Variational Nested Dropout

Cui, Yufei; Mao, Yu; Liu, Ziquan; Li, Qiao; Chan, Antoni B.; Liu, Xue; Kuo, Tei-Wei; Xue, Chun Jason

Published in:
IEEE Transactions on Pattern Analysis and Machine Intelligence

Published: 01/08/2023

Document Version:
Post-print, also known as Accepted Author Manuscript, Peer-reviewed or Author Final version

Publication record in CityU Scholars:
Go to record

Published version (DOI):
10.1109/TPAMI.2023.3241945

Publication details:
https://doi.org/10.1109/TPAMI.2023.3241945

Citing this paper
Please note that where the full-text provided on CityU Scholars is the Post-print version (also known as Accepted Author Manuscript, Peer-reviewed or Author Final version), it may differ from the Final Published version. When citing, ensure that you check and use the publisher's definitive version for pagination and other details.

General rights
Copyright for the publications made accessible via the CityU Scholars portal is retained by the author(s) and/or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights. Users may not further distribute the material or use it for any profit-making activity or commercial gain.

Publisher permission
Permission for previously published items are in accordance with publisher's copyright policies sourced from the SHERPA RoMEO database. Links to full text versions (either Published or Post-print) are only available if corresponding publishers allow open access.

Take down policy
Contact lbscholars@cityu.edu.hk if you believe that this document breaches copyright and provide us with details. We will remove access to the work immediately and investigate your claim.

Download date: 27/07/2024
Variational Nested Dropout

Yufei Cui\textsuperscript{1,2}, Yu Mao\textsuperscript{1}, Ziquan Liu\textsuperscript{1}, Qiao Li\textsuperscript{2}, Antoni B. Chan\textsuperscript{1}, Xue Liu\textsuperscript{2}, Tei-Wei Kuo\textsuperscript{4}, Chun Jason Xue\textsuperscript{1}

\textsuperscript{1}Department of Computer Science, City University of Hong Kong
\textsuperscript{2}School of Computer Science, McGill University
\textsuperscript{3}School of Informatics, Xiamen University
\textsuperscript{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 \cite{11}, \cite{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 \cite{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 \cite{28}, \cite{64}, \cite{63}, \cite{7}, \cite{11}, \cite{61}, to reduce the cost in designing \cite{57}, training \cite{29}, compressing \cite{20} and deploying \cite{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 \cite{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 \cite{50} and auto-encoder \cite{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 \cite{48}, \cite{3}.

An operator for neural networks, nested dropout (also called ordered dropout), was proposed for both ordering in model architectures \cite{11}, \cite{14} and ordering representations \cite{48}, \cite{3}.

It was first developed to order the latent feature representation for the encoder-decoder models \cite{48}, \cite{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 \cite{11}, \cite{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 U-Net with VND also outperform the vanilla probabilistic U-Net [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 $\ell_0$-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 U-Net (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\textsuperscript{2} and supplemental\textsuperscript{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 $p_\ell(\cdot)$ 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 $I \sim p_\ell(\cdot)$ 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.

---

\textsuperscript{2} https://github.com/ralphc1212/variational_nested_dropout
\textsuperscript{3} http://visual.cs.cityu.edu.hk/static/pubs/journal/pami-vnd-supp.pdf
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 [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 \( p(\cdot) \).

### 2.2 Bayesian Ordering Unit

The conventional nested dropout uses a Geometric distribution to sample the tail index \( I \), \( p(I = i) = (1 - \pi)^i \pi \), 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, \( \pi \) 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 indices 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 [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 \( p(\cdot) \).

### 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 \( z \). 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. \( z \) has a Downhill distribution \( z \sim \text{Downhill}(\beta, \tau) \), if its density is:

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

where \( \beta = [\beta_1, \ldots, \beta_K] \) are the probabilities for each dimension.

Two important properties of Downhill distributions are:

- **Property 1.** If \( c \sim \text{Gumbel}_\text{softmax}(\tau, \beta, \epsilon_z)^4 \), then \( z_i = 1 - \text{csum}\text{softmax}(\log(\beta)) \), where \( \epsilon_z \) is a \( K \)-dimensional vector of ones, and \( \text{csum}\text{softmax}(\cdot) = \sum_{j=0}^{i-1} \epsilon_j \). Let \( \epsilon_z \) be a standard uniform variable.

- **Property 2.** For Gumbel-softmax sampling, we first draw \( g_1, \ldots, g_K \) from Gumbel(0, 1), then calculate \( \epsilon_z = \text{softmax}(\log(\beta)) \). The samples of Gumbel(0, 1) can be obtained by first drawing \( \epsilon_z \sim \text{Uniform}(0, 1) \) then computing \( g = -\log(-\log(\epsilon_z)) \).
Dimension of techniques is variational inference, with which the posterior
\( u \) parameters 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 \( \tau \) controls the sharpness of the downhills or the smoothness of the step at the tail index. When \( \tau \) 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 \( \tau \) approaches 0, the shape of the sample becomes a cliff which is similar to the prior \( p(z) \) on ordered masks, where the less important nodes are dropped (i.e., multiplied by 0). Property 1 further implies the gradient of an ordered distribution can be estimated with low variance, for a cost function \( \mathcal{L}(z) \). Because the samples of \( z \) are replaced by a differentiable function \( t(\beta, \epsilon_z) \), then

\[
\frac{\partial}{\partial \beta} q_{\phi}(z) \mathcal{L}(z) = E_{z \sim q_{\phi}(z)} \left[ \frac{\partial}{\partial \beta} \mathcal{L}(z) \right],
\]

where \( t(\cdot) \) represents the whole transformation process in Prop. 1.

