Learning Structured Weight Uncertainty in Bayesian Neural Networks

Shengyang Sun
Tsinghua University

Changyou Chen✉
Duke University

Lawrence Carin
Duke University

Abstract

Deep neural networks (DNNs) are increasingly popular in modern machine learning. Bayesian learning affords the opportunity to quantify posterior uncertainty on DNN model parameters. Most existing work adopts independent Gaussian priors on the model weights, ignoring possible structural information. In this paper, we consider the matrix variate Gaussian (MVG) distribution to model structured correlations within the weights of a DNN. To make posterior inference feasible, a reparametrization is proposed for the MVG prior, simplifying the complex MVG-based model to an equivalent yet simpler model with independent Gaussian priors on the transformed weights. Consequently, we develop a scalable Bayesian online inference algorithm by adopting the recently proposed probabilistic backpropagation framework. Experiments on several synthetic and real datasets indicate the superiority of our model, achieving competitive performance in terms of model likelihood and predictive root mean square error. Importantly, it also yields faster convergence speed compared to related Bayesian DNN models.

1 Introduction

Modern machine learning has witnessed significant success with deep neural networks (DNNs), obtaining state-of-the-art results on various real-world tasks, such as image classification (Krizhevsky et al., 2012; He et al., 2016), machine translation (Sutskever et al., 2014; Wu et al., 2016), image captioning (Xu et al., 2015; Pu et al., 2016) and game playing (Silver et al., 2016). DNNs are typically trained with stochastic optimization algorithms (Bottou, 1998). Care must be taken to prevent overfitting when training. Bayesian methods help mitigate this problem by imposing appropriate priors on network weights to model weight uncertainty, while performing model averaging based on posterior samples when testing (for MCMC-based methods).

In many DNNs, data are given as input-output pairs, with the output a sample drawn from an *output probability distribution* based on a forward nonlinear transformation of the input. In Bayesian neural networks, priors are imposed on the global weights of the network to capture weight uncertainty. Posterior inference is often performed by variational inference, to model multiclass-output probability distributions (for classification) (Blundell et al., 2015; Gal and Ghahramani, 2016) and continuous-output distributions (for regression) (Louizos and Welling, 2016). Variants such as probabilistic backpropagation (PBP) have been considered for general output distributions (Hernández-Lobato and Adams, 2015; Ghosh et al., 2016). Another line of research for posterior inference uses stochastic gradient Markov Chain Monte Carlo (SG-MCMC) (Li et al., 2016; Lu et al., 2016). Although competitive performance has been obtained by SG-MCMC for classification, the nature of MCMC hinders its practical use for *fast* testing in real applications. Consequently, we build our model based on the PBP framework.

For most existing work, weight uncertainty is modeled with independent Gaussian priors Blundell et al. (2015); Hernández-Lobato and Adams (2015); Li et al. (2016), ignoring the structural information associated with the weights. This is somewhat counter-intuitive because the weights, represented as matrices, may be best considered as a whole. In this paper, we adopt a structured prior on matrices, called the matrix variate Gaussian (MVG) distribution (Gupta and Nagar, 1999), to model both row and column correlations for a weight matrix in a DNN. The MVG distribution is flexible in modeling matrix-valued random variables.

Proceedings of the 20^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2017, Fort Lauderdale, Florida, USA. JMLR: W&CP volume 54.

However, posterior inference becomes difficult when applying it to model a DNN. To remedy this problem, we first propose a reparameterization of the MVG, to decompose an MVG-distributed random variable to the product of two deterministic matrices (related to the hyperparameters of a MVG distribution), and a transformed weight matrix with independent Gaussian priors. This reparameterization conveniently transforms our MVG-based model to a simpler yet more tractable model, such that assumed density filtering (Minka, 2001) (ADF) can be applied via the PBP framework (Hernández-Lobato and Adams, 2015) for efficient posterior inference. We compare our model with recently proposed Bayesian DNN models on a number of datasets, obtaining superior performance, in terms of either prediction accuracy or convergence speed.

Note a recent work of Louizos and Welling (2016) also introduces MVG priors for the weights of an DNN. Posterior inference is performed by standard variational inference, as well as using the reparameterization trick introduced by Kingma and Welling (2014) to approximate the infeasible expectation in the variational bound with samples. Compared to their method, the proposed reparameterization of an MVG distribution makes our model naturally fit into the PBP framework, requiring fewer approximations and converging faster than variational inference.

