Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 222 (2023) 104–113

Procedia Computer Science

www.elsevier.com/locate/procedia

International Neural Network Society Workshop on Deep Learning Innovations and Applications (INNS DLIA 2023)

Variational Neural Networks

Illia Oleksiienko[a,*], Dat Thanh Tran[b], Alexandros Iosifidis[a]

[a]*Department of Electrical and Computer Engineering, Aarhus University, Åbogade 34, 8200 Aarhus, Denmark*
[b]*Department of Computing Sciences, Tampere University, Kalevantie 4, 33100 Tampere, Finland*

Abstract

Bayesian Neural Networks provide a tool to estimate the uncertainty of a neural network by considering a distribution over weights and sampling different models for each input. In this paper, we propose a method for uncertainty estimation in neural networks which, instead of considering a distribution over weights, samples outputs of each layer from a corresponding Gaussian distribution, parametrized by the predictions of mean and variance sub-layers. In uncertainty quality estimation experiments, we show that the proposed method achieves better uncertainty quality than other single-bin Bayesian Model Averaging methods, such as Monte Carlo Dropout or Bayes By Backpropagation methods.

© 2023 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0)
Peer-review under responsibility of the scientific committee of the International Neural Network Society Workshop on Deep Learning Innovations and Applications

Keywords: Bayesian Neural Networks; Bayesian Deep Learning; Uncertainty Estimation

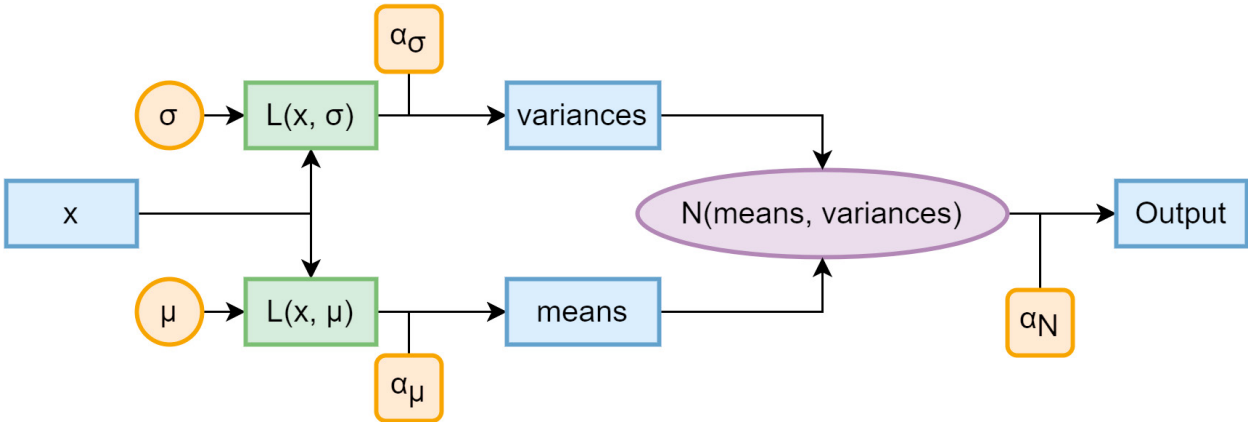
1. Introduction

The ability to estimate the uncertainty of prediction in neural networks provides advantages in using high-performing models in real-world problems, as it enables higher-level decision-making to consider such information in further actions. To do so, one needs the neural network to accompany its output with a measurement of its corresponding uncertainty for each input it processes. Several approaches have been introduced to this end, with Bayesian Neural Networks (BNNs) [5, 22, 23, 29] providing an elegant framework for estimating uncertainty of a neural network by introducing a probability distribution over its weights and sampling different models that are meant to describe the input from different points of view. This allows to determine inputs for which the network predictions are different, leading to a measurement of the network uncertainty in its outputs. Such an approach usually comes with an increased computational cost, but may be valuable for tasks where prediction errors result in high losses.

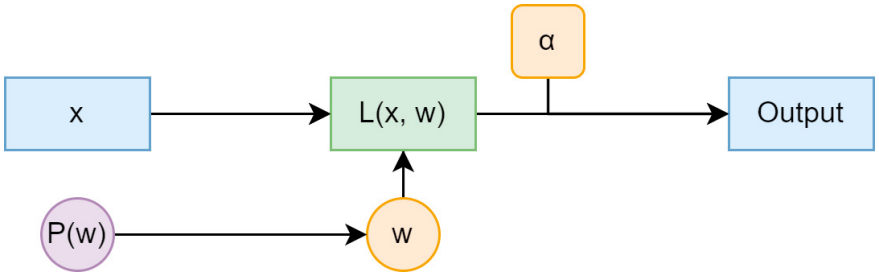
The choice of the weights' prior probability distribution function influences the statistical quality of the model and the computational resources needed to use such neural networks. This creates a possibility to explore different

* Corresponding author. Tel.: +45 91 81 13 39
E-mail address: io@ece.au.dk

1877-0509 © 2023 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0)
Peer-review under responsibility of the scientific committee of the International Neural Network Society Workshop on Deep Learning Innovations and Applications
10.1016/j.procs.2023.08.148



(a) Computational graph of a layer in the proposed VNNs.