Recall that our objective is to approximate the chain of Bernoulli variables \( p(z) \) with \( q_{\phi}(z) \). 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 \( p(z) \), 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 (BN3) based on the fundamental units proposed in Section 2.

### 3.1 Bayesian Inference and SGVB

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

For neural networks, computing the posterior distribution using the Bayes’ rule requires computing intractable integrals over \( \theta \). Thus, approximation techniques are required. One family of techniques is variational inference, with which the posterior

\[ p(\theta|D) \]

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

\[
\mathcal{L}_{\phi} = \mathcal{L}(\theta) - KL[q_{\phi}(\theta)||p(\theta|D)],
\]

(4)

where the expected data log-likelihood is

\[
\mathcal{L}(\theta) = \sum_{i=1}^{N} E_{z \sim q_{\phi}(\theta)} \log p(y_i|x_i, \theta),
\]

(5)

The integration \( \mathcal{L}(\theta) \) is tractable for neural networks. An efficient method for gradient-based optimization of the variational bound is stochastic gradient variational Bayes (SGVB) [31], [30].

\[
\mathcal{L}_{\theta|\beta}(\theta) \approx L_{SGVB}(\theta|\beta) = \frac{1}{M} \sum_{m=1}^{M} \log p(y_i|x_i, \theta = t(\epsilon, \phi), \epsilon \sim \mathcal{N}(0, 1)),
\]

where \( \{ (x_i, y_i) \}_{i=1}^{M} \) is a minibatch of data with \( M \) random instances \( (x_i, y_i) \sim D \) and \( \epsilon \sim \mathcal{N}(0, 1) \).

### 3.2 Bayesian Nested Neural Network

In our model, the r.v. \( u = (W, z) \) consists of two parts: weight matrix \( W \) and ordering units \( z \). 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 \( \phi = (\theta, \beta) \), where \( \theta \) and \( \beta \) are the variational parameters for the weights and ordering units respectively. We then have the following optimization objective,

\[
L_{SGVB} \approx L_{SGVB}(\theta|\beta) = KL[q_{\phi}(W, z)||p(W, z)],
\]

(6)

\[
L_{D}(\phi) \approx L_{D}(\phi) = \sum_{i=1}^{N} \log p(y_i|x_i, \theta),
\]

where \( \epsilon \) and \( \epsilon \) are the random noise, and \( t_{w}(\cdot) \) and \( t_{z}(\cdot) \) are the differentiable functions that transform the noise 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 \( \mathbf{F} \in \mathbb{R}^{d \times D} \), \( d \) and \( D \) are the input and output size, and \( M \) is the batch size. The elements are indexed as \( h_{mij} \), \( f_{mij} \), 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 \( z \in \mathbb{R}^{D} \) 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 \( z \) is flexible, and can be applied to \( W \) row-wise or element-wise as well.

The prior for \( W \) assumes each weight is independent,

\[
p(W) = \prod_{ij} p(w_{ij}), \quad \text{where} \quad i \in \{1, \ldots, d\} \text{ and } j \in \{1, \ldots, D\}.
\]

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:

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

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

where \( \mu_{0ij}, \sigma_{0ij}^2 \) and \( \mu_{1ij}, \sigma_{1ij}^2 \) are the means and standard deviations for the two components. We fix \( \mu_{0ij} = 0 \) and \( \sigma_{0ij}^2 \) to
Appendix A. derivations and proofs for

Note that larger sub-networks have greater penalization. The
\( \ell \) and-slab priors, our KL term in (9) has an interpretation as

\( \text{KL} \text{ function element-wise. The term} \pi \)

To calculate the expected data log-likelihood, our Downhill

distribution allows efficient sampling and differentiable transfor-
mation for the ordering units (Section 2.3). The reparameteriza-
tion 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 \( q_{\theta} (W) \) takes the same form as the prior, while

\( q_{\alpha} (z) \) takes the Downhill distribution \( z \sim \text{Downhill}(\beta, \tau) \). We
consider the case that \( \tau \rightarrow 0 \) for simplicity, while \( \tau \) can be
adjusted in the training process as annealing. For this layer, the

KL divergence in (7) is

\[ \text{KL}[q_{\theta,\beta}(W, z) || p(W, z)] \]

\[ = E_{q_{\alpha}(z)} \log \frac{q_{\alpha}(z)}{p(z)} + E_{q_{\alpha}(z)}E_{q_{\theta}(W|z)} \log \frac{q_{\theta}(W|z)}{p(W|z)} \]

Term \( \Phi_1 \) of (9) is

\[ \Phi_1 = \sum_{\pi \in \mathcal{V}} q_{\beta}(z) \log \frac{q_{\beta}(z)}{p(z)} = \sum_{j=1}^{D} \text{KL}[q_{\beta}(v_j) || p(v_j)], \]

where \( \mathcal{V} = \{v_1, \ldots, v_D\} \) is the set of ordered masks. The number of
components in the z space is reduced from \( D^2 \) to \( D \), because there are only \( D \) possible ordered masks. By definition, the probabilities are

\[ q_{\beta}(v_j) = \beta_j, \quad p(v_j) = (1 - \pi_{j+1}) \prod_{k=1}^{j} \pi_k, \]

