i = 1 - \text{cumsum}'_i(\mathbf{c})$, where $\mathbf{e}$ is a K-dimensional vector of ones, and $\text{cumsum}'_i(\mathbf{c}) = \sum_{j=0}^{i-1} c_j$. $c_0 \coloneqq 1$. ϵ_z is a standard uniform variable.

We show that using the sampling process in Property 1 recovers produce the Downhill random variable. We assume $\mathbf{c}$ follows a Gumbel softmax distribution [16], [39] which has the following form.

$$p(c_1, \dots, c_K) = \Gamma(K)\tau^{K-1}(\sum_{i=1}^{K} \pi_i / c^{\tau})^{-K} \prod_{i=1}^{K} (\pi_i / c^{\tau+1}) \tag{A1}$$

We apply the transformation $T_i(\cdot) = \mathbf{e}_i - \text{cumsum}'_i(\cdot)$ to the variable $\mathbf{c}$. $\mathbf{z} = T(\mathbf{c}) = \mathbf{e} - \text{cumsum}'_i(\mathbf{c})$

To obtain the distribution of $p(\mathbf{z})$, we apply the change of variables formula on $\mathbf{c}$.

$$p(\mathbf{z}) = p(T^{-1}(\mathbf{z}))\left|\det(\partial \frac{T^{-1}(\mathbf{z})}{\partial \mathbf{z}})\right| \tag{A2}$$

$$p(z_{1:K}) = p(T^{-1}(z_{1:K}))\left|\det(\frac{\partial T^{-1}(z_{1:K})}{\partial z_{1:K}})\right| \tag{A3}$$

From the definition of $T(\cdot)$, we can obtain $T_i^{-1}(\mathbf{z}) = z_{i-1} - z_i$, and its Jacobian is

$$\frac{\partial T^{-1}(z_{1:K})}{\partial z_{1:K}} = \begin{bmatrix} -1 & 0 & \dots & 0 & 0 \\ 1 & -1 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & \dots & 1 & -1 \end{bmatrix} \tag{A4}$$

Thus, $|\det(\frac{\partial T^{-1}(z_{1:K})}{\partial z_{1:K}})| = 1$. Finally, we have

$$p(z_{1:K}) = p(T_{1:K}^{-1}(\mathbf{z})) = \Gamma(K)\tau^{K-1}(\sum_{i=1}^{K} \frac{\pi_i}{(z_{i-1} - z_i)^{\tau}})^{-K} \prod_{i=1}^{K} (\frac{\pi_i}{(z_{i-1} - z_i)^{\tau+1}}). \tag{A5}$$

Property 2. When $\tau \to 0$, sampling from the Downhill distribution reduces to discrete sampling, where the sample space is the set of ordered mask vectors $\mathcal{V}$. The approximation of the Downhill distribution to the Bernoulli chain can be calculated in closed-form.

As shown in [38], [23], when $\tau \to 0$, the Gumbel softmax transformation corresponds to an argmax operation that generates a one-hot vector:

$$\mathbf{c} = \text{one\_hot}(\text{argmax}_i(g_i + \log \beta_i)), \tag{A6}$$

where the relative order is preserved.

Say a sample $\mathbf{c}^* \sim \text{Gumbel\_softmax}(\boldsymbol{\beta}, \tau \to 0)$, with b-th entry being one and the remaining entries being 0. The defined transformation generates $\text{cumsum}'(\mathbf{c}^*) = [\underbrace{0, \dots, 0}_{b} \underbrace{1, \dots, 1}_{K-b}]$. Thus, $\mathbf{z}^* = \mathbf{e} - \text{cumsum}'(\mathbf{c}^*) = [\underbrace{1, \dots, 1}_{b} \underbrace{0, \dots, 0}_{K-b}]$. It is easy to see the transformation $t'(\cdot)$ is surjective function where $t' : \{\text{one\_hot}(i)\}_{i=1}^{K} \to \mathcal{V}, t'(\mathbf{c}) = \mathbf{e} - \text{cumsum}'(\mathbf{c})$. $\mathcal{V}$ is exactly the set of ordered mask defined in Section 2.2.