(b) Computational graph of a layer in BNNs.

Fig. 1: Comparison of computational graphs of (a) the proposed VNNs, and (b) BNNs. BNNs consider a distribution $P(w)$ over weights and sample different weights during each inference. VNNs consider a constant set of weights and use them to generate parameters of a Gaussian distribution for each layer, outputs of which are sampled from the corresponding distribution. Layer weights are represented by μ, σ, w, activation functions by $\alpha, \alpha_\mu, \alpha_\sigma, \alpha_N$, classical layers by $L(\cdot)$, and $N(\cdot)$ represents the Gaussian distribution.

approaches to BNNs by using Gaussian [4], Bernoulli [8], Categorical [24] or other distributions. Sampling from the posterior distribution can be difficult, due to the complex nature of it. This leads to methods that avoid direct computation of the posterior, such as Markov Chain Monte Carlo (MCMC) [10] which constructs a Markov chain of samples S_i that are distributed following the desired posterior, or Variational Inference [3] which scales better than MCMC and aims to estimate a parametrized distribution that should be close to the exact posterior. How close the distributions are is computed using the Kullback-Leibler (KL) divergence [19], but it still requires the exact posterior. This is overcome by computing an Evidence Lower Bound (ELBO) instead and optimizing it with Stochastic Variational Inference (SVI) [12].

We introduce Variational Neural Networks (VNNs) which do not consider a distribution over weights, but define sub-layers to generate parameters for the output distribution of the layer. To keep computational and memory resource usage practical, we consider a Gaussian distribution with learnable mean and variance. This is achieved by using two instances of the same regular layer like convolutional or linear with different weights, and using their predictions from the inputs as means and variances of the Gaussian distribution over the outputs. We provide a neural network formulation that describes both related BNNs and the proposed VNNs in a unified manner, and show that VNNs, while being in the same group as Monte Carlo Dropout (MCD) [8] and Bayes By Backprop (BBB) [4] from the Bayesian Model Averaging (BMA) perspective [29], achieve better uncertainty quality and retain it with the increasing data dimensionality. Fig. 1 shows the difference between the computational graphs of the proposed VNNs (Fig. 1a) and the BNNs (Fig. 1b).

2. Bayesian Neural Networks

BNNs [5, 22, 23, 29] consider a distribution over their weights $p(w|a)$ and a distribution over their hyperparameters $p(a)$. A predictive distribution over an output y for a data point x can be obtained by integrating over all possible hyperparameters and model weights, i.e.:

$$p(y|x) = \int\int p(y|x,w)\ p(w|a)\ p(a)\ da\ dw. \tag{1}$$

Given a dataset $D = (X_t, Y_t)$, where X_t and Y_t are the sets of inputs $\{x\}$ and targets $\{y\}$, the distribution of weights can be derived from Bayes' theorem as $p(w|D,a) = \frac{p(Y_t|X_t,w)p(w|a)}{p(D)}$, and the corresponding predictive distribution has a form

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

Classical neural networks can be viewed as BNNs with $p(a|D) = \delta(a-\hat{a})$ and $p(w|D,a) = \delta(w-\hat{w}_a)$ [5], where $\hat{a}$ are the selected model hyperparameters, $\hat{w}_a$ are the weights, optimized by training the model, and $\delta(x)$ is the Dirac delta function which has values 0 everywhere except at $x = 0$ where it equals to 1. In this case, the predictive distribution becomes

$$\begin{aligned} p(y|x,D) &= \int\int p(y|x,w)\ p(w|D,a)\ p(a|D)\ da\ dw \\ &= \int\int p(y|x,w)\ \delta(w-\hat{w}_a)\ \delta(a-\hat{a})\ da\ dw \\ &= p(y|x,\hat{w}_{\hat{a}}), \end{aligned} \tag{3}$$

which is a distribution dictated by a loss of the network.

