Variational Nested Dropout

Yufei Cui[1,2], Yu Mao[1], Ziquan Liu[1], Qiao Li[3], Antoni B. Chan[1], Xue Liu[2], 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 Computer Science, McGill University
[3]School of Informatics, Xiamen University
[4]Department of Computer Science and Information Engineering, National Taiwan University

Abstract—Nested dropout is a variant of dropout operation that is able to order network parameters or features based on the pre-defined importance during training. It has been explored for: **I.** *Constructing nested nets* [11], [10]: the nested nets are neural networks whose architectures can be adjusted instantly during testing time, e.g., based on computational constraints. The nested dropout implicitly ranks the network parameters, generating a set of sub-networks such that any smaller sub-network forms the basis of a larger one. **II.** *Learning ordered representation* [48]: the nested dropout applied to the latent representation of a generative model (e.g., auto-encoder) ranks the features, enforcing explicit order of the dense representation over dimensions.

However, the dropout rate is fixed as a hyper-parameter during the whole training process. For nested nets, when network parameters are removed, the performance decays in a human-specified trajectory rather than in a trajectory learned from data. For generative models, the importance of features is specified as a constant vector, restraining the flexibility of representation learning. To address the problem, we focus on the probabilistic counterpart of the nested dropout. We propose a variational nested dropout (VND) operation that draws samples of multi-dimensional ordered masks at a low cost, providing useful gradients to the parameters of nested dropout. Based on this approach, we design a Bayesian nested neural network that learns the order knowledge of the parameter distributions. We further exploit the VND under different generative models for learning ordered latent distributions. In experiments, we show that the proposed approach outperforms the nested network in terms of accuracy, calibration, and out-of-domain detection in classification tasks. It also outperforms the related generative models on data generation tasks.

1 INTRODUCTION

Modern deep neural networks (DNNs) have achieved great success in fields of supervised learning and representation learning. In the meantime, deep learning models have a high demand for learning ordered information from data, for both *model architecture* and *representations*.

Model architecture: Deep learning models are experiencing rapid growth in model size and computation cost, which makes it difficult to deploy on diverse hardware platforms. Recent works study how to develop a network with flexible size during test time [28], [64], [63], [7], [11], [61], to reduce the cost in designing [57], training [29], compressing [20] and deploying [47] a DNN on various platforms. As these networks are often composed of a nested set of smaller sub-networks, we refer to them as *nested nets* in this paper. In this set, any smaller sub-network forms the basis of a larger one, and the residual information is learned via the incremental parameters added to the smaller sub-network [11]. One basis for creating nested nets is to order the network components (e.g., convolution channels) such that less important components can be removed first when creating the sub-network. To avoid significant performance drop when removing the components, training with explicitly *ordering of the network components* is required.

Representations: Representation (feature) learning is a major branch for modern machine learning research. The unsupervised feature learning is able to discover low-dimensional structure underlying the high-dimensional input data. Typical examples with neural network include restricted Boltzmann machine [50] and auto-encoder [59]. However, the standard approach only provides features that are entangled and with equivalent importance. To clearly identify the more important features and reduce the redundancy, recent works study imposing structural constraints to *order the learned representations* [48], [3].

An operator for neural networks, *nested dropout* (also called *ordered dropout*), was proposed for both ordering in model architectures [11], [14] and ordering representations [48], [3]. It was first developed to order the latent feature representation for the encoder-decoder models [48], [3]. Specifically, a discrete distribution is assigned over the indices of the representations, and the operation of nested dropout samples an index then drops the representations with larger indices. This imposes explicit ordered importance over *representations* by different frequencies that different dimensions of latent representation are activated during training.

Recent studies show that the nested dropout is also able to order the *network components* during training such that *nested nets* can be obtained [11], [14]. The ordering layout is applicable to different granularity levels of network components: single weights, groups of weights, convolutional channels, residual blocks, network layers, and even quantization bits. We refer to the partitions of the network components as *nodes* in this paper.

Despite its successes, nested dropout requires that the probability that an index is sampled is specified by hand as a hyper-parameter, and does not change during training. Thus, the importance of nodes or representations are pre-determined by hand rather than learned from data. To allow the dropout rate to be learned, we propose a fully Bayesian treatment for the nested dropout operator. We first propose a new variant of the nested

dropout, based on a chain of interdependent Bernoulli variables. The chain simulates the Bernoulli trials and can be understood as a special case of a two-state Markov chain, which intuitively generates order information. To save the time cost for sampling during training, we propose a new *Downhill* distribution that approximates this chain. This approximate posterior is built on Gumbel Softmax [25], [40], which efficiently generates more flexible samples compared with the Bernoulli chain. This allows efficient sampling of the multivariate *ordered mask*, and provides useful gradients to update the importance of the nodes or representations. We name this major scheme as variational nested dropout (VND), and apply it to both ordering network components and ordering latent representations.

First, based on the proposed VND, a Bayesian Nested Neural Network (BN^3) is constructed with *learnable ordered importance of network components*, where the independent distributions of nodes are interconnected with the ordering units. A mixture model prior is placed over each node, while the model selection is determined by the Downhill samples (Figure 3). A variational inference problem is formulated and resolved, and we propose several methods to simplify the sampling and calculation of the regularization term. The formulation is proved to be an generalization of ordered L0-regularization over the sub-networks. The trained BN^3 is (sub-)optimal in the trade-off of model size and prediction performance. The full Bayesian treatment further enhances the uncertainty calibration and out-of-domain detection ability of the nested nets. To further exploit the performance gain from the VND, we proposed a two-step knowledge distillation framework to obtain high-performance deterministic student nested nets.

Second, we study using VND with the variational auto-encoder [31] to *order the distributions over the latent representation*. The diversity of generated data is enhanced due to the mixture nature of the latent distribution organized by VND. Our design of approximate posterior guarantee the diversity for data generation. We further applying VND to encode aleatoric uncertainty in the probabilistic UNet [32], which is used for applications whose labels are provided by multiple annotators. The diversity of generated samples are enhanced by VND, and thus it captures the disagreement of noisy labels better.

Experiments on BN^3 with VND show that it outperforms the deterministic nested models with nested dropout in any sub-network, in terms of classification accuracy, calibration and out-of-domain detection. The student nested nets obtained by knowledge distillation further improves the performance. Results on variational auto-encoder with VND show it outperforms the baseline and related methods by a large margin in terms of image reconstruction and generation diversity. BN^3 and probabilistic UNet with VND also outperform the vanilla probabilistic UNet [32] on uncertainty-critical tasks with noisy labels.

In summary, the contributions of this paper are:
- We propose a variational nested dropout (VND) unit with a novel pair of prior and posterior distributions.
- We propose a novel Bayesian nested neural network (BN^3) that can generate large sets of uncertainty-calibrated sub-networks. The formulation can be viewed as a generalization of *ordered ℓ_0-regularization* over the sub-networks. We propose a distillation methods with VND for higher performance gain.
- We propose a new generative model, VND enhanced auto-encoder (VND-AE), whose training objective is shown to encourage diversity. We propose a new formulation using

VND to encode the aleotoric uncertainty for capturing the disagreement in annotations.

A preliminary version of this paper appears in [10]. The major differences between this paper and [10] are:
- This paper generalizes the VND to the case of generative models, formulating a new family of generative models with an explicit ordered latent structure. The concrete implementations with VND include a variational autoencoder (VND-AE) and a probabilistic UNet (VND-PUNet). Experimental results show a highly competitive performance of our generative models in generating images, compared with the recent advances.
- This paper extends the BN^3 with a knowledge distillation (KD) process for higher performance gain from VND. The KD process maintains a nested structure for the generated student nets and is executed in a compatible way with the sampling of VND.

The code[2] and supplemental[3] are publicly available.

The remainder of this paper is organized as follows. Section 2 introduces the general formulation of variational nested dropout (VND), including the chain of Bernoulli variables as prior and the proposed Downhill distribution as the approximate posterior. Section 3 provides the formulation of BN^3 using VND with a full Bayesian treatment. Section 4 presents the probabilistic generative models with the ordered latent structure. Section 5 reviews the background and recent advances of this paper. Section 6 shows the detailed evaluations of the proposed techniques. Section 7 summarize the paper and provides possible future directions.

2 VARIATIONAL NESTED DROPOUT

We first review nested dropout, and then propose our Bayesian ordering unit and variational approximation.

2.1 A Review of Nested Dropout

The previous works [48] that order the representations use either Geometric or Categorical distributions to sample the last index of the kept units, then drop the neurons with indices greater than it. Specifically, the distribution $p_{\mathbb{I}}(\cdot)$ is assigned over the representation indices $1, \ldots, K$. The nested/ordered dropout operation proceeds as follows:
1) *Tail sampling:* A tail index is sampled $I \sim p_{\mathbb{I}}(\cdot)$ that represents the last element be kept.
2) *Ordered dropping:* The elements with indices $I + 1, \ldots, K$ are dropped.

We also refer to this operation as an *ordering unit* as the representations are sorted in order.

In [48], which focuses on learning ordered representations, this operation is proved to exactly recover PCA with a one-layer neural network. Cui *et al* [11] shows this operation, when applied to groups of neural network weights or quantization bits, generates nested sub-networks that are optimal for different computation resources. They further prove that increasing from a smaller sub-network to a larger one maximizes the incremental information gain. A large network only needs to be trained once with nested dropout, yielding a set of networks with varying sizes for deployment. However, the above methods treat the nested dropout rate as a hyper-parameter, and hand-tuning the dropout rate is tedious
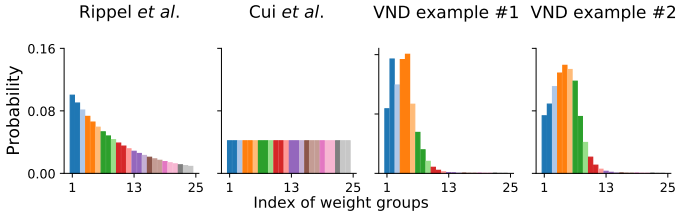
Fig. 1: The probability of tail index being sampled in different nested dropout realizations. Rippel *et al* [48] and Cui *et al* [11] adopt Geometric and Categorical distributions, which are static over different layers and the learning process. The proposed variational nested dropout (VND) learns the importances of nodes or representations from data. The two examples are from two different layers in a Bayesian nested neural network.

and may lead to suboptimal performance of the sub-networks, as compared to learning this hyperparameter from the data. As illustrated in Figure 1, the previous works use hand-specified parameters for nested dropout, which freezes the importance of the network components or representations over different layers during training.

A common practice for *regular Bernoulli dropout* is to treat the dropout rate as a variational parameter in Bayesian neural networks [15]. To find the optimal dropout rate, grid-search is first adopted [16], whose complexity grows exponentially with the number of dropout units. To alleviate the cost of searching, a continuous relaxation of the discrete dropout is proposed by which the dropout rate can be optimized directly [17], improving accuracy and uncertainty, while keeping a low training time. However, for *nested dropout*, two aspects are unclear: 1) how to take a full Bayesian treatment with nested dropout units; 2) how the relaxation can be done for these units or how the gradients can be back-propagated to the parameters of $p_{\mathbb{I}}(\cdot)$.

2.2 Bayesian Ordering Unit

The conventional nested dropout uses a Geometric distribution to sample the tail index I, $p_{\mathbb{I}}(I = i) = (1 - \pi)^i \pi$, for $i \in \{1, \ldots, K\}$. By definition, the Geometric distribution models the probability that the i-th trial is the first "success" in a sequence of independent Bernoulli trials. In the context of slimming neural networks, a "failure" of a Bernoulli trial indicates that node is kept, while a "success" indicates the tail index, where this node is kept and all subsequent nodes are dropped. Thus, π is the conditional probability of a node being a tail index, given the previous node is kept.

Sampling from the Geometric only generates the tail index of the nodes to be kept. A hard selection operation of ordered dropping is required to drop the following nodes. The ordered dropping can be implemented using a set of ordered mask vectors $\mathcal{V} = \{\mathbf{v}_1, \cdots, \mathbf{v}_K\}$, where $\mathbf{v}_j$ consists of j ones followed by $K - j$ zeros, $\mathbf{v}_j = [\underbrace{1, \ldots, 1}_{j}, \underbrace{0, \ldots, 0}_{K-j}]$. Given the sampled tail index $I \sim p_{\mathbb{I}}(\cdot)$, the appropriate mask $\mathbf{v}_I$ is selected and applied to the nodes (e.g., multiplying the weights). However, as the masking is a non-differentiable transformation and does not provide a well-defined probability distribution, the nested dropout parameters cannot be learned using this formulation.