2 Preliminaries

We briefly review basic definitions of the DNN, then introduce the MVG distribution and list some of its properties. Notation will be introduced throughout the paper where necessary, for ease of understanding. We denote vectors as bold lower-case letters, and matrices as bold upper-case letters.

2.1 Deep neural networks

We focus on DNNs with L layers. Starting from the input data, each layer ℓ is represented as a nonlinear transformation, denoted as $g_{\mathbf{W}_\ell} : \mathbb{R}^{V_{\ell-1}} \longmapsto \mathbb{R}^{V_\ell}$, where V_ℓ is the number of neurons in layer* ℓ, and $\mathbf{W}_\ell$ represents the corresponding model parameters. For simplicity, we drop the parameter $\mathbf{W}$ and rewrite $g_{\mathbf{W}_\ell}$ as g_ℓ in the following. As a result, an L-layer DNN with input, $\mathbf{x} \in \mathbb{R}^{V_0}$, can be represented as a set of nonlinear function compositions (MacKay, 1992), *i.e.*, the output of the top layer, $\mathbf{z}_L$, is the result of composing L nonlinear functions:

$$\mathbf{z}_L = g_L \circ g_{L-1} \circ \cdots \circ g_1(\mathbf{x}) \ , \tag{1}$$

where $\circ$ represents function composition, *i.e.*, $\mathcal{A} \circ \mathcal{B}$ means $\mathcal{A}$ is evaluated on the output of $\mathcal{B}$. On top of $\mathbf{z}_L$, a probability distribution is often imposed to model the output data, *e.g.*, in many DNNs, a multinomial distribution parameterized by the softmax output of $\mathbf{z}_L$ is used to model the output label $\mathbf{y}$.

We consider the feedforward neural network† (FNN), such that $\mathbf{W}_\ell \in \mathbb{R}^{V_{\ell-1} \times V_\ell}$, and g_ℓ takes the form:

$$g_\ell(\mathbf{x}) \triangleq \mathrm{ReLU}\left(\mathbf{W}_\ell^T \mathbf{x} + \mathbf{b}_\ell\right) \ ,$$

where $\mathbf{b}_\ell \in \mathbb{R}^\ell$ is the bias term, $\mathrm{ReLU}(x) \triangleq \max(0, x)$ is the Rectified Linear Unit (ReLU) activation function (Glorot et al., 2011). Note that $\mathbf{b}_\ell$ can be absorbed into $\mathbf{W}_\ell$ by augmenting the input with an additional dimension with value one, thus we will ignore the term $\mathbf{b}_\ell$ and write $g_\ell(\mathbf{x}) \triangleq \mathrm{ReLU}(\mathbf{W}_\ell^T \mathbf{x})$ in the following.

2.2 Matrix variate Gaussian distributions

The matrix variate Gaussian (MVG) distribution is a three-parameter distribution describing the probability of a random matrix $\mathbf{W} \in \mathbb{R}^{l_1 \times l_2}$, with density function:

$$\begin{aligned} p(\mathbf{W}) &\triangleq \mathcal{MN}(\mathbf{W}; \mathbf{M}, \mathbf{U}, \mathbf{V}) \qquad (2) \\ &= \frac{\exp\left(-\frac{1}{2}\mathrm{tr}\left[\mathbf{V}^{-1}(\mathbf{W}-\mathbf{M})^T \mathbf{U}^{-1}(\mathbf{W}-\mathbf{M})\right]\right)}{(2\pi)^{l_1 l_2/2} |\mathbf{V}|^{l_1/2} |\mathbf{U}|^{l_2/2}} \ , \end{aligned}$$

where $\mathbf{M} \in \mathbb{R}^{l_1 \times l_2}$ is the mean of the distribution, $\mathbf{U} \in \mathbb{R}^{l_1 \times l_1}$ and $\mathbf{V} \in \mathbb{R}^{l_2 \times l_2}$ encode covariance information of the rows and columns of $\mathbf{W}$, respectively, and are invertible.