When training classical neural networks, hyperparameters are considered fixed at point $\hat{a}$ and weights are optimized either by maximum likelihood estimation (MLE), i.e.:

$$\begin{aligned} w_{\text{mle}} &= \underset{w}{\text{argmax}}\Big[\log\ p(D|w,\hat{a})\Big] \\ &= \underset{w}{\text{argmax}}\left[\sum_i \log\ p(Y_i|X_i,w,\hat{a})\right], \end{aligned} \tag{4}$$

or by maximum a posteriori (MAP), i.e.:

$$\begin{aligned} w_{\text{map}} &= \underset{w}{\text{argmax}}\Big[\log\ p(w|D,\hat{a})\Big] \\ &= \underset{w}{\text{argmax}}\Big[\log\ p(D|w,\hat{a}) + \log\ p(w)\Big], \end{aligned} \tag{5}$$

where $\log\ p(w)$ is a regularization term.

Due to the complexity of neural networks, direct computation of w_{mle} or w_{map} cannot be achieved, and therefore approximate methods are used to find these values. The most popular process to estimate the weight values is the Backpropagation algorithm [14], where an initial randomly selected w is updated following the direction of negative gradient of the loss function with respect to w.

3. Related Works

The use of BNNs in real-world applications is limited due to the complex nature of the possible prior and predictive distributions. Therefore, simplified versions are used. Assumptions that are proposed in different methods below aim to reduce memory, inference and training time, but they come with the cost of reducing the statistical quality of the resulting models. This problem is further discussed in Section 5.

MCD [8] considers a neural network with Dropout [28] added to each layer. The Dropout layer effectively turns off random neurons of the layer by multiplying connection weights with a random binary mask sampled from a Bernoulli distribution. This allows to avoid overfitting specific neurons. After training, standard neural networks replace Dropout with a scaled identity function and all neurons are used for inference. Instead of replacing Dropout with identity, MCD uses it during inference leading to a stochastic model. The model uncertainty for an input is computed by performing inference multiple times and computing mean and variance of predictions. BBB [4] samples model parameters from a Gaussian distribution and trains it using regular Backpropagation. By doing so, the family of models with different weights is sampled from the learned distribution, and the uncertainty of the network is computed as the variation in predictions of different samples.

Ensembles of neural networks [24] can also be used for uncertainty estimation. Ensembles are trained in parallel for the same task, but with different random seeds, which results in different weight initialization and training order. Outputs from members of an ensemble will vary, and this can be used to improve performance by taking an average of their predictions, or to estimate uncertainty by computing the variance of their outputs. Such an approach can be viewed as a BNN with a categorical distribution over weights that randomly selects one of the trained model weights. Hypermodels [7] use an additional model $\theta = g_\nu(\mathbf{z})$ to generate parameters for a base model $f_\theta(x)$. Linear Hypermodels set $g_\nu(\mathbf{z}) = a + B\mathbf{z},\ \mathbf{z} \sim \mathcal{N}(0, I)$. Using different samples of $\mathbf{z}$, one can sample different model parameters and estimate uncertainty in the same way as for the aforementioned methods.

The proposed method is based on the idea of computing parameters of a distribution and utilizing the stochastic nature of the created distribution. This idea is also studied in Mixture Density Networks (MDN) [2] and Variational Auto Encoders (VAE) [17]. Mixture Density Networks use a regular neural network, but, instead of predicting values directly in the last layer of the network, this layer is used to create a set of parameters $(\alpha_i, \mu_i, \sigma_i), i \in [1, N]$ to create a mixture model, where N is the number of distributions in the mixture. The choice of distributions in the mixture model can be arbitrary, but since the Gaussian distribution is highly flexible and easy to parametrize, it is a default choice for MDN models. The output of the model can be generated by sampling from the mixture model, or by calculating the mean and variance values of it. Variational Auto Encoder is a decoder-encoder neural network, where the output of an encoder part is a set of mean μ and variance σ values that are used to generate a Gaussian distribution $\mathcal{N}(\mu, \text{diag}[\sigma])$. Samples from these distributions are served as inputs to a decoder subnetwork, resulting in a stochastic model, in which most computations are deterministic, but the random distribution controls the second half of the model.

In this paper, we further study the approach of computing distribution parameters inside a neural network. Instead of integrating it at the end of the model, as in Mixture Density Networks, or in the middle of the Variational Auto Encoder, we build a neural network architecture fully based on this idea, implementing distribution generation inside each layer in the network, as further described in Section 4.

4. Variational Neural Networks

As introduced in Section 2, a neural network is described by its weights w and hyperparameters a. Hyperparameters include the structure of the network, i.e., the type and number of layers, their size and connections. Usually, we limit the hyperparameters by defining some of them in the beginning, e.g., by selecting that we want to use convolutional layers. This is a reasonable approach, as it is impractical to iterate through all possible types and structures of networks during training. We are using the neural network formulation $\text{NN}(x) := F^\Lambda(x, w)$, where $\text{NN}(x)$ is a neural network

applied to an input x, w are the trainable weights, F is a neural network function that incorporates structure and other hyperparameters inside it, and Λ is a set of layer implementations, which are used by F to process layer inputs.