To find a more natural prior for the nodes, we propose to use a chain of Bernoulli variables to *directly model the distribution of the ordered masks*. Let the set of binary variables

$\mathbf{z} = [z_1, \ldots, z_K]$ represent the random ordered mask. Specifically, we model the conditional distributions with Bernoulli variables,

$$p(z_1 = 1) = \pi_1, \qquad p(z_1 = 0) = 1 - \pi_1, \qquad (1)$$
$$p(z_i = 1|z_{i-1} = 1) = \pi_i, \quad p(z_i = 0|z_{i-1} = 1) = 1 - \pi_i,$$
$$p(z_i = 1|z_{i-1} = 0) = 0, \qquad p(z_i = 0|z_{i-1} = 0) = 1,$$

where π_i is the conditional probability of keeping the node given the previous node is kept, and $\pi_1 = 1$ (the first node is always kept). Note that we also allow different probabilities π_i for each z_i. The marginal distribution of z_i is

$$p(z_i = 1) = \prod_{k=1}^{i} \pi_k, \quad p(z_i = 0) = 1 - \prod_{k=1}^{i} \pi_k. \qquad (2)$$

We define the Bernoulli chain as $\mathrm{BernChain}(\mathbf{z}, \boldsymbol{\pi})$ with random variable $\mathbf{z}$ and parameter $\boldsymbol{\pi}$.

A property of this chain is that if 0 occurs at the i-th position, the remaining elements with indices $i + 1, \ldots, K$ become 0. That is, sampling from this chain generates an ordered mask, which can be directly multiplied on the nodes to realize *ordered dropping*. Another benefit is that applying a continuous relaxation [17] of the Bernoulli variables in the chain allows its parameters $\boldsymbol{\pi}$ to be optimized.

However, the sampling of $\mathbf{z}$ requires stepping through each element z_i, which has complexity $\mathcal{O}(K)$, and is thus not scalable in modern DNNs where K is large. Thus we apply the variational inference framework, while *treating $p(\mathbf{z})$ as the prior of the ordered mask* in our Bayesian treatment. One challenge is to find a tractable variational distribution $q(\mathbf{z})$ that approximates the true posterior and is easy to compute. Another challenge is to define a $q(\mathbf{z})$ that allows efficient re-parameterization, so that the gradient of the parameter of $q(\mathbf{z})$ can be estimated with low variance.

2.3 Downhill Distribution as Approximate Posterior

We next propose a novel *Downhill* distribution based on Gumbel Softmax distribution [25], [40] that generates the ordered mask $\mathbf{z}$. Differently, the binary ordered mask with element z_i is extended to the real values between range $[0, 1]$. Specifically, $z_i \in [0, 1]$, $1 \geq z_1 \geq z_2 \geq \cdots \geq z_K \geq 0$.

Definition 1. Downhill Random Variables (r.v.). Let the temperature parameter $\tau \in (0, \infty)$. An r.v. $\mathbf{z}$ has a Downhill distribution $\mathbf{z} \sim \mathrm{Downhill}(\boldsymbol{\beta}, \tau)$, if its density is:

$$q(z_1, \ldots, z_K) \qquad (3)$$
$$= \Gamma(K) \tau^{K-1} \left[\sum_{i=1}^{K} \frac{\beta_i}{(z_{i-1} - z_i)^\tau} \right]^{-K} \prod_{i=1}^{K} \frac{\beta_i}{(z_{i-1} - z_i)^{\tau+1}},$$

where $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_K]$ are the probabilities for each dimension.

Two important properties of Downhill distributions are:

- **Property 1.** If $\mathbf{c} \sim \mathrm{Gumbel\_softmax}(\tau, \beta, \epsilon_z)$[4], then $z_i = 1 - \mathrm{cumsum}_i'(\mathbf{c})$, where $\mathbf{e}$ is a K-dimensional vector of ones, and $\mathrm{cumsum}_i'(\mathbf{c}) = \sum_{j=0}^{i-1} c_j$. $c_0 := 1$. ϵ_z is a standard uniform variable.

---

4. For Gumbel-softmax sampling, we first draw $g_1 \ldots g_K$ from Gumbel$(0, 1)$, then calculate $c_i = \mathrm{softmax}(\frac{\log(\beta_i) + g_i}{\tau})$. The samples of Gumbel$(0, 1)$ can be obtained by first drawing $\epsilon_z \sim \mathrm{Uniform}(0, 1)$ then computing $g = -\log(-\log(\epsilon_z))$.
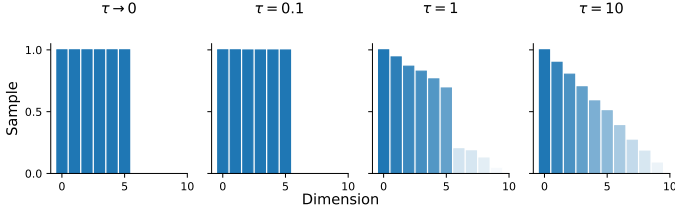
Fig. 2: The multivariate Downhill samples under different temperatures τ. When $\tau \to 0$, a clear cliff is observed as the dimension increases, which is beneficial for differentiating important or unimportant nodes. As τ increases, the shape becomes a slope where the gaps between important/unimportant nodes are smoother, which is beneficial for training.

- **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.

Property 1 shows the sampling process of the Downhill distribution. We visualize the Downhill samples in Figure 2. As each multivariate sample has a shape of a long descent from left to right, we name it *Downhill* distribution. The temperature variable τ controls the sharpness of the downhill or the smoothness of the step at the tail index. When τ is large, the slope is gentle in which case no nodes are dropped, but the less important nodes are multiplied with a factor less than 1. When $\tau \to 0$, the shape of the sample becomes a cliff which is similar to the prior $p(\mathbf{z})$ on ordered masks, where the less important nodes are dropped (i.e., multiplied by 0). Property 1 further implies the gradient $\frac{\partial}{\partial \boldsymbol{\beta}} \mathbb{E}_{\mathbf{z} \sim q_{\boldsymbol{\beta}}(\mathbf{z})}[\zeta(\mathbf{z})]$ can be estimated with low variance, for a cost function $\zeta(\mathbf{z})$. Because the samples of $\mathbf{z}$ are replaced by a differentiable function $t(\boldsymbol{\beta}, \epsilon_z)$, then $\frac{\partial}{\partial \boldsymbol{\beta}} \mathbb{E}_{\mathbf{z} \sim q_{\boldsymbol{\beta}}(\mathbf{z})}[\zeta(\mathbf{z})] = \mathbb{E}_{\epsilon_z \sim \text{Uniform}(0,1)}[\frac{\partial \zeta}{\partial t} \frac{\partial t}{\partial \boldsymbol{\beta}}]$, where $t(\cdot, \cdot)$ represents the whole transformation process in Prop. 1.

Recall that our objective is to approximate the chain of Bernoulli variables $p(\mathbf{z})$ with $q_{\boldsymbol{\beta}}(\mathbf{z})$. Property 2 shows why the proposed distribution is consistent with the chain of Bernoullis in essence, and provides an easy way to derive the evidence lower bound (ELBO) for variational inference. The proof for the two properties is in Appendix A.1. This simple transformation of Gumbel softmax samples allows fast sampling of an ordering unit. Compared with $p(\mathbf{z})$, the only sequential process is a cumulative summation, which could be effectively accelerated by modern computation library.

3 BAYESIAN NESTED NEURAL NETWORK

In this section, we present the Bayesian nested neural network (BN^3) based on the fundamental units proposed in Section 2.

3.1 Bayesian Inference and SGVB

Consider a dataset $\mathcal{D}$ constructed from N pairs of instances $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$. Our objective is to estimate the parameters $\mathbf{u}$ of a neural network $p(\mathbf{y}|\mathbf{x}, \mathbf{u})$ that predicts $\mathbf{y}$ given input $\mathbf{x}$ and parameters $\mathbf{u}$. In Bayesian learning, a prior $p(\mathbf{u})$ is placed over the parameters $\mathbf{u}$. After data $\mathcal{D}$ is observed, the prior distribution is transformed into a posterior distribution $p(\mathbf{u}|\mathcal{D})$.

For neural networks, computing the posterior distribution using the Bayes' rule requires computing intractable integrals over $\mathbf{u}$. Thus, approximation techniques are required. One family of techniques is variational inference, with which the posterior

$p(\mathbf{u}|\mathcal{D})$ is approximated by a parametric distribution $q_{\boldsymbol{\phi}}(\mathbf{u})$, where $\boldsymbol{\phi}$ are the variational parameters. $q_{\boldsymbol{\phi}}(\mathbf{u})$ is approximated by minimizing the Kullback-Leibler (KL) divergence with the true posterior, $\text{KL}[q_{\boldsymbol{\phi}}(\mathbf{u})||p(\mathbf{u}|\mathcal{D})]$, which is equivalent to maximizing the *evidence lower bound* (ELBO):

$$\mathcal{L}_{\boldsymbol{\phi}} = L_{\mathcal{D}}(\boldsymbol{\phi}) - \text{KL}[q_{\boldsymbol{\phi}}(\mathbf{u})||p(\mathbf{u})], \quad (4)$$

where the expected data log-likelihood is

$$L_{\mathcal{D}}(\boldsymbol{\phi}) = \sum_{i=1}^N \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{u})}[\log p(\mathbf{y}_i|\mathbf{x}_i, \mathbf{u})]. \quad (5)$$

The integration $L_{\mathcal{D}}$ is not tractable for neural networks. An efficient method for gradient-based optimization of the variational bound is *stochastic gradient variational Bayes* (SGVB) [31], [30]. SGVB parameterizes the random variables (r.v.) $\mathbf{u} \sim q_{\boldsymbol{\phi}}(\mathbf{u})$ as $\mathbf{u} = t(\boldsymbol{\epsilon}, \boldsymbol{\phi})$ where $t(\cdot)$ is a differentiable function and $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$ is a noise variable with fixed parameters. With this parameterization, an unbiased differentiable minibatch-based Monte Carlo estimator of the expected data log-likelihood is obtained:

$$L_{\mathcal{D}}(\boldsymbol{\phi}) \simeq L_{\mathcal{D}}^{\text{SGVB}}(\boldsymbol{\phi}) = \frac{N}{M} \sum_{i=1}^M \log p(\mathbf{y}_i|\mathbf{x}_i, \mathbf{u} = t(\boldsymbol{\epsilon}, \boldsymbol{\phi})), \quad (6)$$

where $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^M$ is a minibatch of data with M random instances $(\mathbf{x}_i, \mathbf{y}_i) \sim \mathcal{D}$, and $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$.

3.2 Bayesian Nested Neural Network

In our model, the r.v. $\mathbf{u} = (\mathbf{W}, \mathbf{z})$ consists of two parts: weight matrix $\mathbf{W}$ and ordering units $\mathbf{z}$. The ordering units order the network weights and generate sub-models that minimize the residual loss of a larger sub-model [48], [11]. We define the corresponding variational parameters $\boldsymbol{\phi} = (\boldsymbol{\theta}, \boldsymbol{\beta})$, where $\boldsymbol{\theta}$ and $\boldsymbol{\beta}$ are the variational parameters for the weights and ordering units respectively. We then have the following optimization objective,

$$\mathcal{L}_{\boldsymbol{\theta}, \boldsymbol{\beta}}^{\text{SGVB}} \simeq L_{\mathcal{D}}^{\text{SGVB}}(\boldsymbol{\theta}, \boldsymbol{\beta}) - \text{KL}[q_{\boldsymbol{\theta}, \boldsymbol{\beta}}(\mathbf{W}, \mathbf{z})||p(\mathbf{W}, \mathbf{z})], \quad (7)$$

$$L_{\mathcal{D}}^{\text{SGVB}}(\boldsymbol{\theta}, \boldsymbol{\beta}) =$$
$$\frac{N}{M} \sum_{i=1}^M \log p(\mathbf{y}_i|\mathbf{x}_i, \mathbf{W} = t_w(\boldsymbol{\epsilon}_w, \boldsymbol{\theta}), \mathbf{z} = t_z(\boldsymbol{\epsilon}_z, \boldsymbol{\beta})), \quad (8)$$

where $\boldsymbol{\epsilon}_z$ and $\boldsymbol{\epsilon}_w$ are the random noise, and $t_w(\cdot)$ and $t_z(\cdot)$ are the differentiable functions that transform the noises to the probabilistic weights and ordered masks.

Next, we focus on an example of a fully-connected (FC) layer. Assume the FC layer in neural network takes in activations $\mathbf{H} \in \mathbb{R}^{M \times d}$ as the input, and outputs $\mathbf{F} = f(\mathbf{H}) = \mathbf{HW}$, where the weight matrix $\mathbf{W} \in \mathbb{R}^{d \times D}$, d and D are the input and output size, and M is the batch size. The elements are indexed as h_{mi}, f_{mj} and w_{ij} respectively. We omit the bias for simplicity, and our formulation can easily be extended to include the bias term. We have the ordering unit $\mathbf{z} \in \mathbb{R}^D$ with each element z_j applied on the column of $\mathbf{W}$, by which the columns of $\mathbf{W}$ are given different levels of importance. Note that $\mathbf{z}$ is flexible, and can be applied to $\mathbf{W}$ row-wise or element-wise as well.