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

\[ \Phi_1 = \beta^T \text{log}(\beta) - \text{log}(1 - \pi) J_{\beta} \pi) \]

where \( \log(\cdot) \) is an element-wise log function. \( \pi = [\pi_1, \ldots, \pi_D] \).
\( \pi = [\pi_1, \ldots, \pi_{D+1}] \), \( \mathbf{e} \) is a vector of 1s and \( J_{\beta} \) is a lower

triangular matrix with each element being 1.

We define \( \kappa_{ij}^k (\theta) \) as the KL of \( w_{ij} \) for component \( k \in \{0, 1\} \).
Consider the matrices \( \kappa_{ij}^0 = [\kappa_{ij}^0]_{ij} \in \mathbb{R}^{D \times D} \) and \( \kappa_{ij}^{1} = [\kappa_{ij}^{1}]_{ij} \in \mathbb{R}^{D \times D} \), which are easily computed by applying the KL function element-wise. The term \( \Phi_2 \) is then expressed as

\[ \Phi_2 = \mathbf{e}^T \kappa_{ij}^0 (\mathbf{J} - J_{\beta})^T \beta + \mathbf{e}^T \kappa_{ij}^{1} J_{\beta}, \]

where \( J \) is a matrix of 1s.

Ordered \( \ell_0 \)-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 \( \ell_0 \) regularization over the sub-networks. The corresponding reduced objective for deterministic networks is

\[ \min_{\theta, \beta} E_{q_{\theta} (\beta)} L_D (\theta, \beta) + \lambda \sum_{j} j \beta_j \]

Note that larger sub-networks have greater penalization. The derivations and proofs for \( p(v_j) \), \( \Phi_2 \) and regularization are in Appendix A.

3.4 Implementation

For efficient sampling of the weight distributions, we put multi-
plicative Gaussian noise \( q_{\theta} (W) \) on the weight \( w_{ij} \), similar to [30], [42], [37]. We take \( w_{ij} \) as an example.

\[ w_{ij} = \theta_{ij} q_{\theta} (1 + \sqrt{\sigma_{ij} \epsilon_w}), \epsilon_w \sim \mathcal{N}(0, 1), \]

\[ w_{ij} \sim \mathcal{N}(w_{ij} | \theta_{ij}, \alpha_{ij} \sigma_{ij}^2) \]

We also assume a log-uniform prior [30], [42], [37], i.e.,

\[ p(\log | w_{ij} |) \] const

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 \( \theta_{ij} \) is eliminated from the
computation of KL.

As the second term in (16) cannot be computed analyti-
cally and should be estimated by sampling, Kingma et al [30] propose
to sample first and design a function to approximate it. But their approxima-
tion of \( \kappa_{ij}^1 (\theta) \) does not encourage \( \alpha_{ij} > 1 \) as the opti-
mization would be difficult. They truncate \( \alpha_{ij} \leq 1 \) corre-
sponding to a small variance, which is not flexible. Molchanov et al [42] use
a different parameterization that pushes \( \alpha_{ij} \rightarrow \alpha_{ij} \infty \), as illustrated in Figure 7. This means that \( w_{ij} \) 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 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 \( -\kappa_{ij}^1 (\theta) \)

\[ a_1 e^{-a_1 (\alpha_1 + a_3 \log \alpha_{ij})} - 0.5 \log (1 + \alpha_{ij}^{-1}) + C, \]

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 \( \epsilon_w \) 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 \( \log \alpha \in \)
Algorithm 1: Pseudocode for training BN³.

Input: Parameters \( \{\beta^{(l)}(i), \alpha^{(l)}(i), \beta^{(l)}(i)\} \), data \( \{X, Y\} \), layer input \( H^{(0)} = X_{\text{batch}} \), total KL \( KL = 0 \).

1: while the network is not converged do
2: for \( l = 1 \) to \( L \) do
3: Sample \( h^{(l)}(i) \) from \( \alpha^{(l)}(i), \beta^{(l)}(i) \) for dense layers or \( (A22) \) for convolution layers.
4: Sample \( \mu^{(l)} = q_{\beta^{(l)}(i)}(x) \) and \( F^{(l)} = [f_{m_j(i)}]_{m_j} = b^{(l)}(i) \).
5: Compute KL using (18), \( KL \leftarrow KL + KL \).
6: \( H^{(l+1)} = F^{(l)} \).
7: end for
8: Compute loss as \( L_{D}(Y_{\text{batch}}, F^{(l)}) \), \( L = L_{D} - KL \).
9: Compute \( \frac{\partial L}{\partial \theta}, \frac{\partial L}{\partial \pi}, \frac{\partial L}{\partial \nu}, \frac{\partial L}{\partial \beta} \), update the network.
10: end while

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

The pseudocode 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 \( e^{T} \kappa_0 J_{T}^T \beta \). Therefore, the KL divergence (9) is simplified to

\[
KL[q_{\beta, \theta}(W, z) || p(W, z)] = \Phi_1 + \Phi_2
\]

\[
= \beta^T \log(\beta) - \log((e - \pi) J_{T}^T \pi) + e^T \kappa_0 J_{T}^T \beta,
\]

where \( \kappa_0 \) is calculated by applying (17) element-wise.