Lemma 1. *Let* $\mathbf{W}$ *follows the MVG distribution in* (2)*, then*

$$vec(\mathbf{W}) \sim \mathcal{N}(vec(\mathbf{M}), \mathbf{V} \otimes \mathbf{U}) \ ,$$

where $vec(\mathbf{M})$ *is the vectorization of* $\mathbf{M}$ *by stacking the columns of* $\mathbf{M}$*, and* $\otimes$ *denotes the standard Kronecker product (Henderson and Searle, 1980).*

Lemma 2. *Let* $\mathbf{W}$ *follows the MVG distribution in* (2)*,* $\mathbf{A} \in \mathbb{R}^{l_0 \times l_1}$*,* $\mathbf{C} \in \mathcal{R}^{l_2 \times l_3}$*, then*

$$\begin{aligned} \mathbf{B} &\triangleq \mathbf{A}\,\mathbf{W} \sim \mathcal{MN}(\mathbf{B}; \mathbf{A}\,\mathbf{M}, \mathbf{A}^T\,\mathbf{U}\,\mathbf{A}, \mathbf{V}) \\ \mathbf{B} &\triangleq \mathbf{W}\,\mathbf{C} \sim \mathcal{MN}(\mathbf{B}; \mathbf{M}\,\mathbf{C}, \mathbf{U}, \mathbf{C}^T\,\mathbf{V}\,\mathbf{C}) \ . \end{aligned}$$

Lemma 1 indicates that an MVG distribution is a structured Gaussian distribution, with covariance matrix $\mathbf{V} \otimes \mathbf{U}$; Lemma 2 reveals the distribution property of a linear transformed MVG random variable, useful in deriving our reparametrization for the MVG below.

*For conciseness, we use V_0 to represent the input (data) dimension.

†More complex network structures such as the convolutional neural network can also be adapted to our framework.

3 Deep Neural Networks with Matrix Variate Gaussian Priors

We consider a Bayesian DNN model, assigning MVG priors on the weights, and model the top-layer output with a multivariate Gaussian random variable. To make model inference tractable, we first simplify the model by introducing a reparametrization of the MVG distribution.

Denote all the weight parameters as $\mathbf{W} \triangleq \{\mathbf{W}_\ell\}_{\ell=1}^L$. To explicitly write out the dependency of the DNN output $\mathbf{z}_L$ in (1) with $\mathbf{W}$, we rewrite it as $f(\mathbf{x}, \mathbf{W}) \triangleq \mathbf{z}_L$. Following Hernández-Lobato and Adams (2015), we consider the final output of a DNN to be a Gaussian random variable[‡] $\mathbf{y}$ with mean $f(\mathbf{x}, \mathbf{W})$ and covariance matrix $\sigma^{-1}\mathbf{I}_{V_L}$, where $\mathbf{I}_n$ denotes an identity matrix of size $n \times n$. Furthermore, we assign an MVG prior for each of the weights in $\mathbf{W}$, *i.e.*, for $\ell = 1, \cdots, L$:

$$\begin{aligned} \mathbf{y} \,|\, \mathbf{x}, \mathbf{W}, \gamma &\sim \mathcal{N}\left(\mathbf{y}; f(\mathbf{x}, \mathbf{W}), \gamma^{-1}\mathbf{I}_{V_L}\right) , \\ \mathbf{W}_\ell \,|\, \mathbf{M}_\ell, \mathbf{U}_\ell, \mathbf{V}_\ell &\sim \mathcal{MN}\left(\mathbf{W}_\ell; \mathbf{M}_\ell, \mathbf{U}_\ell, \mathbf{V}_\ell\right), \end{aligned} \quad (3)$$

For layer ℓ of the DNN, following Hernández-Lobato and Adams (2015), we rescale the corresponding inputs from the layer below by factor $1/\sqrt{V_{\ell-1}+1}$, *i.e.*, $g_\ell(\mathbf{z}_{\ell-1})$ is defined as: $g_\ell(\mathbf{z}_{\ell-1}) \triangleq \text{ReLU}\left(\mathbf{W}_\ell^T\, \mathbf{z}_{\ell-1}/\sqrt{V_{\ell-1}+1}\right)$. This makes the scale of the input to each neuron independent of the number of incoming connections.

Note from (3) that if we assume each element of $\mathbf{W}_\ell$ to be independent Gaussian distributed with mean 0 and variance λ^{-1}, we recover the Gaussian DNN model proposed in (Hernández-Lobato and Adams, 2015). The MVG priors for $\mathbf{W}$ allows one to model correlations between the elements in each weight matrix, making the model more flexible and robust. However, it brings significant challenges for posterior inference, especially in a big-data setting.