Such a neural network formulation allows to accurately describe all the discussed uncertainty estimation methods, as well as the proposed VNNs. For instance, $\Lambda = \{\text{Conv2D}(x, w), \text{FC}(x, w)\}$ results in a regular CNN, where $\text{Conv2D}(x, w)$ is a 2D convolutional layer function and $\text{FC}(x, w)$ is a fully connected layer function. If we select $\Lambda = \{\text{Conv2D}(x, w_c \sim \mathcal{N}(\mu_c, \Sigma_c)), \text{FC}(x, w_l \sim \mathcal{N}(\mu_l, \Sigma_l))\}$ with layer weights sampled from a corresponding Gaussian distribution, then the resulting network is a BBB CNN.

4.1. Variational Layer

We define a Variational Layer (VL) that takes an input x and weights w as

$$\begin{aligned} &\text{VL}(x, w) = \alpha_{\mathcal{N}}(f(x, w)), \\ &f(x, w) \sim \mathcal{N}\Big(\alpha_\mu(L(x, \mu)), \text{diag}[(\alpha_\sigma(L(x, \sigma)))^2]\Big), \\ &w = (\mu, \sigma), \end{aligned} \tag{6}$$

where $L(x, w)$ is a regular neural network layer, such as fully connected, convolutional or a recurrent layer. $L(x, \mu)$ and $L(x, \sigma)$ represent instances of the same layer with different values of parameters and corresponding activation functions $\alpha_\mu(\cdot)$ and $\alpha_\sigma(\cdot)$. The activation function $\alpha_{\mathcal{N}}(\cdot)$ can be used to apply nonlinearity to the randomly sampled values $f(x, w)$. By selecting which of $\alpha_\mu(\cdot)$, $\alpha_\sigma(\cdot)$, $\alpha_{\mathcal{N}}(\cdot)$ are set to identity and which are set to actual activation functions (such as the Rectified Linear Unit (ReLU)) one can create networks that are described by different mathematical models. In the following, we show how different selections can lead to specific types of uncertainties, i.e., epistemic and aleatoric uncertainties [15].

4.2. Output uncertainty estimation

Estimation of prediction uncertainties in VNNs and BNNs can be done following the same formulation, but it obtained from different characteristics of these methods. Below, we first describe how BNNs can be reformulated by splitting the parametrized distribution over weights into isolated parameters and a non-parametric distribution, and then show that this formulation can be applied to VNNs.

Following [25], we consider a neural network $F(x, w)$ with a parametric distribution over weights $q_m(w)$. We assume the choice of $q_m(w)$ in a form

$$w = Q(m, z), \qquad w \sim q_m(w), \qquad z \sim p(z), \tag{7}$$

where $p(z)$ is a non-parametric distribution and $Q(\cdot)$ applies a deterministic transformation, parametrized by m, to a non-parametric random variable z. Such formulation is suitable for every uncertainty estimation method described in Section 3. For BBB models, $Q(\cdot)$ is defined as

$$Q(m, z) = \mu + \sigma^2 z, \qquad z \sim \mathcal{N}(0, I), \qquad m = (\mu, \sigma), \tag{8}$$

where we break down a parametric Gaussian distribution $\mathcal{N}(\mu, \sigma I)$ into two parts: a parametric deterministic transformation $z \longrightarrow \mu + \sigma^2 z$ and a non-parametric random variable $z \sim \mathcal{N}(0, I)$.

Defining an epistemic index $z \sim p(z)$ [25], we can formulate a deterministic neural network $F_d(\cdot)$ function that takes a draw of a random non-parametric variable z, instead of using $F(\cdot)$ with a complex distribution over w:

$$F_d(x, m, z) := F(x, w), \qquad w = Q(m, z), \qquad z \sim p(z). \tag{9}$$

With this formulation, a predictive distribution (2) for fixed hyperparameters is defined by splitting w into m and z as follows [25, 29]:

$$\begin{aligned} p(y|x, D) &= \int p(y|x, w)\, q_m(w|D)\, dw = \int p(y|x, m, z)\, p(z)\, dz, \\ \mathbb{E}[y] &= \int y\, p(y|x, D) dy \approx \frac{1}{T} \sum_{i}^{T} F_d(x, m, z_i), \\ \mathrm{Cov}[y] &= \int (y - E[y])(y - E[y])^T p(y|x, D)\, dy, \\ &\approx \frac{1}{T} \sum_{i}^{T} (E[y] - F_d(x, m, z_i))(E[y] - F_d(x, m, z_i))^T, \end{aligned} \tag{10}$$

where expectation and variance are computed using Monte Carlo integration, which can be viewed as an approximation of $p(z)$ with $\sum_{i=0}^{T} \frac{\delta(z-z_i)}{T}$, $z_i \sim p(z)$, $i \in 1, \ldots, T$ [29]. Variance of the outputs is computed by taking main diagonal values of the $\mathrm{Cov}[y]$ representing the uncertainty of the model.