The prior for $\mathbf{W}$ assumes each weight is independent, $p(\mathbf{W}) = \prod_{ij} p(w_{ij})$, where $i \in \{1, \ldots, d\}$ and $j \in \{1, \ldots, D\}$. We choose to place a mixture of two univariate variables as the prior over each element of the weight matrix w_{ij}. For example, if we use the univariate normal distribution, then each w_{ij} is a Gaussian mixture, where the 2 components are:

$$p(w_{ij}|z_j = 0) = \mathcal{N}(w_{ij}|\mu_{ij}^0, \sigma_{ij}^{0}{}^2)$$
$$p(w_{ij}|z_j = 1) = \mathcal{N}(w_{ij}|\mu_{ij}^1, \sigma_{ij}^{1}{}^2)$$

where $(\mu_{ij}^0, \sigma_{ij}^0)$ and $(\mu_{ij}^1, \sigma_{ij}^1)$ are the means and standard deviations for the two components. We fix $\mu_{ij}^0 = 0$ and σ_{ij}^0 to

be a small value, resulting in a spike at zero for the component when $z_j = 0$. The variable z_j follows the chain of Bernoulli distributions proposed in (1). Using (2), the marginal distribution of w_{ij} is then

$$p(w_{ij}) =$$
$$(1 - \prod_{k=1}^{i} \pi_k)\mathcal{N}(w_{ij}|\mu_{ij}^0, {\sigma_{ij}^0}^2) + (\prod_{k=1}^{i} \pi_k)\mathcal{N}(w_{ij}|\mu_{ij}^1, {\sigma_{ij}^1}^2).$$

To calculate the expected data log-likelihood, our *Downhill* distribution allows efficient sampling and differentiable transformation for the ordering units (Section 2.3). The reparameterization of weight distributions has been widely studied [30], [38], [31] to provide gradient estimate with low variance. Our framework is compatible with these techniques, which will be discussed in Section 3.4. An overview of sampling is shown in Figure 3.

3.3 Posterior Approximation

Next, we introduce the computation of the KL divergence. We assume the posterior $q_{\boldsymbol{\theta}}(\mathbf{W})$ takes the same form as the prior, while $q_{\boldsymbol{\beta}}(\mathbf{z})$ takes the Downhill distribution $\mathbf{z} \sim \text{Downhill}(\boldsymbol{\beta}, \tau)$. We consider the case that $\tau \to 0$ for simplicity, while τ can be adjusted in the training process as annealing. For this layer, the KL divergence in (7) is

$$\text{KL}[q_{\boldsymbol{\beta},\boldsymbol{\theta}}(\mathbf{W}, \mathbf{z})||p(\mathbf{W}, \mathbf{z})] \tag{9}$$
$$= \underbrace{\mathbb{E}_{q_{\boldsymbol{\beta}}(\mathbf{z})}[\log \frac{q_{\boldsymbol{\beta}}(\mathbf{z})}{p(\mathbf{z})}]}_{\Phi_1} + \underbrace{\mathbb{E}_{q_{\boldsymbol{\beta}}(\mathbf{z})}\mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{W}|\mathbf{z})}[\log \frac{q_{\boldsymbol{\theta}}(\mathbf{W}|\mathbf{z})}{p(\mathbf{W}|\mathbf{z})}]}_{\Phi_2}.$$

Term Φ_1 of (9) is

$$\Phi_1 = \sum_{\mathbf{z} \in \mathcal{V}} q_{\boldsymbol{\beta}}(\mathbf{z}) \log \frac{q_{\boldsymbol{\beta}}(\mathbf{z})}{p(\mathbf{z})} = \sum_{j=1}^{D} \text{KL}[q_{\boldsymbol{\beta}}(\mathbf{v}_j)||p(\mathbf{v}_j)],$$

where $\mathcal{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_D\}$ is the set of ordered masks. The number of components in the $\mathbf{z}$ space is reduced from D^2 to D, because there are only D possible ordered masks. By definition, the probabilities are

$$q_{\boldsymbol{\beta}}(\mathbf{v}_j) = \beta_j, \quad p(\mathbf{v}_j) = (1 - \pi_{j+1}) \prod_{k=1}^{j} \pi_k, \tag{10}$$

where we define $\pi_{D+1} = 0$. Then we can write

$$\Phi_1 = \boldsymbol{\beta}^T (\log(\boldsymbol{\beta}) - \log((\mathbf{e} - \tilde{\boldsymbol{\pi}})^T \mathbf{J}_L \boldsymbol{\pi})), \tag{11}$$

where $\log(\cdot)$ is an element-wise log function, $\boldsymbol{\pi} = [\pi_1, \ldots, \pi_D]$, $\tilde{\boldsymbol{\pi}} = [\pi_2, \ldots, \pi_{D+1}]$, $\mathbf{e}$ is a vector of 1s and $\mathbf{J}_L$ is a lower triangular matrix with each element being 1.

We define $\kappa_{ij}^k(\boldsymbol{\theta})$ as the KL of w_{ij} for component $\kappa \in \{0, 1\}$. Consider the matrices $\boldsymbol{\kappa}_{\boldsymbol{\theta}}^0 = [\kappa_{ij}^0(\boldsymbol{\theta})]_{ij} \in \mathbb{R}^{d \times D}$ and $\boldsymbol{\kappa}_{\boldsymbol{\theta}}^1 = [\kappa_{ij}^1(\boldsymbol{\theta})]_{ij} \in \mathbb{R}^{d \times D}$, which are easily computed by applying the KL function element-wise. The term Φ_2 is then expressed as

$$\Phi_2 = \mathbf{e}^T \boldsymbol{\kappa}_{\boldsymbol{\theta}}^0 (\mathbf{J} - \mathbf{J}_L)^T \boldsymbol{\beta} + \mathbf{e}^T \boldsymbol{\kappa}_{\boldsymbol{\theta}}^1 \mathbf{J}_L^T \boldsymbol{\beta}, \tag{12}$$

where $\mathbf{J}$ is a matrix of 1s.

Ordered ℓ_0-Regularization. We show that, if given the spike-and-slab priors, our KL term in (9) has an interpretation as *a generalization of an ordered ℓ_0 regularization over the sub-networks*. The corresponding reduced objective for deterministic networks is

$$\min_{\boldsymbol{\theta},\boldsymbol{\beta}} \mathbb{E}_{q(\mathbf{z}|\boldsymbol{\beta})} L_{\mathcal{D}}(\boldsymbol{\theta}, \boldsymbol{\beta}) + \lambda \sum_{j}^{D} j \beta_j \tag{13}$$

Note that larger sub-networks have greater penalization. The derivations and proofs for $p(\mathbf{v}_j)$, Φ_2 and regularization are in Appendix A.
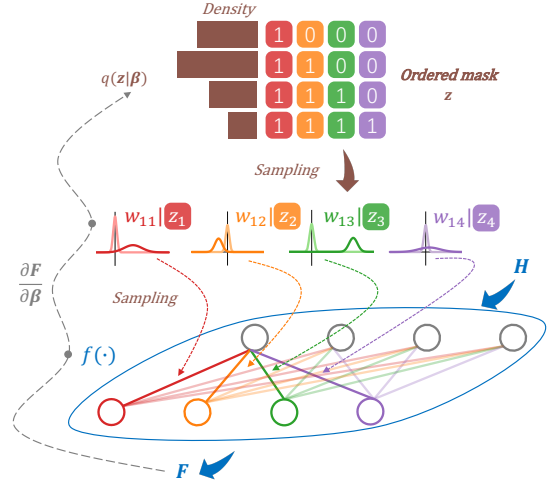

Fig. 3: Sampling process in a layer for calculating the data log-likelihood (Equation 8). A fully connected layer $f(\cdot)$ takes $\mathbf{H}$ as an input and outputs $\mathbf{F}$. The variational ordering unit $q(\mathbf{z}|\boldsymbol{\beta})$ generates ordered mask $\mathbf{z} = [z_j]_j$. Nodes w_{ij}'s with the same color share an element z_j. The gradient through stochastic nodes $\frac{\partial \mathbf{F}}{\partial \boldsymbol{\beta}}$ can be estimated efficiently, to update the importance $\boldsymbol{\beta}$.

3.4 Implementation

For efficient sampling of the weight distributions, we put multiplicative Gaussian noise $\eta_{ij} \sim \mathcal{N}(1, \alpha)$ on the weight w_{ij}, similar to [30], [42], [37]. We take w_{ij} for $z_j = 1$ as an example.

$$w_{ij} = \theta_{ij} \eta_{ij} = \theta_{ij}(1 + \sqrt{\alpha_{ij}}\epsilon_w), \ \epsilon_w \sim \mathcal{N}(0, 1), \tag{14}$$
$$w_{ij} \sim \mathcal{N}(w_{ij}|\theta_{ij}, \alpha_{ij}\theta_{ij}^2). \tag{15}$$

We also assume a log-uniform prior [30], [42], [37], i.e., $p(\log|w_{ij}| \mid z_j = 1) = \text{const}$. With this prior, the negative KL term $-\kappa_{ij}(\boldsymbol{\theta})^1$ does not depend on the variational parameter θ_{ij}^1 [30], when the parameter α_{ij} is fixed,

$$- \text{KL}[q(w_{ij}|\theta_{ij}^1, \alpha_{ij}, z_j = 1)||p(|w_{ij}| \, |z_j = 1)]$$
$$= \frac{1}{2} \log \alpha_{ij} - \mathbb{E}_{\epsilon_w \sim \mathcal{N}(1, \alpha_{ij})} \log |\epsilon_w| + C, \tag{16}$$

where C is a constant. Note that the prior can be flexibly replaced by other distributions like Gaussian, while we choose log-uniform for simplicity of optimization, as θ_{ij} is eliminated from the computation of KL.

As the second term in (16) cannot be computed analytically and should be estimated by sampling, Kingma *et al* [30] propose to sample first and design a function to approximate it. But their approximation of $\kappa_{ij}^1(\theta)$ does not encourage $\alpha_{ij} > 1$ as the optimization would be difficult. They truncate $\alpha_{ij} \leq 1$ corresponding to a small variance, which is not flexible. Molchanov *et al* [42] use a different parameterization that pushes $\alpha_{ij} \to \infty$, as illustrated in Figure 7. This means the w_{ij} can be pruned, generating a single sparse neural network. In our model, we want the order or sparsity of weights to be explicitly controlled by the ordering unit $\mathbf{z}$, otherwise the network would collapse to a single model rather than generate a nested set of sub-models. Thus, we propose another approximation to $-\kappa_{ij}(\boldsymbol{\theta})^1$ (16),

$$a_1 e^{-e^{a_4} \cdot (a_2 + a_3 * \log \alpha_{ij})^2} - 0.5 \log(1 + \alpha_{ij}^{-1}) + C, \tag{17}$$

where $a_1 = 0.7294$, $a_2 = -0.2041$, $a_3 = 0.3492$ and $a_4 = 0.5387$. We obtained these parameters by sampling from $\boldsymbol{\epsilon}_w$ to estimate (16) as the ground-truth and fit these parameters for 10^5 epochs. For fitting the curves, the input range is limited to $\log \alpha \in$
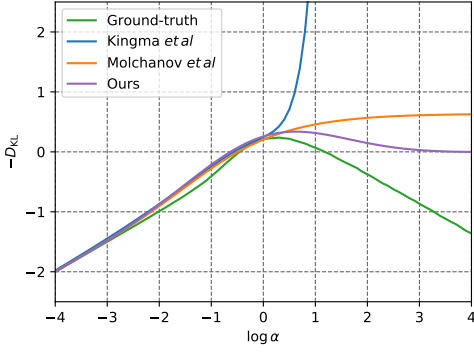
Fig. 4: Approximation to (16). Our approximation allows $\alpha > 1$ (c.f., [30]) and does not push $\alpha \to 0$ to generate a collapsed model (c.f., [42]).

Algorithm 1: Pseudocode for training BN^3.

Input: Parameters $\{\theta_{ij}^{(l)}, \alpha_{ij}^{(l)}, \beta_j^{(l)}\}_{ij}^{(l)}$, data $\{\mathbf{X}, \mathbf{Y}\}$, layer input $\mathbf{H}^{(0)} = \mathbf{X}_{\mathrm{batch}}$, total KL $\bar{\mathrm{KL}} = 0$.

1: **while** the network is not converged **do**
2:     **for** $l = 1 : L$ **do**
3:         Sample $\mathbf{b}^{(l)} = [b_{mj}^{(l)}]_{mj}$ with $\alpha_{ij}^{(l)}$, and (19) for dense layers or (A22) for convolution layers.
4:         Sample $\mathbf{z}^{(l)} \sim q_{\boldsymbol{\beta}^{(l)}}(\mathbf{z})$ and $\mathbf{F}^{(l)} = [f_{mj}^{(l)}]_{mj} = b_{mj}^{(l)} z_j^{(l)}$.
5:         Compute KL using (18), $\bar{\mathrm{KL}} \leftarrow \bar{\mathrm{KL}} + \mathrm{KL}$.
6:         $\mathbf{H}^{(l+1)} = \mathbf{F}^{(l)}$.
7:     **end for**
8:     Compute loss as $\mathcal{L}_{\mathcal{D}}(\mathbf{Y}_{\mathrm{batch}}, \mathbf{F}^{(l)})$, $\mathcal{L} = \mathcal{L}_{\mathcal{D}} - \bar{\mathrm{KL}}$.
9:     Compute $\frac{\partial \mathcal{L}}{\partial \theta}, \frac{\partial \mathcal{L}}{\partial \beta}, \frac{\partial \mathcal{L}}{\partial \alpha}$, update the network.
10: **end while**

$[-5, 0.5]$. As shown in Figure 7, our parameterization allows $\alpha > 1$ and maximizing $-\mathrm{KL}$ does not push α to infinity (c.f. [30] and [42]), providing *more flexible* choices for the weight variance.