3.1 Reparameterization of the MVG

Since $\mathbf{U}_\ell$ and $\mathbf{V}_\ell$ are positive-definite matrices, we write their orthogonal decomposition as:

$$\mathbf{U}_\ell = \mathbf{P}_\ell\, \mathbf{\Lambda}_\ell^{(1)}\, \mathbf{P}_\ell^T, \quad \mathbf{V}_\ell = \mathbf{Q}_\ell\, \mathbf{\Lambda}_\ell^{(2)}\, \mathbf{Q}_\ell^T \ ,$$

where $\mathbf{P}_\ell$ and $\mathbf{Q}_\ell$ are the corresponding orthogonal matrices, $\mathbf{\Lambda}_\ell^{(1)}$ and $\mathbf{\Lambda}_\ell^{(2)}$ are diagonal matrices with positive diagonal elements. Defining $\mathbf{B}_\ell \triangleq \mathbf{P}_\ell^T\, \mathbf{W}_\ell\, \mathbf{Q}_\ell$, by applying Lemma 2, we have that

$$\mathbf{B}_\ell \sim \mathcal{MN}\left(\mathbf{B}_\ell; \mathbf{P}_\ell^T\, \mathbf{M}_\ell\, \mathbf{Q}_\ell, \mathbf{\Lambda}_\ell^{(1)}, \mathbf{\Lambda}_\ell^{(2)}\right) \ , \quad (4)$$

[‡]Other output distributions such as the Poisson distribution for count-valued output can be similarly dealt with by adopting techniques from Ghosh et al. (2016).

or equivalently (by Lemma 1):

$$\text{vec}(\mathbf{B}_\ell) \sim \mathcal{N}\left(\text{vec}(\mathbf{B}_\ell); \text{vec}(\mathbf{P}_\ell^T\, \mathbf{M}_\ell\, \mathbf{Q}_\ell), \mathbf{\Lambda}_\ell^{(1)} \otimes \mathbf{\Lambda}_\ell^{(2)}\right) \ ,$$

where $\mathbf{\Lambda}_\ell^{(1)} \otimes \mathbf{\Lambda}_\ell^{(2)}$ is a diagonal matrix. This makes all the elements of $\mathbf{B}_\ell$ independent of each other. Consequently, we conclude that modeling $\mathbf{W}_\ell$ with an MVG prior is equivalent to modeling $\mathbf{B}_\ell$ with independent Gaussian priors for each element, and then transforming $\mathbf{B}_\ell$ with two orthogonal matrices related to the distribution of $\mathbf{W}_\ell$, as illustrated in Figure 1.

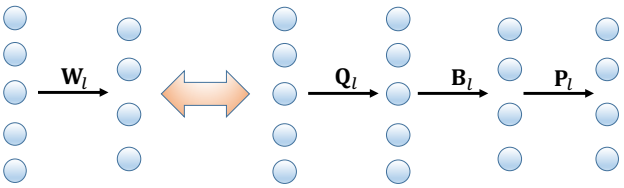

Figure 1: Equivalence between a DNN with an MVG prior on its weights and a DNN with two rotations via $\mathbf{P}_\ell$ and $\mathbf{Q}_\ell$ (related to the hyperparameters of the MVG distribution) and independent Gaussian priors on the new weights $\mathbf{B}_\ell$.

We also must address the orthogonal matrices $\mathbf{P}_\ell$ and $\mathbf{Q}_\ell$, such that standard priors can be imposed for them. Fortunately, a basic argument from linear algebra (Golub and Van Loan, 1996) shows that there exists vectors $\mathbf{v}_\ell^{(1)}$ and $\mathbf{v}_\ell^{(2)}$ of lengths V_ℓ and $V_{\ell-1}$, respectively, such that:

$$\begin{aligned} \mathbf{P}_\ell &= \mathbf{I}_{V_\ell} - 2\, \mathbf{v}_\ell^{(1)}\, \mathbf{v}_\ell^{(1)^T} / \left(\mathbf{v}_\ell^{(1)^T}\, \mathbf{v}_\ell^{(1)}\right), \\ \mathbf{Q}_\ell &= \mathbf{I}_{V_{\ell-1}} - 2\, \mathbf{v}_\ell^{(2)}\, \mathbf{v}_\ell^{(2)^T} / \left(\mathbf{v}_\ell^{(2)^T}\, \mathbf{v}_\ell^{(2)}\right) \ . \end{aligned} \quad (5)$$

This allows a reparameterization of $\mathbf{P}_\ell$ and $\mathbf{Q}_\ell$ with $\mathbf{v}_\ell^{(1)}$ and $\mathbf{v}_\ell^{(2)}$. The correctness of this reparameterization can be proved by verifying that $\mathbf{P}_\ell\, \mathbf{P}_\ell{}^T = \mathbf{I}_{V_\ell}$ and $\mathbf{Q}_\ell\, \mathbf{Q}_\ell{}^T = \mathbf{I}_{V_{\ell-1}}$. Using this reparameterization, the orthogonality constraints in $\mathbf{P}_\ell$ and $\mathbf{Q}_\ell$ can be removed, and standard priors such as a Gaussian prior can be adopted in the model.