VNNs, despite not having a direct distribution over weights, can also be formulated as a deterministic function $F_d(x, w, z)$ with a variational index $z \sim p(z)$. This is done by describing the output Gaussian distribution $\mathcal{N}\Big(\alpha_\mu(L(x, \mu)), \mathrm{diag}[(\alpha_\sigma(L(x, \sigma)))^2]\Big)$ of a VL as a linear transformation of a unit Gaussian $\alpha_\mu(L(x, \mu)) + \mathrm{diag}[(\alpha_\sigma(L(x, \sigma)))^2]\mathcal{N}(0, I)$.

4.3. Epistemic uncertainty

Epistemic uncertainty describes the lack of knowledge of the model and can be improved by providing a better model structure, better dataset or improved training procedure, while aleatoric uncertainty describes the uncertainty in data due to noise in data perceiving process or domain shift [9, 13]. Usually, the epistemic uncertainty in BNNs is modeled by assuming a distribution over weights and fixed hyperparameters. The use of unfixed hyperparameters leads to the Hierarchical Bayes approach [1], where the epistemic uncertainty is represented by both hyperparameters and weight distributions.

Given the fact that model parameters and structure are usually separated, the predictive distribution equation (2) holds only in the case where the hyperparameters' influence is limited to the training procedure. If the model structure is included in hyperparameters, then:

$$p(y|x, D) = \int \int p(y|x, w, a)\, p(w|D, a)\, p(a)\, da\, dw, \tag{11}$$

where the probability of a prediction for a selected model depends on both weights and hyperparameters. Following this approach, the predictive distribution of VNN in Eq. (10) can be interpreted as a predictive distribution computed for a Hierarchical BNN with a unit Gaussian distribution over hyperparameters z and a Dirac delta distribution over

weights:

$$\begin{aligned} p(y|x, D) &= \int\int p(y|x, w, z)\ p(w|D, z)\ p(z)\ dz\ dw \\ &= \int\int p(y|x, w, z)\ \delta(w - \hat{w})\ p(z)\ dz\ dw \\ &= \int p(y|x, \hat{w}, z)\ p(z)\ dz. \end{aligned} \tag{12}$$

This formulation shows that the use of the variational index z models the epistemic uncertainty in VNNs.

4.4. Aleatoric uncertainty

Fully connected and convolutional layers can be described as the operation $L(x, \lambda) = W_\lambda x + b_\lambda$, where $\lambda = (W_\lambda, b_\lambda)$, W_λ and b_λ are weights and biases of the layer, and a corresponding activation function $\alpha_\lambda(\cdot)$ can be applied to the output of $L(x, \lambda)$.

Consider a Variational Layer with $L(x, \sigma) = \Sigma,\ \ \Sigma \in \mathbb{R}$, which can be directly achieved by setting $W_\sigma = 0,\ \ b_\sigma = \Sigma$ and setting the corresponding activation function $\alpha_\sigma(\cdot)$ to identity. Applying the formulation of fully connected and convolutional layers to $f(x, w)$ (6) and using the reparametrization trick [17], we can reformulate it as follows:

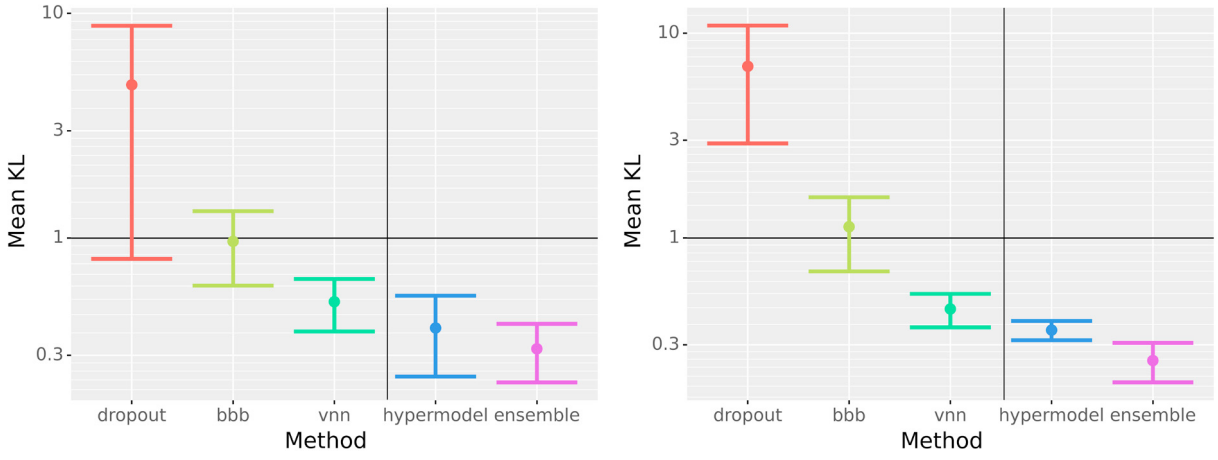
$$\begin{aligned} \epsilon &\sim \mathcal{N}(0, I), \\ f(x, w) &= \alpha_\mu(W_\mu x + b_\mu) + \Sigma\epsilon = L(x, \mu) + \epsilon_\sigma, \\ \epsilon_\sigma &\sim \mathcal{N}(0, \Sigma I). \end{aligned} \tag{13}$$