The pseudocde for training the proposed network is shown in Algorithm 1. For testing, only lines 2-7 are executed.

As the prior of the zero-component w_{ij} is assumed a spike at zero with a small constant variance, we let $q(w_{ij}|z_j = 0)$ be the same spike as in Section 3.2 to save computation. Also, to speed up the sampling process in Figure 3, we directly multiply the sampled masks with the output features of the layer. This saves the cost for sampling from $w_{ij}|z_j = 0$ and simplifies (12) to $\mathbf{e}^T \boldsymbol{\kappa}_{\boldsymbol{\theta}}^1 \mathbf{J}_L^T \boldsymbol{\beta}$. Therefore, the KL divergence (9) is simplified to

$$\mathrm{KL}[q_{\boldsymbol{\beta}, \boldsymbol{\theta}}(\mathbf{W}, \mathbf{z}) || p(\mathbf{W}, \mathbf{z})] = \Phi_1 + \Phi_2 \quad (18)$$
$$= \boldsymbol{\beta}^T (\log(\boldsymbol{\beta}) - \log((\mathbf{e} - \tilde{\boldsymbol{\pi}})^T \boldsymbol{J}_L \boldsymbol{\pi})) + \mathbf{e}^T \boldsymbol{\kappa}_{\boldsymbol{\theta}}^1 \mathbf{J}_L^T \boldsymbol{\beta},$$

where $\boldsymbol{\kappa}_{\boldsymbol{\theta}}^1$ is calculated by applying (17) element-wise.

Using the notation in Section 3.2, the output of a fully connected layer is

$$f_{mj} = b_{mj} z_j, \quad b_{mj} \sim \mathcal{N}(\gamma_{mj}, \delta_{mj}), \quad (19)$$

$$\gamma_{mj} = \sum_{i=1}^{d} h_{mi} \theta_{ij}, \quad \delta_{mj} = \sum_{i=1}^{d} h_{mi}^2 \alpha_{ij} \theta_{ij}^2.$$

The sampling process is similar to that of [30], [42], [37]. This can be easily extended to convolutional layers with the ordering applied to channels (see Appendix B.1).

3.5 Distillation to Deterministic Nested Nets

Using VND and the probabilistic treatment, the BN^3 is able to learn the importance of parameters from data. With a decreasing order, the importance of parameters decays with an optimal trajectory instead of a hand-crafted one.
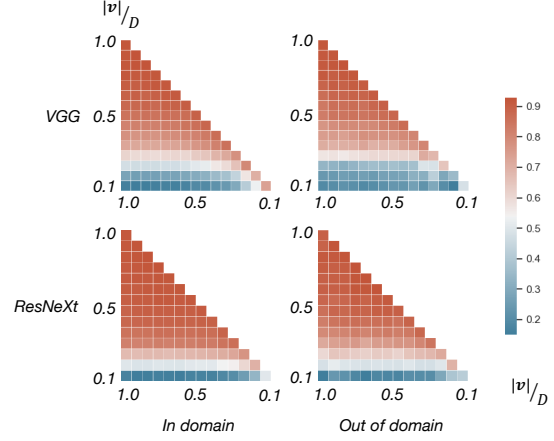

Fig. 5: The correlation between two output sample sets of BN^3. VGG11 and ResNeXt are used as the backbones. The models are trained on the "in domain" data. The correlation are evaluated on both "in domain" and "out of domain" data. Red color indicates a high correlation.
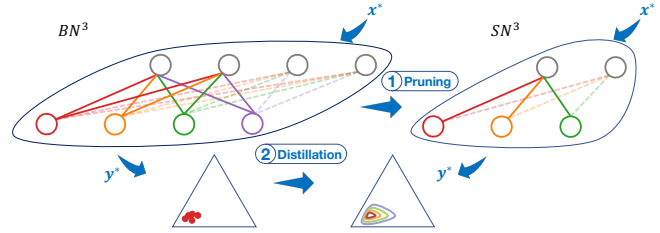

Fig. 6: The two step slimming to obtain a high performance deterministic network, SN^3. The triangle represents a 2-simplex with 3 classes. The left red points are the output of BN^3 given input $\mathbf{x}^*$, $\mathrm{BN}^3(\mathbf{x}^*)$. The right contours represent the Dirichlet distribution parameterized by $\mathrm{SN}^3(\mathbf{x}^*)$.

For pursuing a lighter neural network and a more practical inference speed, we can distill the Bayesian knowledge from BN^3 to its deterministic counterpart. This is motivated by an analysis of correlation of outputs of BN^3 with different widths. As shown in Figure 5, we conduct correlation analysis on two BN^3 with VGG11 and ResNeXt as backbones. For each network and a particular dataset, we obtain the uncertainty of the output under different widths,

$$\mathcal{U} = [U_{\boldsymbol{\theta}|\mathbf{v}}]_{\frac{|\mathbf{v}|}{D} = 0.1, 0.2, \ldots, 1.0},$$
$$U_{\boldsymbol{\theta}|\mathbf{v}} = [U_{\boldsymbol{\theta}|\mathbf{v}, i}]_{i=1}^N = [U[\mathbb{E}_{q(\boldsymbol{\theta}|\mathbf{z})}[p(y|\mathbf{x}_i, \boldsymbol{\theta})]]]_{i=1}^N,$$

where the $\frac{|\mathbf{v}|}{D} = 0.1$ means 10% of BN^3 parameters are used. We use $|\mathbf{v}|$ to indicate the number of non-zero entries in $\mathbf{v}$. The conversion from $\mathbf{v}$ to $\mathbf{z}$ is illustrated in Section 3.3. A common choice for U is the entropy. For calculating the correlation between the uncertainties $[U_{\boldsymbol{\theta}|\mathbf{v}_i}, U_{\boldsymbol{\theta}|\mathbf{v}_j}]$ for any pair $[\mathbf{v}_i, \mathbf{v}_j]$, the Pearson correlation coefficient is applied and visualized in Figure 5.

With in-domain data, strong correlation of uncertainty could be observed with width greater than $0.2 \sim 0.25$. This number is reduced to less than 0.2 with the out-of-domain data. This motivates us to prune the BN^3, then distill the original BN^3 uncertainty to the pruned BN^3 for smaller model size and better performance, as shown in Figure 6. As a knowledge distillation process is adopted, we name it Student Nested Neural Network (SN^3).

The first step is to prune the less useful weights according to the correlation analysis, to generate the base of SN^3. The mean of the nodes and the VND parameters from BN^3 are kept, while the variances in the variational parameters are discarded.

The second step produces a distillation from a Bayesian neural network to a deterministic neural network. For better capturing the uncertainty from BN^3, we adapt the loss function introduced in [12] to our problem, which distills the dark knowledge from a Bayesian neural network (BN^3) to a parameterized Dirichlet (SN^3). Specifically, a Maximum Mean Discrepancy (MMD) is used as the probabilistic distance metric, between the generated samples from BN^3 and a Dirichlet distribution parameterized by the output of SN^3.

$$\mathrm{MMD}_{\mathcal{K}}(q, \mathrm{Dir}) \tag{20}$$
$$= \sup_{\substack{\Psi \in \mathcal{H}_{\mathcal{K}}, \\ ||\Psi||_{\mathcal{H} \leq 1}}} \mathbb{E}_{q(\bar{\mathbf{z}})}[\mathbb{E}_{q(\theta|\mathbf{z})}[\Psi(BN^3(\mathbf{x}))] - \mathbb{E}_{s \sim \mathrm{Dir}(SN^3_{\bar{\mathbf{z}}}(\mathbf{x}))}[\Psi(s)]]$$

Note that $\mathcal{H}_{\mathcal{K}}$ is a reproducing kernel Hilbert space defined by a positive-definite kernel $\mathcal{K}$. Ψ is known as the critic. Dir is the Dirichlet distribution whose parameters are given by the output of SN^3.

Different from the formulation in [12], during the distillation, the size of SN^3 and BN^3 should vary together, controlled by sampling from $q(\bar{\mathbf{z}})$ ($q(\bar{\mathbf{v}})$), where $\bar{\mathbf{z}}$ and $\bar{\mathbf{v}}$ are the VND random variable for the pruned SN^3. $\mathbf{v}$ is no longer sampled from $q(\mathbf{v})$ but determined by $\bar{\mathbf{v}}$

$$|\mathbf{v}| = r|\bar{\mathbf{v}}|, \tag{21}$$

where $|\mathbf{v}|$ is the number of dimensions in vector $\mathbf{v}$, and r is the ratio of parameters between BN^3 and SN^3. For example, suppose a layer in BN^3 has 16 nodes, the corresponding layer of SN^3 is initialized with 4 nodes by the correlation analysis, then $r = 16/4 = 4$.

4 VND FOR GENERATIVE MODELS

Variational nested dropout encodes the importance of dimensions and optimizes by learning from data. Another interesting branch is to order the latent distributions of generative models. The latent variables with specified orders are expected to implicitly learn information with different levels of importance. Due to the stochastic nature of the variational nested dropout, we study two Bayesian deep generative models: variational auto-encoder and probabilistic UNet.

4.1 Ordered Encoding Variational Bayes

4.1.1 Overview

The proposed variational nested dropout is useful for organizing the latent space of the generative models, like variational auto-encoders. We propose VND-AE, which generates diverse samples compared by organizing the latent space.

To introduce the methodology, we show the basic form of prior using a *Gaussian-Bernoulli chain* distribution as the latent distribution for a generative model. The prior is based on the $\mathrm{BernChain}(\mathbf{z}, \boldsymbol{\pi})$, introduced in Section 2.2, with random variable $\mathbf{z} = [z_1, \ldots, z_D]$ for the mask and $\boldsymbol{\pi} = [\pi_1, \ldots, \pi_D]$ for the parameters. We let a uni-variate Gaussian variable conditioned on each node of the Bernoulli chain,

$$p(h_i \mid z_i) = z_i \, \mathcal{N}(h_i|0, \sigma^2) + (1 - z_i) \, \mathcal{N}(h_i|0, \sigma_\infty^2) \tag{22}$$

where z_i is sampled from the above Bernoulli chain. h_i is the i-th latent representation for $\mathbf{x}$. σ_∞^2 represents a constant variance with large value. Note that, the log-uniform prior (Section 3.4) or other distributions could also adopted for the conditional variable $h_i|z_i$ here.
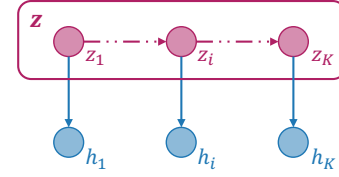

Fig. 7: The graphical representation of the Gaussian-Bernoulli chain prior.
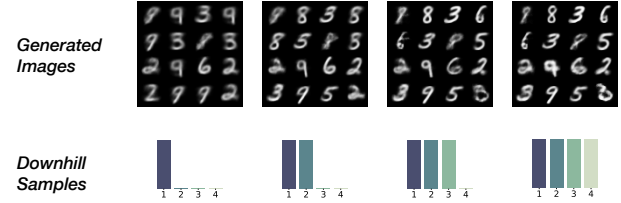

Fig. 8: Digit images generated from 16 samples of $\mathbf{z}$ (the 4x4 grid of digits) using different latent variable (code) lengths (increasing from left to right).

Each node of the latent representation could be regarded as a spike-and-slab model [43]. Traditionally, such models use *independent* Bernoulli r.v.s on the latent z_i's to control the sparsity through randomly "selecting" an h_i. However, due to the independence, there is no specific order for the selected variables. Different from the traditional models, our proposed prior is chained by the sequence of Bernoulli variables. As a result, there is a strict order due to the Bernoulli chain, as the $(i-1)$-th failure would stop the sampling of subsequent variables.

The approximate posterior are univariate Gaussian variables conditioned on the Downhill distribution presented in Section 2.3. The sampling process of the posterior is

$$\mathbf{z}|\mathbf{x} \sim \mathrm{Downhill}(\cdot|\mathrm{f}^{\mathbf{z}}(\mathbf{x}), \tau)$$
$$h_i|z_i, \mathbf{x} \sim z_i \, \mathcal{N}(h_i|f_i^{\boldsymbol{\mu}}(\mathbf{x}), f_i^{\boldsymbol{\sigma}^2}(\mathbf{x})) + (1 - z_i) \, \mathcal{N}(h_i|0, \sigma_\infty^2) \tag{23}$$

where $f_i^{\boldsymbol{\mu}}(\cdot)$ and $f_i^{\boldsymbol{\sigma}}(\cdot)$ are the encoder network for mean and variance of latent variables, while $f^{\mathbf{z}}(\mathbf{x})$ generates parameters for the Downhill distribution.