Thus, we can calculate the approximation of Downhill variable to the Bernoulli chain,

$$\text{KL}[q(\mathbf{z})||p(\mathbf{z})] = \sum_{j=1}^{K} q(\mathbf{v}_j) \log \frac{q(\mathbf{v}_j)}{p(\mathbf{v}_j)}, \tag{A7}$$

1. For Gumbel-softmax sampling, we first draw $g_1 \dots g_K$ from $\text{Gumbel}(0, 1)$, then calculate $c_i = \text{softmax}(\frac{\log(\beta_i) + g_i}{\tau})$. The samples of $\text{Gumbel}(0, 1)$ can be obtained by first drawing $\epsilon_z \sim \text{Uniform}(0, 1)$ then computing $g = -\log(-\log(\epsilon_z))$.

where $q(\mathbf{v}_j) = \beta_j, p(\mathbf{v}_j) = (1-\pi_{j+1})\prod_{k=1}^{j}\pi_k$ (See Appx A.2). The KL divergence in (A7) minimized to 0 when $\beta_j = (1-\pi_{j+1})\prod_{k=1}^{j}, \forall j \in [1,\dots,K]$.

A.2 Probability of ordered masks

Recall the formulation of the Bernoulli chain:

$$\begin{aligned} &p(z_1=1)=\pi_1, && p(z_1=0)=1-\pi_1, \\ &p(z_i=1|z_{i-1}=1)=\pi_i, && p(z_i=0|z_{i-1}=1)=1-\pi_i, \\ &p(z_i=1|z_{i-1}=0)=0, && p(z_i=0|z_{i-1}=0)=1, \end{aligned} \tag{A8}$$

It is observed, there is a chance that $z_i=1$ only when $z_{i-1}=1$. However, if $z_i=0$, then $z_j=0$ for $j>i$. Thus,

$$p(\mathbf{z}=\mathbf{v}_j) = (1-\pi_{j+1})\prod_{k=1}^{j}\pi_k, \tag{A9}$$

where $j+1$ is the index of first zero. For convenience, we define $\pi_{K+1}=0$ as $p(\mathbf{z}=\mathbf{v}_K)=\prod_{k=1}^{K}\pi_k$, which means all nodes are kept.

A.3 Posterior approximation - Φ_2

Define $\mathbf{w}_j=[w_{1j},\dots,w_{Dj}]$ as the j-th column of $\mathbf{W}$, and $q_{\boldsymbol{\theta}}(\mathbf{w}_j|z_j=k)=q_{\boldsymbol{\theta}}(\mathbf{w}_j|z_j^k)$ where $k\in\{0,1\}$. The term Φ_2 of (9) is

$$\begin{aligned} \Phi_2 &= \sum_{\mathbf{z}\in\mathcal{V}} q_{\boldsymbol{\beta}}(\mathbf{z}) \sum_j \int_{\mathbf{w}_j} q_{\boldsymbol{\theta}}(\mathbf{w}_j|z_j^k)\log\frac{q_{\boldsymbol{\theta}}(\mathbf{w}_j|z_j^k)}{p(\mathbf{w}_j|z_j^k)}d\mathbf{w}_j \\ &= \sum_{\mathbf{z}\in\mathcal{V}} q_{\boldsymbol{\beta}}(\mathbf{z}) \sum_i\sum_j \int_{w_{ij}} q_{\boldsymbol{\theta}}(w_{ij}|z_j^k)\log\frac{q_{\boldsymbol{\theta}}(w_{ij}|z_j^k)}{p(w_{ij}|z_j^k)}dw_{ij} \\ &= \sum_{\mathbf{z}\in\mathcal{V}} q_{\boldsymbol{\beta}}(\mathbf{z}) \sum_{i,j} \mathrm{KL}[q_{\boldsymbol{\theta}}(w_{ij}|z_j^k)||p(w_{ij}|z_j^k)]. \end{aligned} \tag{A10}$$