In this formulation, ϵ_σ models the aleatoric uncertainty [13] for the next subnetwork, which takes outputs of the current layer as inputs and cannot improve this uncertainty by improving the model.

4.5. Training

Training of VNNs is performed by averaging outputs from different network passes for the same input. The Backpropagation process is performed similarly as in Variational Auto Encoders [17], which is based on the reparametrization trick. The Gaussian distribution over the outputs of the layer in VNNs is represented as a sum and multiplication of a non-parametric unit-Gaussian random variable and a set of deterministic variables, obtained from the corresponding sub-layers. While computing the gradient, the value of the random variable is fixed, and is used in the process of computing the error responsibilities of model parameters. Networks can also be trained with a single pass, which results in the same training procedure as for classical neural networks. The models are trained with the usual loss functions that are suitable for the task.

Training of VNNs is stochastic, which means that for the same model weights and input data, the Backpropagation algorithm will result in different updates if applied multiple times. This may resemble stochastic training approaches, such as Stochastic Gradient Descent (SGD) [16, 27] or Adaptive Gaussian Noise Injection [21]. However, the source of noise in VNNs is heavily dependent on both layer inputs and parameters and not applied to the network's weights or outputs in a constant or weight-scaled manner. Furthermore, the stochastic manner of VNNs, as well as BNNs, is not limited to training, as the inference of the trained models is also stochastic. The fact that VNNs are not limited to a training framework does not mean that they cannot be used as such, which is a direction for future work.



(a) All parameters

(b) $D_x = 1000$

Fig. 2: Comparison of mean KL value with 1 STD range for each method averaged across different experiment parameters.

5. Experiments

A recently proposed framework called Epistemic Neural Networks [25] aims to provide a possibility to rank BNNs based on their ability to accurately estimate output uncertainty. This is done by first generating a synthetic dataset $D_T = \{(x, y)_t \text{ for } t \in [0, T-1]\}$ for a simple regression task $y = f(x) + \epsilon$, where y is an output scalar, x is an input data point with D_x number of dimensions, ϵ is a random variable sampled from a Gaussian distribution $\mathcal{N}(0, \sigma^2)$ representing an aleatoric uncertainty. The dataset size T is determined as $T = D_x\lambda$, where λ is a hyperparameter, meaning that more data points are created for a higher dimensionality of x. The dataset is used to train a Neural Network Gaussian Process (NNGP) [20] and an uncertainty estimation model of interest. NNGP serves as an ideal probabilistic model for this data, and a predictive distribution of a selected uncertainty estimation model should be as close as possible to the predictive distribution of the NNGP model. The above process is used to create two datasets, one used for training the uncertainty estimation model and one (test set) used to evaluate the uncertainty estimation performance. Following [25], random noise is added to the data belonging to the training set, as it has been shown to increase the uncertainty estimation performance, which is measured by computing the KL-divergence between the true posterior $\mathcal{N}(\mu_{\text{GP}}, k_{\text{GP}})$ and a model predictive distribution $\mathcal{N}(\mu_B, k_B)$. Lower values of KL-divergence represent better uncertainty quality for a selected model, and therefore can be used to rank different approaches for uncertainty estimation.