As the Downhill distribution is fully re-parameterizable, we let $\mathbf{z}$ be conditioned on $\mathbf{x}$ for learning a flexible representation. The latent variables (codes) for different inputs may have varying effective latent dimension (code length). Thus, the VND-VAE is trained to learn the optimal code length for different input $\mathbf{x}$. The evidence lower bound (ELBO) is written as:

$$\log p(\mathbf{x}) \geq \underbrace{\mathbb{E}_{\mathbf{h}, \mathbf{z}}[\log p(\mathbf{x}|\mathbf{h}, \mathbf{z})]}_{\text{reconstruction term}} - \underbrace{\mathrm{KL}(\mathrm{q}(\mathbf{h}, \mathbf{z}|\mathbf{x})||\mathrm{p}(\mathbf{h}, \mathbf{z}))}_{\text{KL term}} \tag{24}$$

A toy example on the image generation task is shown in Figure 8, where different length codes are used for generation. During testing time, we use fixed Downhill samples to control the code length, by which the effective dimension of $\mathbf{h}$ is controlled. We can observe, the first dimensions generate blurred "meta digits" with regular fonts and similar styles. With longer code length, the digits are sharpened and transformed to different styles. Thus, the VND-AE realizes the encoding of ordered information into the latent probabilistic distributions.

We present detailed discussion of the KL divergence in Appendix A.5, to understand how the VND affects the training objective of variational auto-encoder.
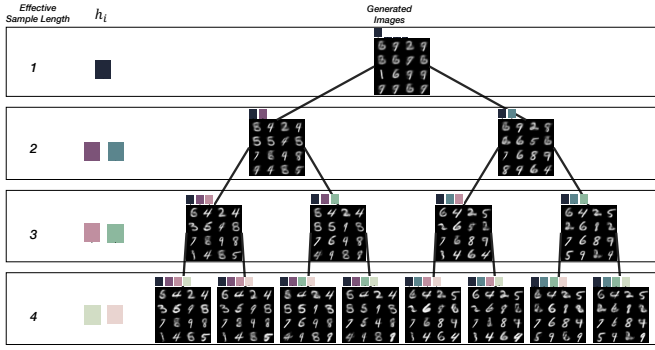
Fig. 9: The generated images given specific samples of $\mathbf{h}$. Different colored blocks for h_i represent different values sampled from a univariate Gaussian. The group of colored blocks at the top left corner of a image represents the sample $\mathbf{h}$. In this example, the maximum length of $\mathbf{h}$ is 4.

4.1.2 Discussion of Diversity

We discuss the diversity of learned approximate posterior. For the single-modal prior like Gaussian with diagonal matrix as the covariance matrix, the diversity is poor as the posterior will seek to approximate to that mode even if a positive lower bound is given (Appendix A.5). Our prior has a mixture structure as shown in Figure 8 and (22). Different from a vanilla mixture, the first several modes are weighted with higher importance, which decays with increasing index value.

The posterior shares similar multi-modal property with the prior, but with higher efficiency for sampling. During training, the approximation is done by selecting a chain of modes with a strict order under the mode-seeking reverse KL. A simple guarantee of posterior diversity is that, the optimal code length is greater than 1 (the case of *single-modal*). The *single-modal* case corresponds to $\beta_1 = 1$ and $\beta_i = 0, \forall i > 1$. We show this case *never* happens due to our design of posterior in Appendix A.5. The approximation that collapses to a single mode is thus avoided.

The diversity is empirically illustrated via the toy example used in Section 4.1.1, with a different sample generation method, as shown in Figure 9. After training VND-AE, the samples $\mathbf{h}$ are drawn from the latent space with fixed Downhill samples $\mathbf{z}$. Except for the first dimension, we give each dimension i two choices of h_i. This forms a tree structured data generation process, which conforms to the induced nested (hierarchical) structure of the latent space. At each level, with the increase of the sample length, the generated images are sharpened and refined. More importantly, different h_i's at level i leads to different writing styles or output results. The data generation process in this toy example also conforms with the training process; as the first several levels with shorter code lengths are sampled more often, the corresponding outputs in the first levels are more generic. VND-AE progressively enriches the diversity by controlling the code length. Samples at a greater code length level are more diverse, as it involves more modes in the mixture. By comparion, VAE and its variants see a single-modal multivariate distribution during the whole training process, which is not beneficial for the diversity.

4.2 Ordered Encoding Aleatoric Uncertainty

The probabistic UNet [32] and its variants have shown the unique property of encoding the aleatoric uncertainty. The methodology is to jointly use a deterministic neural network and a VAE whose prior is conditioned on the input $\mathbf{x}$. The posterior is conditioned on

both $\mathbf{x}$ and $\mathbf{y}$ such that it could help capture the noise from data. This is useful when there are observable noise from the labels, e.g., the case when multiple experts provide labels with disagreement for the same input.

We apply VND to the probabilistic UNet to effectively bring ordered information to its latent distributions. We name this variant VND-PUNet where "P" refers to probabilistic. As the prior is not a fixed probability distribution and requires gradients, we let it be a parameterized Downhill distribution in our VND-PUNet,

$$\mathbf{z}|\mathbf{x} \sim p(\mathbf{z}|\mathbf{x}) = \text{Downhill}(f^{\mathbf{z},\text{pr}}(\mathbf{x}), \tau), \qquad (25)$$

$$h_i|z_i, \mathbf{x} \sim p(h_i|z_i, \mathbf{x}) = z_i \mathcal{N}(h_i|f_i^{\boldsymbol{\mu},\text{pr}}(\mathbf{x}), f_i^{\boldsymbol{\sigma}^2,\text{pr}}(\mathbf{x})), \quad (26)$$

where $f^{\mathbf{z},\text{pr}}, f_i^{\boldsymbol{\mu},\text{pr}}, f_i^{\boldsymbol{\sigma}^2,\text{pr}}$ are the encoder networks for the prior. By taking $\mathbf{y}$ as the input for the VAE part, the posterior is written as,

$$\mathbf{z}|\mathbf{x}, \mathbf{y} \sim q(\mathbf{z}|\mathbf{x}, \mathbf{y}) = \text{Downhill}(f^{\mathbf{z},\text{po}}(\mathbf{x}, \mathbf{y}), \tau), \qquad (27)$$

$$h_i|z_i, \mathbf{x}, \mathbf{y} \sim q(h_i|z_i, \mathbf{x}, \mathbf{y})$$
$$= z_i \mathcal{N}(h_i|f_i^{\boldsymbol{\mu},\text{po}}(\mathbf{x}, \mathbf{y}), f_i^{\boldsymbol{\sigma}^2,\text{po}}(\mathbf{x}, \mathbf{y})), \quad (28)$$

where $f^{\mathbf{z},\text{po}}, f_i^{\boldsymbol{\mu},\text{po}}, f_i^{\boldsymbol{\sigma}^2,\text{po}}$ are the encoder networks for the posterior.

We adapt the training objective in [32] to the VND-PUNet,

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = \mathbb{E}_{\mathbf{z} \sim q(\mathbf{z}|\mathbf{y}, \mathbf{x})}[\mathbb{E}_{\mathbf{h} \sim q(\mathbf{h}|\mathbf{z}, \mathbf{y}, \mathbf{x})}[\log p(\mathbf{y}|f_{\text{base}}(\mathbf{x}, \mathbf{h}))]]$$
$$- \text{KL}(q(\mathbf{h}, \mathbf{z}|\mathbf{y}, \mathbf{x})||p(\mathbf{h}, \mathbf{z}|\mathbf{x})) \tag{29}$$

where f_{base} is the deterministic neural network, e.g., a UNet, which takes $\mathbf{x}$ and the samples from the latent space as the input. The KL terms is written as

$$\text{KL}[q(\mathbf{h}, \mathbf{z}|\mathbf{y}, \mathbf{x})||p(\mathbf{h}, \mathbf{z}|\mathbf{x})]$$
$$= \sum_i f_i^{\mathbf{z},\text{po}}(\mathbf{x}, \mathbf{y}) \Big[\log \frac{f_i^{\mathbf{z},\text{po}}(\mathbf{x}, \mathbf{y})}{f_i^{\mathbf{z},\text{pr}}(\mathbf{x})}, +\kappa_i(\mathbf{x}, \mathbf{y}) \Big] \tag{30}$$

where $\kappa_i(\mathbf{x}, \mathbf{y})$ is the KL divergence between two univariate Gaussian variables of h_i. During testing time, the samples are obtained from the prior distribution via (25-26), $\mathbf{z}^* \sim p(\mathbf{z}|\mathbf{x}^*), \mathbf{h}^* \sim p(\mathbf{h}|\mathbf{z}^*, \mathbf{x}^*)$, and then fed to the deterministic neural network to obtain the results $p(\mathbf{y}^*|f_{\text{base}}(\mathbf{x}^*, \mathbf{h}^*))$.

5 RELATED WORK

In this section, we reviewed deep nets with ℓ_0-regularization and nested nets, while the comparisons with Bayesian neural network are elaborated in Section 3.4. The related generative models are discussed, as well.

ℓ_0-**regularization.** The Bernoulli-Gaussian linear model with independent Bernoulli variables is shown to be equivalent to ℓ_0 regularization [43]. Recent works [39], [62] investigate ℓ_0 norm for regularizing DNNs. [39] presents a general formulation of a ℓ_0-regularized learning objective for a single deterministic neural network,

$$\min_{\tilde{\boldsymbol{\theta}}, \tilde{\boldsymbol{\pi}}} \mathbb{E}_{q(\tilde{\mathbf{z}}|\tilde{\boldsymbol{\pi}})}[L_D(\tilde{\boldsymbol{\theta}})] + \lambda \sum_{j=1}^{|\tilde{\boldsymbol{\theta}}|} \tilde{\pi}_j, \tag{31}$$

where the variable $\tilde{\mathbf{z}}$ is a binary gate with parameter $\tilde{\pi}_j$ for each network node $\tilde{\theta}_j$, and L_D is the loss. It was shown that ℓ_0-regularization over the weights is a special case of an ELBO over parameters with spike-and-slab priors. These works present the *uniform* ℓ_0-regularization as the coefficient λ is a constant over the weights. It is interesting that our training objective (7) can be viewed as a generalization of a new training objective of

deterministic networks, which includes a weighted penalization over the choices of sub-networks, interpretable as *an ordered ℓ_0-regularization* (13).

Nested neural networks. Nested nets have been explored in recent years, for its portability in DNN deployment on different platforms. [28] proposes a network-in-network structure for a nested net. which consists of internal networks from the core level to the full level. [64], [63] propose slimmable NN that trains a network that samples multiple sub-networks of different channel numbers (widths) simultaneously, where the weights are shared among different widths. The network needs to switch between different batch normalization parameters that correspond to different widths. To alleviate the interference in optimizing channels in slimmable NN, [7] proposes a once-for-all network that is elastic in kernel size, network depth and width, by shrinking the network progressively during training. [11] proposes using *nested dropout* to train a fully nested neural network, which generates more sub-networks in nodes, including weights, channels, paths, and layers. However, *none* of the previous works consider learned importance over the nodes and the predictive uncertainty. Our work provides a well-calibrated uncertainty and the learned importance, with a full Bayesian treatment of nested nets.

Variational auto-encoder. The variational auto-encoder (VAE) was introduced in [31] to bring scalability and training stability for the variational learning of latent variable models. The reparameterization trick for the latent variables and stochastic variational Bayes framework provides the fundamental support for the scalable variational inference. β-VAE [24] augments the VAE with a single hyper-parameter β which helps to quantify the degree of learnt disentanglement. Joint-VAE [13] further augments a discrete latent unit for discovering the categorical factors from the latent space. Other works study different probabilistic distance metrics like Wasserstein distance [58]. Sliced-Wasserstein auto-encoder [33] regularizes the auto-encoder loss with the sliced-Wasserstein distance between the distribution of the encoded training samples and a samplable prior distribution. Different from these works, our VND-AE builds an explicit ordered structure over the latent distributions, showing better diversity while maintaining a high generation performance.

Probabilistic UNet. The family of probabilistic UNet [32] is designed for capturing the aleatoric uncertainty, when the labels of data are noisy. For example, there are multiple doctors provides segmentation masks for the same histopathological image. The VAE part of probabilistic UNet encodes both the input and output into the posterior. The epistemic uncertainty is captured by pairing a UNet as the backbone for segmentation. During testing time, only the prior is used for sampling different possible segmentation masks. Our VND-PUNet place the order structure into the latent space of the input image and labels, which helps organize the underlining latent space that learns aleatoric uncertainty.

6 EXPERIMENTS

We next present experiments showing that VND can be applied to a variety of model architectures and tasks. The experiments include three main tasks: image classification using BN^3 and SN^3, image generation for VND-AE, and probabilistic semantic segmentation for VND-PUNet.

6.1 Image Classification with BN^3

6.1.1 Experiment setup

Dataset and Models. The image classification experiments are conducted on Cifar10 SVHN and Tiny Imagenet (see Appendix C.4 for results on Cifar100). The tested NN models are VGG11 with batch normalization layers [54], ResNeXt-Cifar model from [60], and MobileNetV2 [53].