Note that the term inside the integration over w_{ij} is the KL divergence between the univariate conditional density in the prior and the posterior, with $z_j=0$ or $z_j=1$. Define $K_{ij}^k(\boldsymbol{\theta})$ as the KL of w_{ij} for component $k\in\{0,1\}$. The term Φ_2 can then be re-organized as

$$\begin{aligned} \Phi_2 =& q(\mathbf{z}=\mathbf{v}_1)(\sum_i K_{i1}^1(\boldsymbol{\theta}) + \sum_{j=2}^{D}\sum_i K_{ij}^0(\boldsymbol{\theta})) \\ &+ q(\mathbf{z}=\mathbf{v}_2)(\sum_{j=1}^{2}\sum_i K_{ij}^1(\boldsymbol{\theta}) + \sum_{j=3}^{D}\sum_i K_{ij}^0(\boldsymbol{\theta})) \\ &+ \dots \end{aligned} \tag{A11}$$

There are totally D^2d terms, which potentially causes a large computation cost in every epoch. Consider the matrices $\boldsymbol{\kappa}_{\boldsymbol{\theta}}^0 = [K_{ij}^0(\boldsymbol{\theta})]_{ij} \in \mathbb{R}^{d\times D}$ and $\boldsymbol{\kappa}_{\boldsymbol{\theta}}^1 = [K_{ij}^1(\boldsymbol{\theta})]_{ij} \in \mathbb{R}^{d\times D}$, which are easily computed by applying the KL function element-wise. The term Φ_2 is then expressed as

$$\Phi_2 = \mathbf{e}^T\boldsymbol{\kappa}_{\boldsymbol{\theta}}^0(\mathbf{J}-\mathbf{J}_L)^T\boldsymbol{\beta} + \mathbf{e}^T\boldsymbol{\kappa}_{\boldsymbol{\theta}}^1\mathbf{J}_L^T\boldsymbol{\beta}, \tag{A12}$$

where $\mathbf{e}$ is a vector of 1s, $\mathbf{J}$ is a matrix of 1s and $\mathbf{J}_L$ is a lower triangular matrix with each element being 1. Then the calculation in (A12) can be easily parallelize with a modern computation library.

A.4 ℓ-0 regularization

We consider the case when the prior over each weight is a spike-and-slap distribution, i.e., $p(w_{ij}|z_j=0)=\delta(w_{ij})$ and $p(w_{ij}|z_j=1)=\mathcal{N}(w_{ij}|0,1)$, using the notation in Section 3.3. The posterior is also in this form. The derivations of KL term $\boldsymbol{\kappa}$ remain unchanged as it makes nothing but mean-field assumption on the weight prior. With Φ_1 and Φ_2, the objective (7) can be re-organized as

$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\beta}}^{\mathrm{SGVB}} \simeq L_{\mathcal{D}}^{\mathrm{SGVB}}(\boldsymbol{\theta},\boldsymbol{\beta}) - \sum_{j=1}^{D}\mathrm{KL}[q_{\boldsymbol{\beta}}(\mathbf{v}_i)||p(\mathbf{v}_i)] - \mathbf{e}^T\boldsymbol{\kappa}_{\boldsymbol{\theta}}^0(\mathbf{J}-\mathbf{J}_L)^T\boldsymbol{\beta} - \mathbf{e}^T\boldsymbol{\kappa}_{\boldsymbol{\theta}}^1\mathbf{J}_L^T\boldsymbol{\beta} \tag{A13}$$

$$= L_{\mathcal{D}}^{\mathrm{SGVB}}(\boldsymbol{\theta},\boldsymbol{\beta}) - \sum_{j=1}^{D}\mathrm{KL}[q_{\boldsymbol{\beta}}(\mathbf{v}_i)||p(\mathbf{v}_i)] - \mathbf{e}^T\boldsymbol{\kappa}_{\boldsymbol{\theta}}^1\mathbf{J}_L^T\boldsymbol{\beta}, \tag{A14}$$