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

\[ f_{m_{j}} = b_{m_{j}} z_{j}, \quad b_{m_{j}} \sim \mathcal{N}(\gamma_{m_{j}}, \delta_{m_{j}}), \]

\[ \gamma_{m_{j}} = \sum_{i=1}^{d} h_{m_{i}} \theta_{i_{j}}, \quad \delta_{m_{j}} = \sum_{i=1}^{d} h_{m_{i}}^{2} \alpha_{i_{j}} \beta_{i_{j}}^2. \]

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 BN³ 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.

For pursuing a lighter neural network and a more practical inference speed, we can distill the Bayesian knowledge from BN³ to its deterministic counterpart. This is motivated by an analysis of correlation of outputs of BN³ with different widths. As shown in Figure 5, we conduct correlation analysis on two BN³ with VGG11 and ResNeXt as backbones. For each network and a particular dataset, we obtain the uncertainty of the output under different widths.

\[
U = |U_{\theta, \nu}^x|_{0.1, 0.2, \ldots, 1.0},
\]

\[
U_{\theta, \nu} = [U_{\theta, \nu}^x]_{i=1}^N = [U_{\|x_i, \theta, \nu\|}]_{i=1}^N,
\]

where the \( |x_i, \theta, \nu| = 0.1 \) means 10% of BN³ parameters are used. We use \( |x_i, \theta, \nu| \) to indicate the number of non-zero entries in \( x \). The conversion from \( v \) to \( z \) is illustrated in Section 3.3. A common choice for \( U \) is the entropy. For calculating the correlation between the uncertainties \([U_{\theta, \nu}^x]_{i=1}^N\) for any pair \([v_i, v_j]\), 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 BN³, then distill the original BN³ uncertainty to the pruned BN³ 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 (SN³).

The first step is to prune the less useful weights according to the correlation analysis, to generate the base of SN³. The mean of the nodes and the VND parameters from BN³ 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 adopt 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$^3$:

$$\text{MMD}_K(q, \text{Dir}) = \sup_{\Psi \in \mathcal{H}_K, \|\Psi\|_{\mathcal{H}_K}} \mathbb{E}_q(\Psi(BN^3(x))) - \mathbb{E}_{s \sim \text{Dir}(SN^3(x))}(\Psi(s))$$

(20)

Note that $\mathcal{H}_K$ is a reproducing kernel Hilbert space defined by a positive-definite kernel $K$. 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 $q(\tilde{z})(q(\tilde{v}))$, where $\tilde{z}$ and $\tilde{v}$ are the VND random variable for the pruned SN$^3$. $\tilde{v}$ is no longer sampled from $q(\tilde{v})$ but determined by $\tilde{v}$

$$|\tilde{v}| = r|\tilde{v}|$$

(21)

where $|\tilde{v}|$ is the number of dimensions in vector $\tilde{v}$, 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 U-Net.

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 BernChain($z$, $\pi$), introduced in Section 2.2, with random variable $z = [z_1, \ldots, z_D]$ for the mask and $\pi = [\pi_1, \ldots, \pi_D]$ for the parameters. We let a uni-variate Gaussian variable conditioned on each node of the Bernoulli chain,

$$p(h_i|z_i) = z_i \mathcal{N}(h_i|0, \sigma^2_i) + (1 - z_i) \mathcal{N}(h_i|0, \sigma^2_\infty)$$

(22)

where $z_i$ is sampled from the above Bernoulli chain. $h_i$ is the $i$-th latent representation for $x$. $\sigma^2_\infty$ 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 $z$ (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

$$z|x \sim \text{Downhill}(|f^x(x), \tau)$$

$$h_i|z_i, x \sim \mathcal{N}(h_i|f_i^x(x), f_i^x(x)) + (1 - z_i) \mathcal{N}(h_i|0, \sigma^2_\infty)$$

(23)

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

As the Downhill distribution is fully re-parameterizable, we let $z$ be conditioned on $x$ 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 $x$. The evidence lower bound (ELBO) is written as:

$$\log p(x) \geq \mathbb{E}_{h,z}[\log p(x|h, z)] - KL(q(h, z|x)||p(h, z|x))$$

(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 $h$ 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.
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 $h_i$ are drawn from the latent space with fixed Downhill samples $z$. 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 comparison, 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 probabilistic 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 $x$. The posterior is conditioned on both $x$ and $y$ 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,

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

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

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

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

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

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

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

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

(29)

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

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

(30)

where $\kappa_i(x, y)$ 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), $z^* \sim p(z|x^*), h^* \sim p(h|z^*, x^*)$, and then fed to the deterministic neural network to obtain the results $p(y^*|f_{\text{base}}(x^*, h^*))$.

5 Related work

In this section, we reviewed deep nets with $\ell_0$-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.