Methods. To train the proposed Bayesian nested neural network (denoted as BN^3), we use the cross-entropy loss for the expected log-likelihood in (8). The computation of the KL term follows Section 3.4. For ordering the nodes, in every layer, we assign each dimension of the prior (Bernoulli chain) and posterior (Downhill variable) of the ordering unit to a group of weights. Thus, the layer width is controlled by the ordering unit. We set the number of groups to 32 for VGG11 and ResNext-Cifar, and to 16 for MobileNetV2. We compare our BN^3 with the fully nested neural network (FN^3) [11], since it is an extension of slimmable NN [64], [63] to fine-grained nodes. We also compare with the Bayesian NN with variational Gaussian dropout [30], where we train a set of independent Bayesian NNs (IBNN) for different fixed widths. Conceptually, the performance of IBNN, which trains separate sub-networks, is the ideal target for BN^3, which uses nested sub-networks. In addition, the deterministic BN^3 (DT-BN^3) which only uses the mean of posterior is considered. For comparing the calibration performance, we conduct the temperature scaling [19] using the sub-networks of FN^3 (FN^3-TS).

During testing time, we generate fixed width masks for BN^3 and FN^3 as in [11]. We re-scale the node output by the probability that a node is kept (see Appendix B.2). The batch normalization statistics are then re-collected for 1 epoch with the training data (using fewer data is also feasible as shown in Appendix C.2). The number of samples used in testing BN^3 and IBNN is 6. The detailed hyper-parameter settings for training and testing are in Appendix C.1.

Evaluation metrics. For the evaluation, we test accuracy, uncertainty calibration, and out-of-domain (OOD) detection. Calibration performance is measured with the expected calibration error [19] (ECE), which is the expected difference between the average confidence and accuracy. The OOD detection performance is evaluated on the SVHN dataset. OOD performance is measured with the area under the precision-recall curve (AUPR) [6], [22], [35] (see Appendix C.3 for AUROC curves). If we take the OOD class as positive, precision is the fraction of detected OOD data that are true OOD, while recall is the fraction of true OOD data that are successfully detected. Note that a better model will have higher accuracy and OOD AUPR, and lower calibration ECE. As the sampling and collection of batch-norm statistics are stochastic, we repeat each trial 3 times and report the average results.

6.1.2 Results

By design, BN^3 and DT-BN^3 are expected to have better performance than FN^3 and FN^3-TS when changing the size of the neural network, while IBNN should have the best performance as it trains a network again when the size changes. The results are presented in Figs. 10 and 11. First, looking at performance versus width, BN^3 exhibits the well-behaved property of sub-networks, where the performance increases (accuracy and AUPR increase, ECE decreases) or is stable as the width increases. This demonstrates that the variational ordering unit successfully orders the information within each layer.
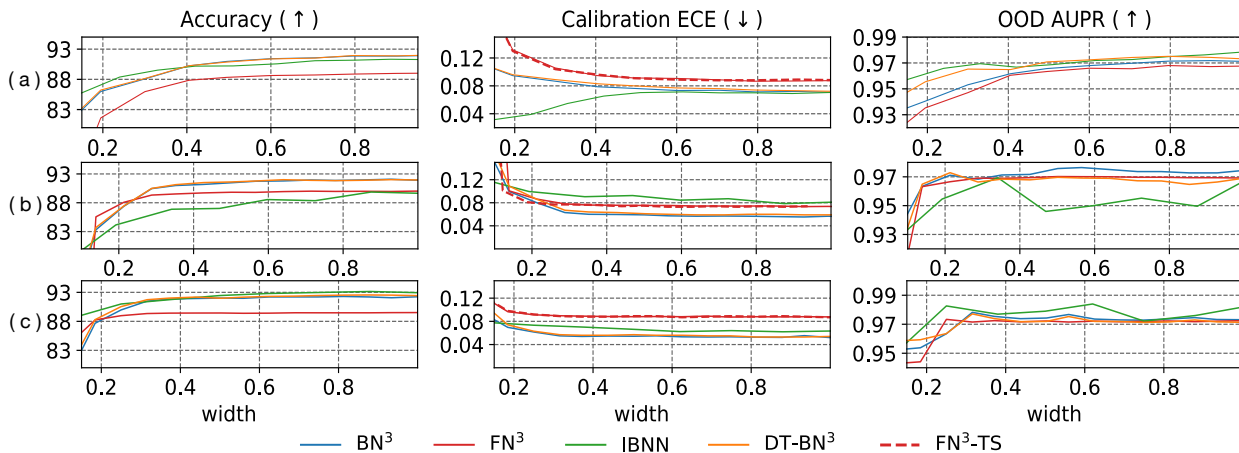
Fig. 10: Results on Cifar10 for (a) VGG11, (b) MobileNetv2, and (c) ResNeXt-Cifar. Each curve plots performance versus the network width.
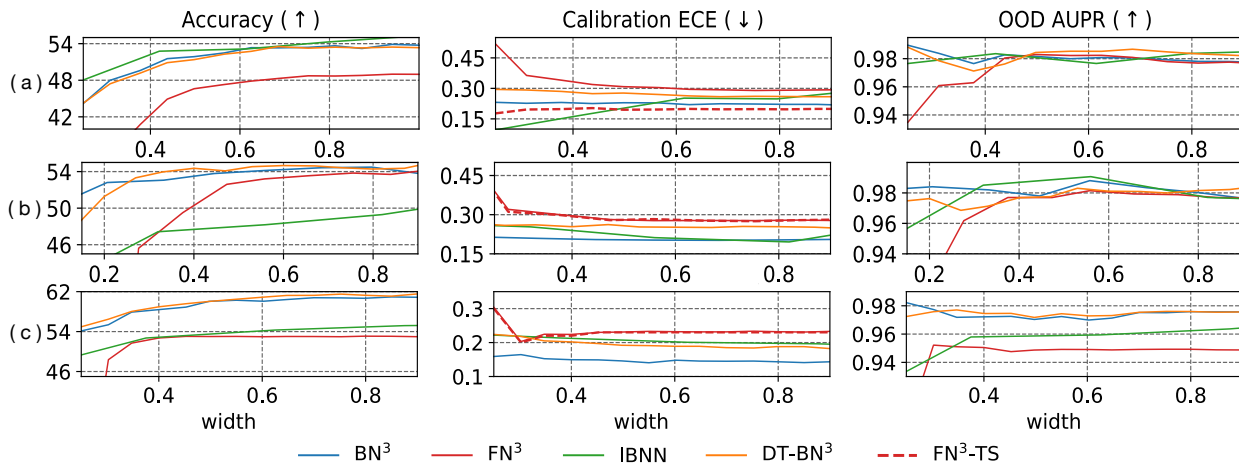

Fig. 11: Results on Tiny ImageNet for (a) VGG11, (b) MobileNetv2, (c) ResNeXt-Cifar.

Despite learning nested sub-networks, in general, BN^3 has similar performance as IBNN (which separately learns sub-networks) for all models and datasets, with the following exceptions. For MobileNetV2 on both datasets, BN^3 outperforms IBNN in all metrics, as IBNN fails to perform well in prediction and uncertainty (outperformed by FN^3 too). For VGG11 on both datasets, IBNN tends to have lower ECE with smaller widths, showing its advantage in providing uncertainty for small and simple models. However, IBNN has larger ECE when the model size is large, e.g., BN^3 has lower ECE than IBNN with the ResNeXt model. Finally, BN^3 outperforms IBNN by a large margin for ResNeXt on Tiny ImageNet, which we attribute to its ability to prune the complex architecture via learning ordered structures (Section 2.3) and the ordered ℓ-0 regularization effect (Section 3.3), which are absent in IBNN. Comparing the two nested models, BN^3 outperforms FN^3 in all metrics, which shows the advantage of learning the nested dropout rate for each node. The temperature scaling method (FN^3-TS) improves calibration under few settings, but is not scalable when the number of sub-networks is large.

Comparisons with SNR-based pruning methods. We show the comparisons of BN^3 and signal-to-noise (SNR) ratio based pruning methods [5] in Table 1. For the latter method, we train a full-width Bayes Neural Network as IBNN and gradually prune its weights with lowest signal-to-noise ratio. It could be con-cluded that the SNR-based method could provide a comparable performance with BN^3 when there is no much removed weights. However, the performance drops quickly after the percentage reaches 60%.

TABLE 1: Performance of BN^3 and SNR-based pruning methods

Cifar10				
Removed weights (%)	Acc. (↑)		ECE (↓)	
	SNR	BN^3	SNR	BN^3
0	90.4	91.8	0.076	0.071
20	90.3	91.9	0.076	0.070
40	90.4	91.5	0.076	0.072
60	89.4	91.1	0.080	0.080
80	29.6	85.6	0.494	0.097
Cifar100				
Removed weights (%)	Acc. (↑)		ECE (↓)	
	SNR	BN^3	SNR	BN^3
0	69.4	70.1	0.19	0.16
20	69.2	69.9	0.18	0.16
40	67.5	68.7	0.20	0.17
60	61.2	65.1	0.20	0.16
80	40.0	57.4	0.31	0.15

Efficiency. We compare the performance with limited number of parameters of BN^3 with the sparse variational Bayes (SVB) [42] on Cifar10 with VGG11 in Table 2. The small-

est BN^3 (BN^3 with the least number of parameters by default) outperforms SVB, and has a similar size. MC inference (BN^3-MC) averages the predictions from the stochastic forward passes of different widths, and performs slightly worse than full-width BN^3 but uses fewer parameters.

TABLE 2: Performance with limited parameters, SVB, Monte-Carlo integration and full-width BN^3.

	Acc.($\uparrow$)	ECE($\downarrow$)	OOD($\uparrow$)	Weight usage (%)
smallest BN^3	83.5	0.10	0.94	6.2
SVB [42]	82.9	0.17	0.94	5.7
BN^3-MC	89.4	0.08	0.97	$46_{mean} \pm 27_{std}$
BN^3	92.6	0.07	0.97	100

Training of BN^3 is twice as slow as training a *single* BNN, but much faster than training IBNNs for large number of widths N. BNN test inference is slow when using sampling, but can be sped up by using the parameters' mean (denoted as DT-BN^3) with similar performance (see Figures 10 and 11).

TABLE 3: Training and testing time for BN^3/DT-BN^3, IBNN and FN^3.

	BN^3 / DT-BN^3	IBNN	FN^3
Train	14.9 h (BN^3)	$7.5 \times N$ h	5.6 h
Test	1.78 s (DT-BN^3)	1.68 s	1.82 s

Knowledge distillation. We perform the knowledge distillation experiments on the harder Tiny Imagenet dataset. We test two teacher networks, VGG11 and ResNeXt-Cifar, as their number of channels is power of 2, which is suitable for pairing the teacher network and student network. According to the correlation analysis, we keep 25% of channels for SN^3 for both networks. For uncertainty calibration, we leave the last layer of SN^3 to be a Bayesian layer with stochastic nodes described in Section 3.4. The numerical results are shown in Table 4.

TABLE 4: Performance of SN^3 with different sizes and relative sizes.

Model	Rel. Size	Accuracy(%)	ECE	OOD AUPR
VGG11-BN^3-100%	1	54.00	0.22	0.980
VGG11-SN^3-25%	1/16	54.98	0.21	0.981
VGG11-SN^3-50%	1/8	55.26	0.21	0.981
VGG11-SN^3-75%	3/16	55.32	0.20	0.980
VGG11-SN^3-100%	1/4	**55.36**	**0.20**	**0.982**
ResNeXt-BN^3-100%	1	61.19	0.15	0.977
ResNeXt-SN^3-25%	1/16	63.10	0.15	0.977
ResNeXt-SN^3-50%	1/8	63.52	0.15	0.976
ResNeXt-SN^3-75%	3/16	63.78	0.14	**0.980**
ResNeXt-SN^3-100%	1/4	**63.90**	**0.14**	**0.980**

Note that the size of SN^3-100% is 1/4 of BN^3-100%. With the correlation analysis, model pruning and distillation method, the SN^3 with the smallest width (1/16 of the original size) outperforms BN^3. By setting the last layer to be a stochastic layer with our Bayesian treatment, the uncertainty calibration and out-of-domain detection performance are improved through knowledge distillation.

6.2 Image Generation with VND-AE

6.2.1 Experiment setup
Dataset. The image generation experiments are conducted on five datasets across image domains: CelebA [36], Cifar10 [34],

TABLE 5: Frechet Inception Distance (FID) scores for image generation. Lower FID is better.