since $\mathrm{KL}[q(w_{ij}|z_j=0)||p(w_{ij}|z_j=0)]=0$. We assume $\mathrm{KL}[q(w_{ij}|z_j=1)||p(w_{ij}|z_j=1)]=\chi$ as in [37], which means that transforming $p(w_{ij}|z_j=1)$ to $q(w_{ij}|z_j=1)$ requires χ nats. Thus, $\mathbf{K}_{\boldsymbol{\theta}}^1=[\chi]_{d\times D}$. The last term is then simplified to

$$-\chi d\sum_{j=1}^{D} j\beta_j \tag{A15}$$

Then,

$$\mathcal{L}^{\text{SGVB}}_{\boldsymbol{\theta},\boldsymbol{\beta}} = L^{\text{SGVB}}_{\mathcal{D}}(\boldsymbol{\theta},\boldsymbol{\beta}) - \sum_{j=1}^{D} \text{KL}[q_{\boldsymbol{\beta}}(\mathbf{v}_i)||p(\mathbf{v}_i)] - \chi d \sum_{j=1}^{D} j\beta_j, \tag{A16}$$

$$\leq L^{\text{SGVB}}_{\mathcal{D}}(\boldsymbol{\theta},\boldsymbol{\beta}) - \chi d \sum_{j=1}^{D} j\beta_j \tag{A17}$$

where the inequality is because KL is non-negative. Let $\lambda = \chi d$. Then, maximizing the evidence lower bound presents the same objective in (16). This objective assigns greater penalization to the larger sub-networks with more redundant nodes. To compare with (23) [37] that uses a constant coefficient over the probabilities, our reduced formulation provides an *ordered ℓ-0 regularization* instead of a *uniform ℓ-0 regularization*.

Note that (A16) ignores the weight uncertainty compared with (7). (A17) further ignores the uncertainty over the ordered mask, reduced to a deterministic formulation for a *nested neural network with learned weight importance*.

A.5 Discussion of Regularization

The vanilla variational auto-encoder suffers from the problem of posterior collapse. During optimization, the KL term could reduce to 0, thus the approximate posterior equals the prior, which indicates no information is learned from the data. However, a useful approximate inference requires the KL to be positive. Previously, δ-VAE [44] proposes autoregressive latent variables to enforce the KL term to be positive. Our method has a simple but effective structure over the latent space, and is compatible to the previous advances.

We assume a hard sample obtained from the Downhill distribution, and denote κ_i as the KL divergence calculated for variable h_i. According to the parameterization in Section 4.1.1, the KL term is

$$\begin{aligned}
&\text{KL}[q(\mathbf{h},\mathbf{z}|\mathbf{x})||p(\mathbf{h},\mathbf{z})] \\
&= \mathbb{E}_{q(\mathbf{z})}[\log \frac{q(\mathbf{z})}{p(\mathbf{z})}] + \mathbb{E}_{q(\mathbf{z})} \underbrace{\mathbb{E}_{q(\mathbf{h}|\mathbf{z})}[\log \frac{q(\mathbf{h}|\mathbf{z})}{p(\mathbf{h}|\mathbf{z})}]}_{\boldsymbol{\kappa}=[\kappa_i]_i} \\
&= \underbrace{\sum_i \beta_i \log \frac{\beta_i}{(1-\pi_{i+1})\prod_{k=1}^{i}\pi_k}}_{\Phi_1} + \underbrace{\sum_i \beta_i \kappa_i}_{\Phi_2}
\end{aligned} \tag{A18}$$

Given a fixed $[\kappa_i]_i$, we solve for $[\beta_i]_i$ to understand the structure that VND brings to the regularization term. We could massage Φ_2 to such that

$$\Phi_2 = \sum_i \beta_i \kappa_i = -\sum_i \beta_i \log e^{-\kappa_i} \tag{A19}$$