$\ell_0$-regularization. The Bernoulli-Gaussian linear model with independent Bernoulli variables is shown to be equivalent to $\ell_0$ regularization [43]. Recent works [39], [62] investigate $\ell_0$ norm for regularizing DNNs. [39] presents a general formulation of a $\ell_0$-regularized learning objective for a single deterministic neural network,

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

(31)

where the variable $\tilde{z}$ 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 $\ell_0$-regularization over the weights is a special case of an ELBO over parameters with spike-and-slab priors. These works present the uniform $\ell_0$-regularization as the coefficient $\lambda$ 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
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 $\text{BN}^3$ and $\text{SN}^3$, image generation for VND-AE, and probabilistic semantic segmentation for VND-PUNet.

6.1 Image Classification with $\text{BN}^3$

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 $\text{BN}^3$), 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 $\text{BN}^3$ with the fully nested neural network ($\text{FN}^3$) [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 $\text{BN}^3$, which uses nested sub-networks. In addition, the deterministic $\text{BN}^3$ (DT-$\text{BN}^3$) 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 $\text{FN}^3$ ($\text{FN}^3$ - TS).

During testing time, we generate fixed width masks for $\text{BN}^3$ and $\text{FN}^3$ 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 $\text{BN}^3$ 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, $\text{BN}^3$ and DT-$\text{BN}^3$ are expected to have better performance than $\text{FN}^3$ and $\text{FN}^3$ - 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, $\text{BN}^3$ 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 concluded 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

<table>
<thead>
<tr>
<th>Removed weights (%)</th>
<th>Acc. (↑)</th>
<th>ECE (↓)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SNR</td>
<td>BN$^3$</td>
</tr>
<tr>
<td>0</td>
<td>90.4</td>
<td>91.8</td>
</tr>
<tr>
<td>20</td>
<td>90.3</td>
<td>91.9</td>
</tr>
<tr>
<td>40</td>
<td>90.4</td>
<td>91.5</td>
</tr>
<tr>
<td>60</td>
<td>89.4</td>
<td>91.1</td>
</tr>
<tr>
<td>80</td>
<td>29.6</td>
<td>85.6</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th></th>
<th>Acc.↑</th>
<th>ECE.↑</th>
<th>OOD.↑</th>
<th>Weight usage (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>smallest BN^3</td>
<td>83.5</td>
<td>0.10</td>
<td>0.94</td>
<td>6.2</td>
</tr>
<tr>
<td>SVB [42]</td>
<td>82.9</td>
<td>0.17</td>
<td>0.94</td>
<td>5.7</td>
</tr>
<tr>
<td>BN^3-MC</td>
<td>89.4</td>
<td>0.08</td>
<td>0.97</td>
<td>46 mean ± 27 std</td>
</tr>
<tr>
<td>BN^3</td>
<td>92.6</td>
<td>0.07</td>
<td>0.97</td>
<td>100</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th></th>
<th>Train</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>BN^3 / DT-BN^3</td>
<td>14.9 h</td>
<td>1.78 s</td>
</tr>
<tr>
<td>IBNN</td>
<td>7.5 X N h</td>
<td>1.68 s</td>
</tr>
<tr>
<td>FN^3</td>
<td>5.6 h</td>
<td>1.82 s</td>
</tr>
</tbody>
</table>

### 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.

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

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], 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 Reconstruction 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
distribution of the original image set. When latent dimensions are the same, VND-AE’s FID score outperforms the other baselines.

**Inception score (IS).** Higher IS indicates better diversity of the generated images. A comparison of VNE-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.

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 applications: 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


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

<table>
<thead>
<tr>
<th>%9-BN³</th>
<th>46%-BN³</th>
<th>80%-BN³</th>
<th>Full-BN³</th>
<th>P-UNet</th>
<th>D-UNet</th>
<th>E-UNet</th>
<th>33%-VND-PUNet</th>
<th>Full-VND-PUNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 sample</td>
<td>0.51±0.28</td>
<td>0.31±0.25</td>
<td>0.29±0.25</td>
<td>0.27±0.27</td>
<td>0.60</td>
<td>1.31</td>
<td>0.83</td>
<td>0.14</td>
</tr>
<tr>
<td>4 sample</td>
<td>0.39±0.12</td>
<td>0.24±0.15</td>
<td>0.23±0.15</td>
<td>0.23±0.14</td>
<td>0.40</td>
<td>0.62</td>
<td>0.55</td>
<td>0.12</td>
</tr>
<tr>
<td>8 sample</td>
<td>0.34±0.15</td>
<td>0.22±0.17</td>
<td>0.21±0.16</td>
<td>0.22±0.17</td>
<td>0.32</td>
<td>0.53</td>
<td>0.49</td>
<td>0.07</td>
</tr>
<tr>
<td>16 sample</td>
<td>0.36±0.14</td>
<td>0.21±0.13</td>
<td>0.21±0.13</td>
<td>0.21±0.12</td>
<td>0.28</td>
<td>0.46</td>
<td>0.50</td>
<td>0.05</td>
</tr>
</tbody>
</table>
Variational Nested Dropout (Appendix)

Yufei Cui\textsuperscript{1,2}, Yu Mao\textsuperscript{1}, Ziquan Liu\textsuperscript{1}, Qiao Li\textsuperscript{3}, Antoni B. Chan\textsuperscript{1}, Xue Liu\textsuperscript{2}, Tei-Wei Kuo\textsuperscript{4}, Chun Jason Xue\textsuperscript{1}
\begin{flushright}
yufecui92@gmail.com, yfcui@bingli.com
\end{flushright}