Model	Dim.	CelebA	Cifar10	Cifar100	3D Chairs	Chest X-ray
BetaVAE	100%	77.33	215.04	154.44	183.67	316.02
JointVAE	100%	67.95	154.57	154.67	134.31	296.57
SWAE	100%	78.33	119.52	118.26	109.15	369.72
VAE	25%	72.29	175.17	162.46	170.93	300.36
	50%	72.43	140.70	161.71	178.84	**282.62**
	75%	73.40	144.35	142.15	173.89	300.49
	100%	73.14	147.35	146.25	175.94	298.53
VND	25%	76.54	174.83	154.97	94.87	287.07
	50%	64.09	134.95	123.71	96.57	285.79
	75%	62.29	117.24	113.00	96.48	285.79
	100%	**62.80**	**114.12**	**111.95**	**96.44**	285.73

Cifar100 [34], 3D Chairs [2] and Chest X-ray [1]. CelebA is a large-scale face dataset with more than 200K celebrity images. 3D Chairs contains a total of 86,366 synthesized images sampled from 1,393 high-quality 3D chair models with 62 different viewpoints. Chest X-ray dataset contains 5,856 validated Chest X-Ray images.

Evaluation Metrics. We adopt the *Frechet Inception Distance score* (FID) [23], *Inception Score* (IS) [51] and *Reconstrction Error* as the evaluation metrics. FID measures the distance between feature vectors extracted by Inception Network for real and generated images. IS evaluates the quality of synthetic images output by generative models by measuring the KL divergence of generated images.

Methods We use a standard Variational Auto-Encoder (VAE) with five convolutional layers following with BatchNorm as the VAE backbone, and proposed VND-AE is developed based on this structure as well. LeakyRelu is selected as the activation function, and the full size latent dimension (code length) is set as 128. We compare a group of four latent dimensions [32, 64, 96, 128] of VND-AE and VAE, which corresponding to 25%, 50%, 75% and 100% of full latent dimension size, respectively. The training of VND-AE uses a model with latent dimension 128, and in testing time, we only truncate the latent dimension to the specified length. The training of VAE requires re-initializing the model and training for each particular latent dimension. Comparison with other VAE variants (BetaVAE [24], JointVAE [13] and SWAE [33]) are also presented, and their latent dimension is also 128.

6.2.2 Results
Tables 5, 6, 7 present the results on the 5 datasets for the 3 metrics. In general, VND-AE outperforms the VAE baseline for most metrics, and it requires *training only once* to obtain models with different latent dimensions, as expected. By comparisons, the VAE baseline requires repetitively training for different latent dimensions. Under most metrics and datasets, VND-AE with 50% latent dimensions already outperforms BetaVAE, JointVAE and SWAE.

Frechet Inception Distance score (FID). A lower FID indicates a closer distance between the pile of generated and real images. We can conclude from Table 5 that increasing latent dimension leads to better FID for both VAE and VND-AE, and VND-AE is generally superior. Although constructed with the same network structure, the ordered latent variables generated by VND-AE can achieve smaller FID than VAE. One exception is on the Chest X-ray dataset, VAE with 50% latent dimension outperforms VND-AE, while other latent dimensions have a large gap with VND-AE. Note that VND-AE provides consistently decent performance under different latent dimensions for this dataset. The images generated by VND-AE has better overlap with the

Fig. 12: Generated faces by VND-AE for 25%, 50%, 75%, 100% latent dimension (code length). As the code length increases, the face images become more detailed.

Fig. 13: VND generated chairs for 25%, 50%, 75%, 100% sample length from top to bottom. Each row shows generated chair with corresponding sample length.

TABLE 6: Inception scores (IS) for image generation. Higher IS is better.

Model	Dim.	CelebA	Cifar10	Cifar100	3D Chairs	Chest X-ray
BetaVAE	100%	1.60	2.19	2.90	3.37	1.08
JointVAE	100%	**1.85**	3.01	2.83	3.47	1.08
SWAE	100%	1.70	3.00	2.74	3.24	1.20
VAE	25%	1.68	2.71	2.67	3.07	1.11
	50%	1.68	3.26	2.60	3.05	1.12
	75%	1.66	3.19	3.19	3.04	1.12
	100%	1.65	3.10	2.99	3.04	1.02
VND	25%	1.56	2.83	2.73	3.51	1.28
	50%	1.69	3.36	**3.32**	**3.54**	1.28
	75%	1.74	**3.40**	3.23	**3.54**	1.27
	100%	1.75	3.36	3.15	3.53	**1.29**

distribution of the original image set. When latent dimensions are the same, VND-AE's FID score outperforms the other baselines across five datasets from various domains.

Inception score (IS). Higher IS indicates better diversity of the generated images. A comparison of VND-AE and VAE of different latent dimensions and other baselines is shown in Table 6. Like the observation on FID, a larger latent dimension leads to better IS performance, which means a greater diversity of images generated by the model. VND-AE outperforms VAE on this metric on each dimension, although VND-AE is trained only once while VAE had to be re-trained for each latent dimension. VND-AE outperforms other methods with higher IS on all datasets, except for JointVAE on CelebA. This might be due to JointVAE has the advantage of using a mixture of discrete codes and continuous codes for diversity. Extending VND-AE to discrete codes could be interesting future work.

Qualitative study. Besides the improvement on standard generation metrics, VND-AE also offers the possibility of component-by-part generation. Figure 12 shows sample face images generated by proposed VND-AE, for 4 latent dimensions (code lengths). The first row corresponds to 25% code length, and the increasing the code length (the next rows) introduces additional details to the previously generated image. These details include, but are not limited to, facial makeup, sunglasses, beard, mouth opening, gender switching, etc. This function is caused by VND-AEs's

TABLE 7: Reconstruction Error for image generation. Lower error is better.

Model	Dim.	CelebA	Cifar10	Cifar100	3D Chairs	Chest X-ray
BetaVAE	100%	1.338	1.286	0.842	0.904	1.823
JointVAE	100%	1.320	0.774	0.882	0.753	1.833
SWAE	100%	4.264	3.820	3.882	1.426	10.58
VAE	25%	1.277	0.947	1.048	1.360	1.517
	50%	1.276	0.752	0.984	1.382	1.561
	75%	1.277	**0.740**	0.913	1.381	1.523
	100%	1.279	0.743	0.827	1.380	1.529
VND	25%	1.299	1.094	1.131	0.661	0.856
	50%	1.052	0.865	0.899	0.618	0.836
	75%	1.008	0.810	0.843	0.618	0.832
	100%	**1.004**	0.805	**0.838**	**0.618**	**0.832**

explicit order constraints on the latent dimension during training. By adding order to the latent dimension during training, each group of dimensions learns specific features from common to infrequent. The component-by-part generation gives a way to produce images of a person with different expressions, angles and accessories in just one training process, or add artificial details to an already existing synthesized face. This property is also evident when jointly examining the reconstruction errors in Table 7 and IS. Generated images with 25% latent dimensions are with relatively low IS (low diversity), showing a "meta face" which is possible to be refined by adding more details. These "meta faces" at 25% also have high reconstruction error compared to the original images, since the original faces are detailed. As another example, Figure 13 shows how VND controls chair generation. Similar to face generation, VND adds more details when increasing the code length, including filling the back of the chair, adding chair legs, etc.

It is interesting to select the optimal depth for hierarchical VAE [8] with VND. We tested proposed VND on hierarchical VAE on cifar10 and ffhq256. The optimal depth is selected by the largest posterior probability, $k = \arg\max_k q(z_k)$. For the cifar10 dataset, we use a 76-layer hierarchical VAE and found the optimal depth is 72. For the FFHQ256 dataset, we use a 134-layer hierarchical VAE and found the optimal depth is 128. The qualitative results will be available in the github repo.

6.3 Lung Abnormalities Segmentation with VND-PUNet

6.3.1 Experiment setup

Dataset and Evaluation Metrics. The semantic segmentation experiments are conducted on the LIDC-IDRI [9] dataset, which contains 1,018 CT scans from 1,010 lung patients with manual lesion segments annotated by four experts. This dataset is uncertainty-critical as it contains typical ambiguities in labels that

appear in medical applications. We follow [32] to process the data, resulting in 12,870 images in total. We adopt the *generalized energy distance* (GED) [4], [52], [56], [32] as the evaluation metric, with $\delta(A, B) = 1 - \text{IoU}(A, B)$ as the distance function between two segmentation map distributions A, B. GED measures the distance between the output distributions rather than single deterministic predictions. For BN^3, it measure the probabilistic distances between the induced distribution from model posterior given a fixed width, and the noisy labels from the four experts.

Models. We use a standard UNet [49] for BN^3 and the number of groups is 32. We also present the results of VND-PUNet proposed in Section 4.2. We compare with Probabilistic UNet (P-UNet) [32], a deep ensemble of UNet (E-UNet), and Dropout UNet (D-UNet) [26]. Their results are the average results from [32] with the full UNet.

6.3.2 Results

The results are presented in Table 8. We observe that BN^3 outperforms the existing methods in most of the cases, with the difference more obvious when there are fewer posterior samples. The performance of BN^3 stabilizes after width of 32.29%. This indicates BN^3 learns a compact and effective structure compared with other methods, in terms of capturing ambiguities in the labels.

When there are more posterior samples (8 and 16), probabilistic UNet has better performance than the BN^3 with the smallest width ($6/32$ channels are preserved). This means with more posterior samples, the probabilistic UNet can depict the latent structure better, but requires a full-width model. Increasing the width to 32.29%, BN^3 then achieves better performance.

The VND-PUNet consistently outperforms other variants of probabilistic UNet and BN^3 with either $1/3$ or 100% latent dimensions. The diversity of generated samples are effectively enhanced by adopting the VND, resulting in better modeling of uncertainty.

7 CONCLUSION

In this paper, we propose a novel variational nested dropout (VND) that explicitly models the ordered information via our proposed Downhill random variable. From our model, the ordered information can be learned from data, rather than hand-tuned as with previous methods like nested dropout. We validate the effectiveness of the proposed VND on two applciations: constructing nested nets and variational generative models. Experiments show that VND can improve both accuracy and calibrated predictive uncertainty for the nested nets. It can enhance the performance of VAE in image reconstruction and diversity. On the tasks of capturing annotation noise by encoding aleatoric uncertainty, VND-PUNet and BN^3 outperform several variants of UNet and PUNet. Future work will study the VND in language modeling and sequential data where the ordering is also important, e.g., [45]. The Downhill random variable is a well-suited hidden variable for such applications.

REFERENCES

[1] Chest x-ray images for pneumonia detection with deep learning. https://www.kaggle.com/datasets/tolgadincer/labeled-chest-xray-images.

[2] Mathieu Aubry, Daniel Maturana, Alexei Efros, Bryan Russell, and Josef Sivic. Seeing 3d chairs: exemplar part-based 2d-3d alignment using a large dataset of cad models. In *CVPR*, 2014.

[3] Artur Bekasov and Iain Murray. Ordering dimensions with nested dropout normalizing flows. *arXiv preprint arXiv:2006.08777*, 2020.

[4] Marc G Bellemare, Ivo Danihelka, Will Dabney, Shakir Mohamed, Balaji Lakshminarayanan, Stephan Hoyer, and Rémi Munos. The cramer distance as a solution to biased wasserstein gradients. *arXiv preprint arXiv:1705.10743*, 2017.

[5] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International conference on machine learning*, pages 1613–1622. PMLR, 2015.

[6] Kendrick Boyd, Kevin H Eng, and C David Page. Area under the precision-recall curve: point estimates and confidence intervals. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 451–466. Springer, 2013.

[7] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.

[8] Rewon Child. Very deep vaes generalize autoregressive models and can outperform them on images. In *International Conference on Learning Representations*, 2020.

[9] Kenneth Clark, Bruce Vendt, Kirk Smith, John Freymann, Justin Kirby, Paul Koppel, Stephen Moore, Stanley Phillips, David Maffitt, Michael Pringle, et al. The cancer imaging archive (tcia): maintaining and operating a public information repository. *Journal of digital imaging*, 26(6):1045–1057, 2013.

[10] Yufei Cui, Ziquan Liu, Qiao Li, Antoni B. Chan, and Chun Jason Xue. Bayesian nested neural networks for uncertainty calibration and adaptive compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2392–2401, June 2021.

[11] Yufei Cui, Ziquan Liu, Wuguannan Yao, Qiao Li, Antoni B. Chan, Teiwei Kuo, and Chun Jason Xue. Fully nested neural network for adaptive compression and quantization. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 2080–2087. International Joint Conferences on Artificial Intelligence Organization, 7 2020.

[12] Yufei Cui, Wuguannan Yao, Qiao Li, Antoni B Chan, and Chun Jason Xue. Accelerating monte carlo bayesian prediction via approximating predictive uncertainty over the simplex. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[13] Emilien Dupont. Learning disentangled joint continuous and discrete representations. *Advances in Neural Information Processing Systems*, 31, 2018.

[14] Chelsea Finn, Lisa Anne Hendricks, and Trevor Darrell. Learning compact convolutional neural networks with nested dropout. *arXiv preprint arXiv:1412.7155*, 2014.

[15] Yarin Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

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

[17] Yarin Gal, Jiri Hron, and Alex Kendall. Concrete dropout. In *Advances in neural information processing systems*, pages 3581–3590, 2017.

[18] Emil Julius Gumbel. *Statistical theory of extreme values and some practical applications: a series of lectures*, volume 33. US Government Printing Office, 1948.

[19] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. *arXiv preprint arXiv:1706.04599*, 2017.

[20] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[21] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.