The KL divergence term could be combined as

$$\begin{aligned}
\text{KL} &= \Phi_1 + \Phi_2 \\
&= \sum_i \beta_i \log \frac{\beta_i}{e^{-\kappa_i}(1-\pi_{i+1})\prod_{k=1}^{i}\pi_k}
\end{aligned} \tag{A20}$$

To make it a proper KL for the Gaussian-Bernoulli mixture, we normalize the second distribution with the normalizing constant $C = \sum_i [e^{-\kappa_i}(1-\pi_{i+1})\prod_{k=1}^{i}\pi_k]$.

$$\begin{aligned}
\text{KL} &= \sum_i \beta_i \log \frac{\beta_i}{e^{-\kappa_i}(1-\pi_{i+1})\prod_{k=1}^{i}\pi_k} \frac{C}{C} \\
&= \sum_i \beta_i \log \frac{\beta_i}{e^{-\kappa_i}(1-\pi_{i+1})\prod_{k=1}^{i}\pi_k/C} - \sum_i \beta_i \log C \\
&= \sum_i \beta_i \log \frac{\beta_i}{e^{-\kappa_i}(1-\pi_{i+1})\prod_{k=1}^{i}\pi_k/C} - \log C
\end{aligned} \tag{A21}$$

The first term is now a proper KL divergence. The minimum value of 0 occurs when $\beta_i = e^{-\kappa_i}(1-\pi_{i+1})\prod_{k=1}^{i}\pi_k/C$, then $\text{KL} = -\log C$. In this case, if $\kappa_i = 0, \forall i$, then $C = 1$ and $\text{KL} = 0$. If $\kappa_i > 0$, then $C < 1$ and $\text{KL} > 0$. If κ_i is lower bounded with any previous advance, the KL is lower bounded, such that the mode collapse could be avoided.

Diversity: As discussed in Section 4.1.2, as long as the posterior does not collapse to the *single-modal* case, the diversity could be guaranteed. The *single-modal* case corresponds to $\beta_1 = 1$ and $\beta_i = 0, \forall i > 1$. The form of KL in (A21) prevents such case. Consider the situation when the optimal β_i is reached:

- If $\kappa_i = 0, \forall i$, then $C = 1$ and $\text{KL} = 0$. The posterior collapses to the prior (this could be avoided as discussed above). $\beta_i = (1-\pi_{i+1})\prod_{k=1}^{i}\pi_k$, still keeps a geometric distribution form shown in the left graph in Fig. 1 (Rippel *et al*). The multi-modal structure is kept.
- If $\kappa_i > 0$, $\beta_i = e^{-\kappa_i}(1-\pi_{i+1})\prod_{k=1}^{i}\pi_k/C > 0$. The single model case is avoided.

B IMPLEMENTATION

B.1 Extension to Convolutional Layer

We consider a convolutional layer takes in a single tensor $\mathbf{H}_m^{H\times W\times C}$ as input, where m is the index of the batch, H, W and C are the dimensions of feature map. The layer has D filters aggregated as $\mathbf{w}^{D\times H'\times W'\times C}$ and outputs a matrix $\mathbf{F}_{mj}^{\bar{H}\times\bar{W}}$. In the paper, we consider the ordered masks applied over the output channels and each filter corresponds to a dimension in $\mathbf{z}$. As shown in [28], [40], the local reparameterization trick can be applied, due to the linearity of the convolutional layer.

$$f_{mj} = b_{mj} z_j^*, \quad \text{vec}(b_{mj}) \sim \mathcal{N}(\gamma_{mj}, \delta_{mj}) \tag{A22}$$
$$\gamma_{mj} = \text{vec}(\mathbf{H}_m * \mathbf{w}), \quad \delta_{mj} = \text{diag}(\text{vec}(\mathbf{H}_m^2 * \sigma_j^2))$$

where z_j^* is the j-th dimension of the sampled ordered mask $\mathbf{z}^* = \mathbf{v}^* \sim q_{\boldsymbol{\beta}}(\mathbf{z})$.