We implement VNNs inside the ENN's JAX implementation [26] to reproduce results for BBB [4], MCD [8], Ensemble [24], Hypermodel [7] and compare them with VNN[1]. We follow the original framework parameters and repeat experiments with the following options: $D_x \in \{10, 100, 1000\}$, $\lambda \in \{1, 10, 100\}$, and $\epsilon \in \{0.01, 0.1, 1\}$. Each model is trained with 10 different random seeds, and the resulting KL value is the average of individual runs. The average KL values for all experiment parameters are given in Fig. 2a and for the highest input dimension value $D_x = 1000$ are given in Fig. 2b. VNN has better uncertainty quality than BBB and MCD, but it is outperformed by Hypermodel and Ensemble. This can be explained by the difference in BMA for Deep Ensembles and Variation Inference methods, as explained in [29]. The weight probability distribution can be split into basins, where models sampled from the same basin are too similar and will describe the problem from the same point of view, resulting in multiple entries of actually identical model in the prediction voting. Deep Ensembles and Hypermodels avoid this problem by not having a single anchor point with small weight deviations, and therefore having high chances of converging trained models into different basins. This means that VNN has a higher chance than Ensemble to have its

[1] The code is available at https://github.com/ilialiiliili/vnn-pytorch-jax

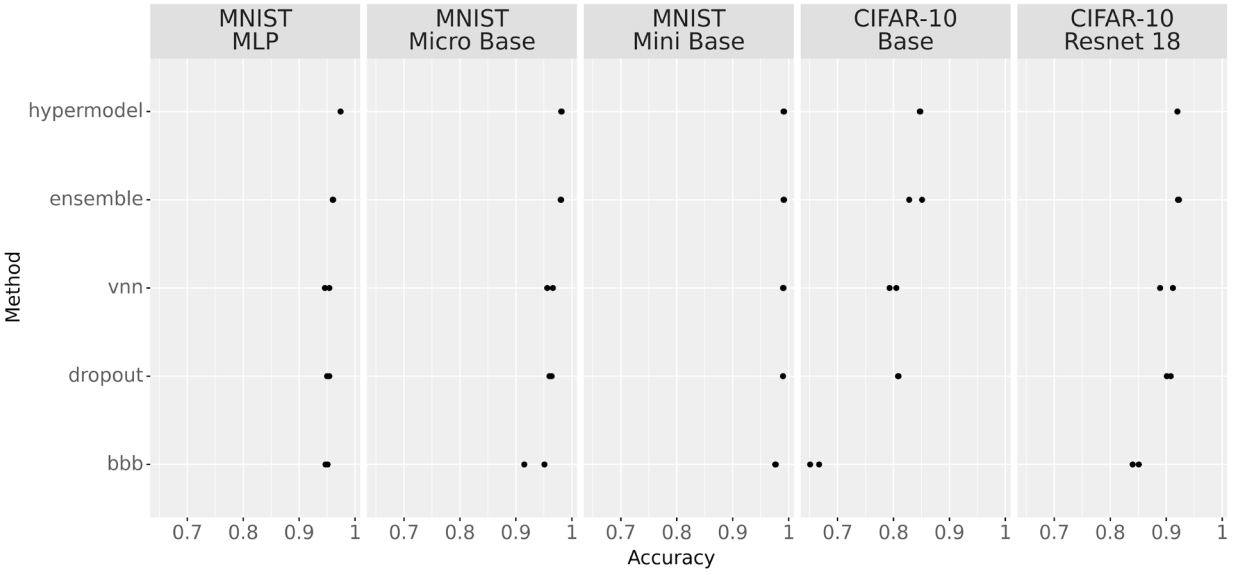


Fig. 3: Comparison of classification accuracy on MNIST and CIFAR-10 datasets with different model architectures.

samples in a single basin, placing it in the same group as BBB and MCD. Additionally, with bigger data dimensionality D_x, MCD and BBB achieve worse results, while VNN, Ensemble and Hypermodel perform better.

We further perform experiments on image classification. We train the same methods for image classification tasks on MNIST [6] and CIFAR-10 [18] datasets. To show the influence of model architecture on the performance, we use a set of architectures $\{F_i\}$ and train each method with the selected architecture F. We select a Base architecture to have 3 convolutional and 1 linear layer for MNIST, and 6 convolutional and 1 linear layer for CIFAR-10. Mini and Micro Base architectures have the same layer structure as the Base one, but a lower number of channels in each layer. MLP architecture consists of 3 fully connected layers. We also use Resnet-18 [11] architecture for experiments on CIFAR-10. For each method, we train models with different hyperparameter values and select the best two models for comparison. The results of classification experiments are given in Fig. 3 and are roughly following the results of uncertainty quality estimation experiments.

6. Conclusion

We proposed Variational Neural Networks that consider a Gaussian distribution over outputs of each layer, the mean and variance of which are generated by the corresponding sub-layers, and evaluated their uncertainty estimation quality within the Epistemic Neural Networks framework. Experiments show that, despite having similar properties of Bayesian Model Averaging to Monte Carlo Dropout and Bayes By Backpropagation, where sampled models are close resulting in similar models' points of view, VNNs achieve better uncertainty quality which is retained when data dimensionality is increased, in contrast to Monte Carlo Dropout and Bayes By Backpropagation methods.