[22] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016.

[23] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[24] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

[25] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[26] Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv preprint arXiv:1511.02680*, 2015.

[27] Mohammad Emtiyaz Khan and Didrik Nielsen. Fast yet simple natural-gradient descent for variational inference in complex models. In *2018*

TABLE 8: Numerical results on semantic segmentation using generalized energy distance (↓) with different numbers of posterior samples.

	19%-BN[3]	46%-BN[3]	80%-BN[3]	Full-BN[3]	P-UNet	D-UNet	E-UNet	33%-VND-PUNet	Full-VND-PUNet
1 sample	0.51 ± 0.28	0.31 ± 0.25	0.29 ± 0.25	0.27 ± 0.27	0.69	1.31	0.83	0.14	**0.12**
4 sample	0.39 ± 0.12	0.24 ± 0.15	0.23 ± 0.15	0.23 ± 0.14	0.40	0.62	0.55	0.12	**0.10**
8 sample	0.34 ± 0.15	0.22 ± 0.17	0.21 ± 0.16	0.22 ± 0.17	0.32	0.53	0.49	0.07	**0.06**
16 sample	0.36 ± 0.14	0.21 ± 0.13	0.21 ± 0.13	0.21 ± 0.12	0.28	0.46	0.50	**0.05**	**0.05**

International Symposium on Information Theory and Its Applications (ISITA), pages 31–35. IEEE, 2018.

[28] Eunwoo Kim, Chanho Ahn, and Songhwai Oh. Nestednet: Learning nested sparse structures in deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8669–8678, 2018.

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

[30] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in neural information processing systems*, pages 2575–2583, 2015.

[31] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[32] Simon Kohl, Bernardino Romera-Paredes, Clemens Meyer, Jeffrey De Fauw, Joseph R Ledsam, Klaus Maier-Hein, SM Eslami, Danilo Jimenez Rezende, and Olaf Ronneberger. A probabilistic u-net for segmentation of ambiguous images. *Advances in neural information processing systems*, 31:6965–6975, 2018.

[33] Soheil Kolouri, Phillip E Pope, Charles E Martin, and Gustavo K Rohde. Sliced-wasserstein autoencoder: An embarrassingly simple generative model. *arXiv preprint arXiv:1804.01947*, 2018.

[34] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[35] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in neural information processing systems*, pages 6402–6413, 2017.

[36] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[37] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *Advances in neural information processing systems*, pages 3288–3298, 2017.

[38] Christos Louizos and Max Welling. Multiplicative normalizing flows for variational bayesian neural networks. *arXiv preprint arXiv:1703.01961*, 2017.

[39] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations*, 2018.

[40] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[41] Chris J Maddison, Daniel Tarlow, and Tom Minka. A* sampling. In *Advances in Neural Information Processing Systems*, pages 3086–3094, 2014.

[42] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. *arXiv preprint arXiv:1701.05369*, 2017.

[43] Kevin P Murphy. *Machine learning: a probabilistic perspective*. 2012.

[44] Daniel Povey, Xiaohui Zhang, and Sanjeev Khudanpur. Parallel training of dnns with natural gradient and parameter averaging. *arXiv preprint arXiv:1410.7455*, 2014.

[45] Piyush Rai, Changwei Hu, Ricardo Henao, and Lawrence Carin. Large-scale bayesian multi-label learning via topic-based label embeddings. In *Advances in Neural Information Processing Systems*, pages 3222–3230, 2015.

[46] Ali Razavi, Aäron van den Oord, Ben Poole, and Oriol Vinyals. Preventing posterior collapse with delta-vaes. *arXiv preprint arXiv:1901.03416*, 2019.

[47] Ao Ren, Tianyun Zhang, Shaokai Ye, Jiayu Li, Wenyao Xu, Xuehai Qian, Xue Lin, and Yanzhi Wang. Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 925–938, 2019.

[48] Oren Rippel, Michael Gelbart, and Ryan Adams. Learning ordered representations with nested dropout. In *International Conference on Machine Learning*, pages 1746–1754, 2014.

[49] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[50] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798, 2007.

[51] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016.

[52] Tim Salimans, Han Zhang, Alec Radford, and Dimitris Metaxas. Improving gans using optimal transport. *arXiv preprint arXiv:1803.05573*, 2018.

[53] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[54] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[55] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[56] Gábor J Székely and Maria L Rizzo. Energy statistics: A class of statistics based on distances. *Journal of statistical planning and inference*, 143(8):1249–1272, 2013.

[57] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[58] I Tolstikhin, O Bousquet, S Gelly, and B Schölkopf. Wasserstein autoencoders. In *6th International Conference on Learning Representations (ICLR 2018)*. OpenReview. net, 2018.

[59] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12), 2010.

[60] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[61] Shichao Xu, Yixuan Wang, Yanzhi Wang, Zheng O'Neill, and Qi Zhu. One for many: Transfer learning for building hvac control. *arXiv preprint arXiv:2008.03625*, 2020.

[62] Huanrui Yang, Wei Wen, and Hai Li. Deephoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures. *arXiv preprint arXiv:1908.09979*, 2019.

[63] Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1803–1811, 2019.

[64] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018.

Variational Nested Dropout (Appendix)

Yufei Cui[1,2], Yu Mao[1], Ziquan Liu[1], Qiao Li[3], Antoni B. Chan[1], Xue Liu[2], 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 Computer Science, McGill University
[3]School of Informatics, Xiamen 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 $\text{cumsum}_i'(\mathbf{c}) = \sum_{j=0}^{i-1} c_j$. $c_0 := 1$. 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 [18], [41] which has the following form.

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

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

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

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 [40], [25], 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 $\text{cumsum}'(\mathbf{c}^*) = [\underbrace{0, \ldots, 0}_{b} \underbrace{1, \ldots, 1}_{K-b}]$. Thus, $\mathbf{z}^* = \mathbf{e} - \text{cumsum}'(\mathbf{c}^*) = [\underbrace{1, \ldots, 1}_{b} \underbrace{0, \ldots, 0}_{K-b}]$. 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 $g_1 \ldots g_K$ 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 $\beta_j = (1 - \pi_{j+1}) \prod_{k=1}^{j}, \forall j \in [1, \ldots, K]$.

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 $\mathbf{w}_j = [w_{1j}, \ldots, w_{Dj}]$ 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 $\kappa_{ij}^k(\boldsymbol{\theta})$ as the KL of w_{ij} for component $k \in \{0, 1\}$. The term Φ_2 can then be re-organized as

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

There are totally $D^2 d$ terms, which potentially causes a large computation cost in every epoch. Consider the matrices $\boldsymbol{\kappa}_{\boldsymbol{\theta}}^0 = [\kappa_{ij}^0(\boldsymbol{\theta})]_{ij} \in \mathbb{R}^{d \times D}$ and $\boldsymbol{\kappa}_{\boldsymbol{\theta}}^1 = [\kappa_{ij}^1(\boldsymbol{\theta})]_{ij} \in \mathbb{R}^{d \times D}$, 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 [39], 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,

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

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

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) [39] 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 [46] 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

$$\mathrm{KL}[q(\mathbf{h},\mathbf{z}|\mathbf{x})||p(\mathbf{h},\mathbf{z})]$$

$$= \mathbb{E}_{q(\mathbf{z})}[\log \frac{q(\mathbf{z})}{p(\mathbf{z})}] + \mathbb{E}_{q(\mathbf{z})} \underbrace{\mathbb{E}_{q(\mathbf{h}|\mathbf{z})}[\log \frac{q(\mathbf{h}|\mathbf{z})}{p(\mathbf{h}|\mathbf{z})}]}_{\boldsymbol{\kappa} = [\kappa_i]_i}$$

$$= \underbrace{\sum_i \beta_i \log \frac{\beta_i}{(1-\pi_{i+1})\prod_{k=1}^{i}\pi_k}}_{\Phi_1} + \underbrace{\sum_i \beta_i \kappa_i}_{\Phi_2} \tag{A18}$$

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

$$\mathrm{KL} = \Phi_1 + \Phi_2$$

$$= \sum_i \beta_i \log \frac{\beta_i}{e^{-\kappa_i}(1-\pi_{i+1})\prod_{k=1}^{i}\pi_k} \tag{A20}$$

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

$$\mathrm{KL} = \sum_i \beta_i \log \frac{\beta_i}{e^{-\kappa_i}(1-\pi_{i+1})\prod_{k=1}^{i}\pi_k} \frac{C}{C}$$

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

$$= \sum_i \beta_i \log \frac{\beta_i}{e^{-\kappa_i}(1-\pi_{i+1})\prod_{k=1}^{i}\pi_k/C} - \log C$$

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 $\mathrm{KL} = -\log C$. In this case, if $\kappa_i = 0, \forall i$, then $C = 1$ and $\mathrm{KL} = 0$. If $\kappa_i > 0$, then $C < 1$ and $\mathrm{KL} > 0$. 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 $\mathrm{KL} = 0$. 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 [30], [42], the local reparameterization trick can be applied, due to the linearity of the convolutional layer.

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

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 [55], we re-scale the weights according to the probability to keep a node.

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

where $\mathcal{D}_{\mathrm{tr}}$ and $\mathcal{D}_{\mathrm{te}}$ 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.

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

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.

$$\beta_1 = (1 - \mu_2)\mu_1 = 1 - \mu_2 \tag{A26}$$
$$\beta_2 = (1 - \mu_3)\mu_2\mu_1$$
$$\cdots$$

Solving each equation sequentially, we obtain

$$p(z_1 = 1) = 1,$$
$$p(z_2 = 1) = 1 - \beta_1,$$
$$p(z_3 = 1) = 1 - \beta_1 - \beta_2,$$
$$\cdots \tag{A27}$$

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 $\mu_k = \mathrm{sigmoid}(\bar{\mu}_k)$.

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 [24].

---

2. We use this parameterization in our implementation, while we use β_j in most of our derivation for simplicity in writing.
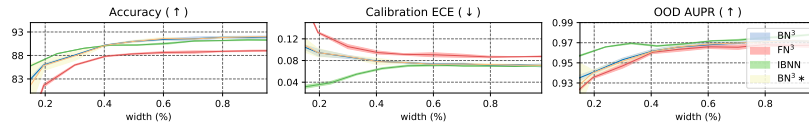
Fig. 8: Performance of VGG11 on Cifar10 with less data for collecting BN statistics.

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) [44], as it was shown to make the Bayesian neural network converge faster [27]. 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 [21]. 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.
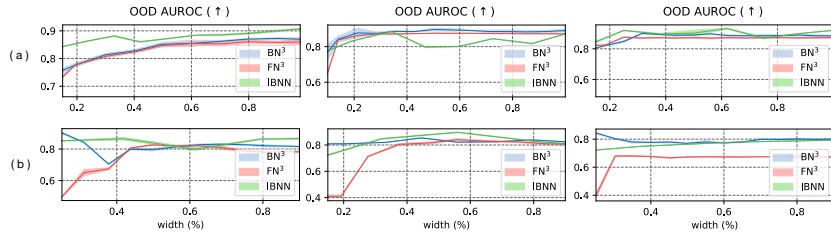4. http://ufldl.stanford.edu/housenumbers/

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.
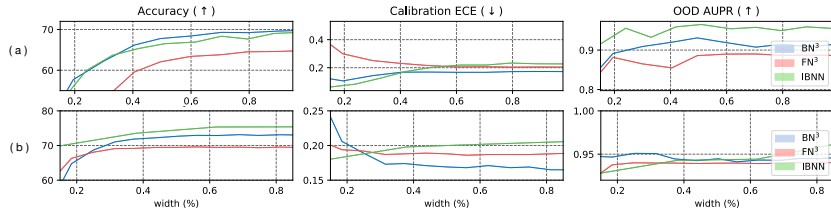

Fig. 10: Results on Cifar100 for (a) VGG11, (b) ResNeXt-Cifar.
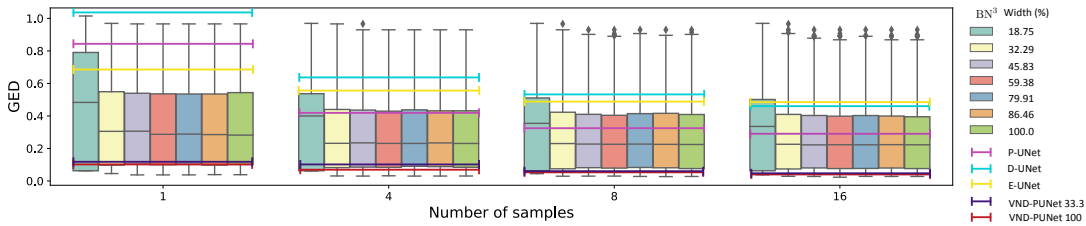
C.5 Full visualization of semantic segmentation results

Fig. 11: Evaluation on semantic segmentation using generalized energy distance (↓) with different numbers of posterior samples. Each box-plot shows the GED of all data of BN^3 for one network width (%). The black horizontal line in the box plot represents the mean. The bold horizontal lines represent the averaged results for comparison methods (using full width) and VND-PUNet (using 33% or 100% width).