To calculate the KL term (11), the only modification is to let the first summation be over the height, width and input channels in (13).

$$\Phi_2 = \sum_{\mathbf{z}\in\mathcal{V}} q_{\boldsymbol{\beta}}(\mathbf{z}) \sum_{i}^{H'\times W'\times C} \sum_{j}^{D} \int_{w_{ij}} q_{\boldsymbol{\theta}}(w_{ij}|z_j^k) \log \frac{q_{\boldsymbol{\theta}}(w_{ij}|z_j^k)}{p(w_{ij}|z_j^k)} \tag{A23}$$

B.2 Re-scale weights for testing

During training, the network drops nodes with the variational nested dropout. In testing, the network fixes width of each layer and no dropout operation is adopted. To make the expectation consistent over training and testing [53], we re-scale the weights according to the probability to keep a node.

$$\mathbb{E}_{\mathbf{z}\sim q_\beta(\mathbf{z}),\mathbf{x}\sim\mathcal{D}_{\text{tr}}}[\mathbf{F}|\mathbf{x},\mathbf{z}] \approx \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_{\text{te}}}[\mathbf{F}|\mathbf{x},\mathbf{z}=\bar{\mathbf{v}}], \tag{A24}$$

where $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{te}}$ are the splits of training set and testing set, and $\bar{\mathbf{v}}$ is the user-specified width according to the real demand during testing time.

We take the fully-connected layer as an example. For simplicity, we treat w_{ij} as deterministic here.

$$\mathbb{E}_{\mathbf{z}\sim q_\beta(\mathbf{z})}[f_{mj}] = \mathbb{E}_{\mathbf{z}\sim q_\beta(\mathbf{z})}\big[z_j \sum_{i=1}^{d} h_{mi}\theta_{ij}\big] = \mathbb{E}_{\mathbf{z}\sim q_\beta(\mathbf{z})}[z_j] \sum_{i=1}^{d} h_{mi}\theta_{ij} \tag{A25}$$

Note that, different from the probability $q_\beta(\mathbf{v}_j) = \beta_j$, $\mathbb{E}_{\mathbf{z}\sim q_\beta(\mathbf{z})}[z_j]$ is the probability that the j-th node is kept.

With a well-trained layer in a Bayesian nested neural network, we have the learned importance $\boldsymbol{\beta} = [\beta_j]_j$. Assume that the $\boldsymbol{\beta}$ is also generated by a chain of hidden Bernoulli variables following (1) with the parameters $\boldsymbol{\mu} = [\mu_j]_j$, with $\mu_1 := 1$ and $\mu_j = q(z_j = 1|z_{j-1} = 1)$. We are interested in the marginal distribution $p(z_j = 1) = \prod_{k=1}^{j} \mu_k$ but we only have β_j's.

$$\begin{aligned} \beta_1 &= (1-\mu_2)\mu_1 = 1-\mu_2 \\ \beta_2 &= (1-\mu_3)\mu_2\mu_1 \\ &\dots \end{aligned} \tag{A26}$$

Solving each equation sequentially, we obtain

$$\begin{aligned} &p(z_1 = 1) = 1, \\ &p(z_2 = 1) = 1-\beta_1, \\ &p(z_3 = 1) = 1-\beta_1-\beta_2, \\ &\dots \end{aligned} \tag{A27}$$

and (A25) becomes

$$\mathbb{E}_{\mathbf{z}\sim q_\beta(\mathbf{z})}[f_{mj}] = (1 - \sum_{k=1}^{j-1} \beta_k) \sum_{i=1}^{d} h_{mi}\theta_{ij}, \tag{A28}$$

where we can define $\beta_0 = 0$. Then, the *scaling factor* is $1 - \sum_{k=1}^{j-1} \beta_k$ for each w_{ij}.