Acknowledgments

We thank Dr. Martin Magris for helpful discussions and feedback.

References

[1] Allenby, G.M., Rossi, P.E., 2006. Hierarchical bayes models. The handbook of marketing research , 418–440.
[2] Bishop, C.M., 1994. Mixture density networks.
[3] Blei, D.M., Kucukelbir, A., McAuliffe, J.D., 2017. Variational inference: A review for statisticians. JASA 112, 859–877.

[4] Blundell, C., Cornebise, J., Kavukcuoglu, K., Wierstra, D., 2015. Weight Uncertainty in Neural Networks, in: ICML, pp. 1613–1622.
[5] Charnock, T., Perreault-Levasseur, L., Lanusse, F., 2020. Bayesian neural networks. arXiv:2006.01490 .
[6] Deng, L., 2012. The mnist database of handwritten digit images for machine learning research. IEEE Signal Process. Mag. 29, 141–142.
[7] Dwaracherla, V., Lu, X., Ibrahimi, M., Osband, I., Wen, Z., Roy, B.V., 2020. Hypermodels for exploration, in: ICLR.
[8] Gal, Y., Ghahramani, Z., 2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning, in: ICML, pp. 1050–1059.
[9] Gawlikowski, J., Tassi, C.R.N., Ali, M., Lee, J., Humt, M., Feng, J., Kruspe, A.M., Triebel, R., Jung, P., Roscher, R., Shahzad, M., Yang, W., Bamler, R., Zhu, X.X., 2021. A survey of uncertainty in deep neural networks. arxiv:2107.03342 .
[10] Hastings, W.K., 1970. Monte Carlo sampling methods using Markov chains and their applications. Biometrika 57, 97–109.
[11] He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition, in: CVPR, pp. 770–778.
[12] Hoffman, M., Blei, D.M., Wang, C., Paisley, J., 2013. Stochastic variational inference. JMLR 14, 1303–1347.
[13] Hüllermeier, E., Waegeman, W., 2021. Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods. Machine Learning 110, 457–506.
[14] Kelley, H.J., 1960. Gradient theory of optimal flight paths. ARSJ 30, 947–954.
[15] Kendall, A., Gal, Y., 2017. What uncertainties do we need in bayesian deep learning for computer vision?, in: NeurIPS.
[16] Kiefer, J., Wolfowitz, J., 1952. Stochastic Estimation of the Maximum of a Regression Function. The Annals of Mathematical Statistics 23, 462 – 466.
[17] Kingma, D.P., Welling, M., 2014. Auto-encoding variational bayes, in: ICLR.
[18] Krizhevsky, A., 2009. Learning multiple layers of features from tiny images. Technical Report.
[19] Kullback, S., Leibler, R.A., 1951. On Information and Sufficiency. Ann. Math. Stat. 22, 79 – 86.
[20] Lee, J., Bahri, Y., Novak, R., Schoenholz, S.S., Pennington, J., Sohl-Dickstein, J., 2018. Deep neural networks as gaussian processes, in: ICLR.
[21] Li, Y., Liu, F., 2020. Adaptive gaussian noise injection regularization for neural networks, in: Han, M., Qin, S., Zhang, N. (Eds.), Advances in Neural Networks – ISNN 2020.
[22] Mackay, D.J.C., 1995. Probable networks and plausible predictions — a review of practical bayesian methods for supervised neural networks. Network 6, 469–505.
[23] Magris, M., Iosifidis, A., 2022. Bayesian learning for neural networks: an algorithmic survey. arxiv:2211.11865 .
[24] Osband, I., Aslanides, J., Cassirer, A., 2018. Randomized prior functions for deep reinforcement learning, in: NeurIPS, pp. 8626–8638.
[25] Osband, I., Wen, Z., Asghari, M., Ibrahimi, M., Lu, X., Roy, B.V., 2021a. Epistemic Neural Networks. arXiv:2107.08924 .
[26] Osband, I., Wen, Z., Asghari, M., Ibrahimi, M., Lu, X., Roy, B.V., 2021b. Github code for epistemic neural networks. https://github.com/deepmind/enn.
[27] Robbins, H.E., 1951. A stochastic approximation method. Annals of Mathematical Statistics 22, 400–407.
[28] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2014. Dropout: A simple way to prevent neural networks from overfitting. JMLR 15, 1929–1958.
[29] Wilson, A.G., Izmailov, P., 2020. Bayesian Deep Learning and a Probabilistic Perspective of Generalization, in: NeurIPS.