\textsuperscript{1}Department of Computer Science, City University of Hong Kong
\textsuperscript{2}School of Computer Science, McGill University
\textsuperscript{3}School of Informatics, Xiamen University
\textsuperscript{4}Department of Computer Science and Information Engineering, National Taiwan University

A DERIVATION AND PROOFS

A.1 Derivation of Properties

Property 1. If $c \sim \text{Gumbel}_\text{softmax}(\tau, \beta, \epsilon_z)$\textsuperscript{1}, then $z_i = 1 - \text{csum}_i(c)$, where $c$ is a $K$-dimensional vector of ones, and $\text{csum}_i(c) = \sum_{j=0}^{i-1} c_j$, $c_0 := 1$. $\epsilon_z$ is a standard uniform variable.

We show that using the sampling process in Property 1 recovers the Downhill random variable. We assume $c$ follows a Gumbel softmax distribution \cite{18}, \cite{41} which has the following form.

\begin{equation}
p(c_1, \ldots, c_K) = \Gamma(K)\tau^{K-1} \prod_{i=1}^{K} (\pi_i/c^i)^{i-K} \prod_{i=1}^{K} (\pi_i/c^{i+1})
\end{equation}

We apply the transformation $T_i(\cdot) = c - \text{csum}_i(\cdot)$ to the variable $c$. $\mathbf{z} = T(c) = c - \text{csum}_i(c)$

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

\begin{equation}
p(\mathbf{z}) = p(T^{-1}(\mathbf{z})) | \det(\frac{\partial T^{-1}(\mathbf{z})}{\partial \mathbf{z}}) |
\end{equation}

\begin{equation}
p(z_{1:K}) = p(T^{-1}(z_{1:K})) | \det(\frac{\partial T^{-1}(z_{1:K})}{\partial z_{1:K}}) |
\end{equation}

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

\begin{equation}
\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 & \ddots & \vdots & \vdots \\
0 & 0 & \ldots & 1 & -1
\end{bmatrix}
\end{equation}

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

\begin{equation}
p(z_{1:K}) = p(T_{1:K}^{-1}(z_{1:K})) = \Gamma(K)\tau^{K-1} \prod_{i=1}^{K} (\pi_i/(z_{i-1} - z_i)^{i-K}) \prod_{i=1}^{K} \left(\frac{\pi_i}{(z_{i-1} - z_i)^{i+1}}\right).
\end{equation}

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 \cite{40}, \cite{25}, when $\tau \to 0$, the Gumbel softmax transformation corresponds to an argmax operation that generates a one-hot vector:

\begin{equation}
c = \text{one}_\text{hot}(\text{argmax}_i(g_i + \log \beta_i)),
\end{equation}

where the relative order is preserved.

Say a sample $c^* \sim \text{Gumbel}_\text{softmax}(\beta, \tau \to 0)$, with $b$-th entry being one and the remaining entries being 0. The defined transformation generates $\text{csum}_i(c^*) = [0, \ldots, 0, 1, 0, \ldots, 0]$. Thus, $z^* = c - \text{csum}_i(c^*) = [1, \ldots, 1, 0, \ldots, 0]$. It is easy to see the transformation $t'(\cdot)$ is surjective function where $t' : \{\text{one}_\text{hot}(i)\}_{i=1}^{K} \to \mathcal{V}, t'(c) = c - \text{csum}_i(c)$. $\mathcal{V}$ 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,

\begin{equation}
\text{KL}[q(\mathbf{z})||p(\mathbf{z})] = \sum_{j=1}^{K} q(\mathbf{v}_j) \log \left(\frac{q(\mathbf{v}_j)}{p(\mathbf{v}_j)}\right)
\end{equation}

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

© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
where $q(v_j) = \beta_j, p(v_j) = (1 - \pi_{j+1}) \prod_{k=1}^j \pi_k$ (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{align*}
p(z_1 = 1) &= \pi_1, \quad p(z_1 = 0) = 1 - \pi_1, \\
p(z_1 = 1|z_{i-1} = 1) &= \pi_i, \quad p(z_1 = 0|z_{i-1} = 1) = 1 - \pi_i, \\
p(z_1 = 1|z_{i-1} = 0) &= 0, \quad p(z_1 = 0|z_{i-1} = 0) = 1,
\end{align*}

(A8)

It is observed, there is a chance that $z_i$ is the index of first zero. For convenience, we define $w_j = [q_j, \ldots, q_K]$, $\pi_{j+1}, \ldots, \pi_K$ as the KL of $\theta_j \sim \mathcal{N}(w_j|z_j^k)$. Define $\kappa_{ij}^k(\theta)$ as the KL of $\theta_j$ for component $k \in \{0, 1\}$. The term $\Phi_2$ of (9) is

\begin{align*}
\Phi_2 &= \sum_{\theta \in \Theta} \sum_{i} q_j(z) \sum_{j} \int_{w_j} q_j(w_j|z_j^k) \log \frac{q_j(w_j|z_j^k)}{p_j(w_j|z_j^k)} dw_j \\
&= \sum_{\theta \in \Theta} \sum_{i} q_j(z) \sum_{j} \int_{w_j} q_j(w_j|z_j^k) \log \frac{q_j(w_j|z_j^k)}{p_j(w_j|z_j^k)} dw_j \\
&= \sum_{\theta \in \Theta} \sum_{i} q_j(z) \sum_{i,j} KL[q_j(w_j|z_j^k)||p_j(w_j|z_j^k)].
\end{align*}