Another way is to optimize the conditional probabilities $[\mu_j]_j$ instead of β_j, with β_j in previous derivation replaced by $(1-\mu_{j+1})\mu_j$[2]. The scaling factor is then $\prod_{k=1}^{j} \mu_k$ for each w_{ij}. Also, for simplicity, one can optimize $\bar{\mu}_k$ where $\mu_k = \text{sigmoid}(\bar{\mu}_k)$.

C EXPERIMENTS

C.1 Experimental setups

We implement FN^3, individual Bayesian neural networks (IBNN) and the proposed Bayesian Nested Neural Network (BN^3) with PyTorch framework. We use the cross-entropy loss for negative expected log-likelihood. For balancing the regularization and likelihood, we add a scaling factor κ for the KL term, which is a common trick in Bayesian learning [22].

2. We use this parameterization in our implementation, while we use β_j in most of our derivation for simplicity in writing.

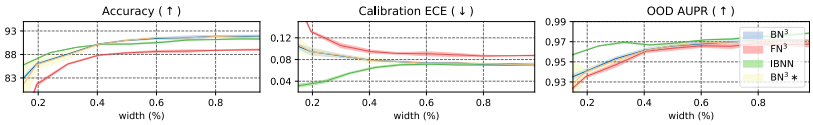

Fig. 8: Performance of VGG11 on Cifar10 with less data for collecting BN statistics.

C.1.1 Cifar10/Cifar100.

For data augmentation, we use random cropping with padding beforehand, and random flipping the image horizontally.

VGG11: We train BN^3-VGG11 with natural gradient descent[3] (NGD) [42], as it was shown to make the Bayesian neural network converge faster [25]. The network is trained for 600 epochs with an initial learning rate 0.1 and momentum 0.9. The learning rate is scaled by a factor 0.1 every 150 epochs. κ is set to 10^{-5}. For training the network, we use VGG11 with $1.5\times$ number of channels and truncate the $2/3$ part with higher importance for testing. We add one dense layer after the stack of convolutional layers. The first feature extraction layer and the last two dense layers for classification are variational Bayes layer without nested dropout, with our parameterization proposed in Section 3.4. For the convolutional layer, we divide the convolutional filters into 32 groups for group sparsity. 30 groups are applied nested dropout while the remaining 2 groups are for extracting the basic features. The $\log \alpha_{ij}$ is initialized to -8 for the first layer and -1 for the rest layers. The $[\bar{\mu}_j]_j$ are all initialized to 3. We train IBNN-VGG11 with NGD for 240 epochs, with an initial learning rate 0.1 and scaled by 0.3 every 40 epochs. Every individual BNN is fixed at some width between the fraction 0 and 1. We train FN^3-VGG11 with SGD and momentum 0.9, as SGD performs better in training FN^3-VGG11. Other setups are similar to BN^3-VGG11.

MobileNetV2: We train BN^3-MobileNetV2 with a similar setup as BN^3-VGG11, except the following. For inverted residual block, we apply nested dropout to the middle depth-wise convolutional layer, because it already sparsifies the convolution filters in the previous point-wise convolution layer, and channels in the following point-wise convolution layer [19]. Introducing more nested dropout units would cause extra and irregular sparsification which deteriorates the performance. We use a normal-size MobileNetV2 and divide the weights into 16 groups. One group is fixed for base feature extraction. The experimental setups for IBNN-MobileNetV2 and FN^3-MobileNetV2 follow that on VGG11.

ResNeXt-Cifar: The setups for ResNeXt-Cifar are similar to that of MobileNetV2, while the number of groups is 32.

C.1.2 Tiny-ImageNet.

For data augmentation, we use random cropping with padding beforehand, random rotation of 20 degree and random flipping the image horizontally. All images are finally cropped to 64×64 and all networks are trained from scratch.

VGG11: To increase the capacity, we take VGG11 with $1.5\times$ number of channels as the base network. The network is trained with NGD for 300 epochs with an initial learning rate 0.1. The learning rate is scaled by 0.3 every 25 epochs. κ is set to 10^{-6}. The weights are divided into 32 groups and 8 groups are fixed for base feature extraction.

MobileNetV2: We train a MobileNetV2 with $1.5\times$ number of channels, and take the $2/3$ part with higher importance as the base network for testing. The network is trained with NGD for 300 epochs with an initial learning rate 0.1. The learning rate is scaled by 0.3 every 40 epochs. The weights are divided into 16 groups and 1 groups are fixed for base feature extraction.

ResNeXt-Cifar: We train a ResNeXt-Cifar with normal size. The weights are divided into 32 groups and 8 groups are fixed for base feature extraction. The network is trained with SGD for 300 epochs with an initial learning rate 0.1. The learning rate is scaled by 0.3 every 30 epochs.

The remaining setups are similar to that on Cifar10/Cifar100.

C.1.3 Lung Abnormalities Segmentation.

The network uses a UNet shape architecture with layers 32-64-128-192 for the encoder (two layers fewer than the standard UNet). The optimizer is Adam with initial learning rate 10^{-4} decayed by 0.1 every 60 epochs. The channels are divided into 32 groups and 6 groups are fixed for base feature extraction.

C.2 BN statistics

We show that collecting batch normalization statistics on a small training set yields similar performance to using the whole dataset In this example, we use VGG11 on Cifar10. The collection proceeds by forwarding the network by 2 iterations, with a batch size 512. Thus, in total, $1024/50000$ training data are used for statistics collection. The results are shown in Figure 8 as BN^3*. We can observe that this results are similar to using all training data for statistics collection, with slightly larger variance using a lower width.

C.3 OOD detection

For out-of-domain detection, we use the SVHN dataset as the OOD data [4]. The OOD detection performance with AUROC metric is shown in Figure 9. The performance is similar to that of AUPR in Figure 5.

3. The PyTorch implementation is from https://github.com/YiwenShaoStephen/NGD-SGD.
4. http://ufldl.stanford.edu/housenumbers/

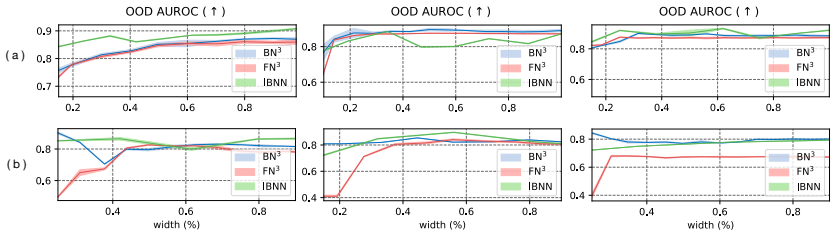

Fig. 9: The AUROC of OOD on (a) Cifar10 (b) Tiny ImageNet datasets with VGG11, MobileNetV2 and ResNeXt-Cifar (left to right).

C.4 Cifar100 results

The results on Cifar100 is shown in Figure 10. As the hyper-parameters are mostly from training on Cifar10, the results may not be optimal. We do not show the comparisons for MobileNetV2 here, as it is observed that IBNN-MobileNetV2 fails provide a decent performance on Cifar100, similar to Figure 5(b). The proposed BN^3 performs well steadily on every task.

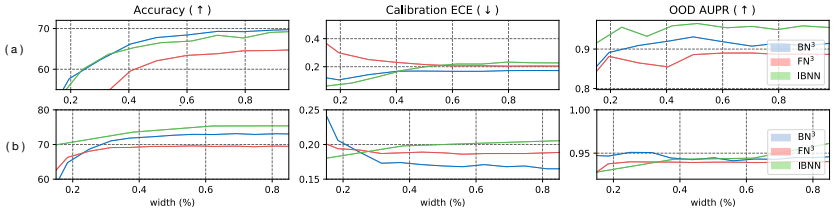

Fig. 10: Results on Cifar100 for (a) VGG11, (b) ResNeXt-Cifar.