(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 $\kappa_{ij}^k(\theta)$ as the KL of $\theta_j$ for component $k \in \{0, 1\}$. The term $\Phi_2$ can then be re-organized as

\begin{align*}
\Phi_2 &= q(z = v_1)(\sum_{i} K_{11}^i(\theta) + \sum_{j=2}^D \sum_{i} \kappa_{ij}^0(\theta)) \\
&+ q(z = v_2)(\sum_{j=1}^2 \sum_{i} \kappa_{ij}^1(\theta) + \sum_{j=3}^D \sum_{i} \kappa_{ij}^0(\theta)) \\
&+ \ldots
\end{align*}

(A11)

There are totally $D^2d$ terms, which potentially causes a large computation cost in every epoch. Consider the matrices $\kappa_{ij}^0(\theta) \in \mathbb{R}^{d \times D}$ and $\kappa_{ij}^1(\theta) \in \mathbb{R}^{d \times D}$, which are easily computed by applying the KL function element-wise. The term $\Phi_2$ is then expressed as

\begin{align*}
\Phi_2 &= e^T \kappa_0(\theta - J_L)^T \beta + e^T \kappa_1^j J_L^T \beta,
\end{align*}

(A12)

where $e$ is a vector of 1s, $J$ is a matrix of 1s and $J_L$ 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.3 Posterior approximation - $\Phi_2$

Define $w_j = [w_{ij}, \ldots, w_{ij}]$ as the $j$-th column of $W$, and $q_\theta(w_j|z_j = k) = q_\theta(w_j|z_j^k)$ where $k \in \{0, 1\}$. 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.4 $\ell$-0 regularization

We consider the case when the prior over each weight is a spike-and-slab 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 $\kappa$ remain unchanged as it makes nothing but mean-field assumption on the weight prior. With $\Phi_1$ and $\Phi_2$, the objective (7) can be re-organized as

\begin{align*}
L_{SGVB}^{\theta, \beta} &= L_{SGVB}^{\theta, \beta} - \sum_{j=1}^D KL[\theta_j|\theta_{\text{prior}}] - \sum_{j=1}^D \kappa_0(\theta - J_L)^T \beta - \sum_{j=1}^D \kappa_1^j J_L^T \beta \\
&= L_{SGVB}^{\theta, \beta} - \sum_{j=1}^D KL[\theta_j|\theta_{\text{prior}}] - \sum_{j=1}^D \kappa_0(\theta - J_L)^T \beta - \sum_{j=1}^D \kappa_1^j J_L^T \beta
\end{align*}

(A12)

since $KL[q(w_{ij}|z_j = 0)||p(w_{ij}|z_j = 0)] = 0$. We assume $KL[q(w_{ij}|z_j = 1)||p(w_{ij}|z_j = 1)] = \chi$ as in [39], which means that transforming $p(w_{ij}|z_j = 1)$ to $q(w_{ij}|z_j = 1)$ requires $\chi$ nats. Thus, $K_0^\beta = [\chi_{D \times D}$. The last term is then simplified to

\begin{align*}
- \chi \sum_{j=1}^D j \beta_j
\end{align*}

(A15)
Then,

\[
\mathcal{L}_{D, \beta}^{\text{SGVB}} = \mathcal{L}_{D}^{\text{SGVB}}(\theta, \beta) - \sum_{j=1}^{D} \text{KL}[q_{\beta}(v_{i}) || p(v_{i})] - \chi d \sum_{j=1}^{D} j \beta_{j},
\]

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 \( \ell_0 \) regularization instead of a uniform \( \ell_0 \) 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, \( \delta \)-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 \( \kappa_{i} \) as the KL divergence calculated for variable \( h_{i} \). According to the parameterization in Section 4.1.1, the KL term is

\[
\text{KL}[q(h, z|x) || p(h, z)] = E_{q(x)} [\log \frac{q(z)}{p(z)}] + E_{q(h|z)} E_{q(h|x)} [\log \frac{q(h|z)}{p(h|z)}]_{\kappa=\kappa_{i}}
\]

\[
= \sum_{i} \beta_{i} \log \frac{\beta_{i}}{(1 - \pi_{i+1}) \prod_{k=1}^{i} \pi_{k}} + \sum_{i} \beta_{i} \kappa_{i} \tag{A18}
\]

Given a fixed \( \kappa_{i} \), we solve for \( \beta_{i} \) to understand the structure that VND brings to the regularization term. We could massage \( \Phi_{2} \) 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

\[
\text{KL} = \Phi_{1} + \Phi_{2} = \sum_{i} \beta_{i} \log \frac{e^{-\kappa_{i}}}{(1 - \pi_{i+1}) \prod_{k=1}^{i} \pi_{k}} \tag{A20}
\]

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

\[
\text{KL} = \sum_{i} \beta_{i} \log \frac{\beta_{i}}{e^{-\kappa_{i}}(1 - \pi_{i+1}) \prod_{k=1}^{i} \pi_{k}} \frac{C}{C} = \sum_{i} \beta_{i} \log \frac{\beta_{i}}{e^{-\kappa_{i}}(1 - \pi_{i+1}) \prod_{k=1}^{i} \pi_{k}} - \sum_{i} \beta_{i} \log C \tag{A21}
\]

The first term is now a proper KL divergence. The minimum value of 0 occurs when \( \beta_{i} = e^{-\kappa_{i}}(1 - \pi_{i+1}) \prod_{k=1}^{i} \pi_{k} / C \), then \( \text{KL} = -\log C \). In this case, if \( \kappa_{i} = 0, \forall i \), then \( C = 1 \) and \( \text{KL} = 0 \). If \( \kappa_{i} > 0 \), then \( C < 1 \) and \( \text{KL} > 0 \). If \( \kappa_{i} \) 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 \( \beta_{i} = 1 \) and \( \beta_{i} = 0, \forall i > 1 \). The form of KL in (A21) prevents such case. Consider the situation when the optimal \( \beta_{i} \) is reached:

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

B.1 Extension to Convolutional Layer

We consider a convolutional layer takes in a single tensor $H_{m}^{H} \times W \times C$ 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 $w^{D} \times H' \times W' \times C$ and outputs a matrix $F_{mj}^{H'} \times W'$. In the paper, we consider the ordered masks applied over the output channels and each filter corresponds to a dimension in $z$. As shown in [30], [42], the local reparameterization trick can be applied, due to the linearity of the convolutional layer.

$$f_{mj} = b_{mj} z_{j}^{*}, \quad \text{vec}(b_{mj}) \sim N(\gamma_{mj}, \delta_{mj}) \quad (A22)$$

where $z_{j}^{*}$ is the $j$-th dimension of the sampled ordered mask $z^{*} \sim v^{*} \sim q_{\theta}(z)$.

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).

$$\Phi_{2} = \sum_{x \in V} H' \times W' \times C \sum_{i} \sum_{j} \int w_{ij} \log \frac{q_{\theta}(w_{ij}|z_{j}^{k})}{p(w_{ij}|z_{j}^{k})} \quad (A23)$$

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.

$$E_{x \sim q_{\theta}(x), x \sim D_{te}}[F|x, z] \approx E_{x \sim D_{te}}[F|x, z = v], \quad (A24)$$

where $D_{tr}$ and $D_{te}$ are the splits of training set and testing set, and $v$ 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.

$$E_{x \sim q_{\theta}(x)}[f_{mj}] = E_{x \sim q_{\theta}(x)}[z_{j} \sum_{i=1}^{d} h_{mi} \theta_{ij}] = E_{x \sim q_{\theta}(x)}[z_{j}] \sum_{i=1}^{d} h_{mi} \theta_{ij} \quad (A25)$$

Note that, different from the probability $q_{\beta}(v_{\beta}) = \beta_{j}$, $E_{x \sim q_{\theta}(x)}[z_{j}]$ 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 $\beta = [\beta_{j}]$. Assume that the $\beta$ is also generated by a chain of hidden Bernoulli variables following (1) with the parameters $\mu = [\mu_{j}]$, 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 $\beta_{j}$’s.

$$\beta_{1} = (1 - \mu_{2}) \mu_{1} = 1 - \mu_{2} \quad (A26)$$

$$\beta_{2} = (1 - \mu_{3}) \mu_{2} \mu_{1}$$

$$\ldots$$

Solving each equation sequentially, we obtain

$$p(z_{1} = 1) = 1,$$
$$p(z_{2} = 1) = 1 - \beta_{1},$$
$$p(z_{3} = 1) = 1 - \beta_{1} - \beta_{2},$$

$$\ldots$$

(A27)

and (A25) becomes

$$E_{x \sim q_{\theta}(x)}[f_{mj}] = (1 - \sum_{k=1}^{j-1} \beta_{k}) \sum_{i=1}^{d} h_{mi} \theta_{ij}, \quad (A28)$$

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 $[\mu_{j}]$ instead of $\beta_{j}$, with $\beta_{j}$ in previous derivation replaced by $(1 - \mu_{j+1}) \mu_{j}^{2}$. The scaling factor is then $\prod_{k=1}^{j} \mu_{k}$ for each $w_{ij}$. Also, for simplicity, one can optimize $\bar{\mu}_{k}$ where $\mu_{k} = \text{sigmoid}(\bar{\mu}_{k})$.

C EXPERIMENTS

C.1 Experimental setups

We implement FN$^{3}$, individual Bayesian neural networks (IBNN) and the proposed Bayesian Nested Neural Network (BN$^{3}$) with PyTorch framework. We use the cross-entropy loss for negative expected log-likelihood. For balancing the regularization and likelihood, we add a scaling factor $\kappa$ for the KL term, which is a common trick in Bayesian learning [24].

2. We use this parameterization in our implementation, while we use $\beta_{j}$ in most of our derivation for simplicity in writing.
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\(^1\) (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. \(\kappa\) is set to \(10^{-5}\). For training the network, we use VGG11 with 1.5× 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 \([\tilde{\mu}_{ij}]\) 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 intermediate 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× 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. \(\kappa\) 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× 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 2 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.
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 to 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).