To increase model capacity, we can replace the orthogonal matrices above with a product of orthogonal matrices, *e.g.*, $\mathbf{P} = \mathbf{P}_1\, \mathbf{P}_2 \cdots \mathbf{P}_n$, where $\mathbf{P}_i$ is in the same form of (5). Similar strategy applies for $\mathbf{Q}$. In our implementation, we use $n = 1$ for simplicity

3.2 The full model

Let $\mathbf{B} \triangleq \{\mathbf{B}_\ell\}_{\ell=1}^L$, $\mathbf{V}^{(1)} \triangleq \{\mathbf{v}_\ell^{(1)}\}_{\ell=1}^L$, $\mathbf{V}^{(2)} \triangleq \{\mathbf{v}_\ell^{(2)}\}_{\ell=1}^L$. As shown above, the model defined in

(3) can be equivalently modeled with parameters $\{\mathbf{B}, \mathbf{V}^{(1)}, \mathbf{V}^{(2)}\}$ by the reparameterizations in (4) and (5). We further impose hierarchical priors for the hyperparameters $\{\mathbf{M}_\ell, \mathbf{U}_\ell, \mathbf{V}_\ell\}$ in (3). When choosing independent Gaussian priors for these hyperparameters, it is equivalent to imposing independent Gaussian priors for $\{\mathbf{B}, \mathbf{V}^{(1)}, \mathbf{V}^{(2)}\}$. As a result, the full Bayesian DNN model is defined as[§]:

$$
\begin{aligned}
p(\mathbf{y}_n \,|\, \mathbf{x}_n, \mathbf{B}, \gamma) &= \mathcal{N}\left(\mathbf{y}_n; f(\mathbf{x}_n, \mathbf{P}_L\, \mathbf{B}_L\, \mathbf{Q}_L^T), \gamma^{-1}\, \mathbf{I}_{V_L}\right), \\
p(\mathbf{B}\,|\lambda) &= \prod_{\ell=1}^{L} \prod_{i=1}^{V_\ell} \prod_{j=1}^{V_{\ell-1}} \mathcal{N}\left((\mathbf{B}_\ell)_{ij}; 0, \lambda^{-1}\right) \ , \\
p(\mathbf{V}^{(1)}\,|\phi) &= \prod_{\ell=1}^{L} \mathcal{N}\left(\mathbf{v}_\ell^{(1)}; \mathbf{0}, \phi^{-1}\, \mathbf{I}_{V_\ell}\right) \ , \\
p(\mathbf{V}^{(2)}\,|\psi) &= \prod_{\ell=1}^{L} \mathcal{N}\left(\mathbf{v}_\ell^{(2)}; \mathbf{0}, \psi^{-1}\, \mathbf{I}_{V_{\ell+1}}\right) \ , \\
\gamma, \lambda, \phi, \psi &\sim \text{Gamma}(x; \alpha_0^x, \beta_0^x) \ ,
\end{aligned}
\tag{6}
$$

for $n = 1, \cdots, N$, and $x \in \{\gamma, \lambda, \phi, \psi\}$. Letting $\mathbf{R} \triangleq \{\mathbf{B}, \mathbf{V}^{(1)}, \mathbf{V}^{(2)}, \gamma, \lambda, \phi, \psi\}$ represent all model parameters, the task is to compute the posterior $p(\mathbf{R} \,|\, \mathbf{X}, \mathbf{y})$, for given data $\{\mathbf{X}, \mathbf{y}\} = (\mathbf{x}_n, \mathbf{y}_n)_{n=1}^N$.

4 Approximate Posterior Inference with Probabilistic Backpropagation

Following Hernández-Lobato and Adams (2015), we adopt the assumed density filtering method (ADF) (Opper, 1998) for approximate posterior inference. In ADF, we use a set of variational distributions to approximate the true posterior distribution. We propose

$$
\begin{aligned}
q(\mathbf{B}) &= \prod_{\ell=1}^{L} \prod_{i=1}^{V_\ell} \prod_{j=1}^{V_{\ell-1}} \mathcal{N}\left((\mathbf{B}_\ell)_{ij}; m_{ij\ell}, \lambda_{ij\ell}^{-1}\right) \\
q(\mathbf{V}^{(1)}) &= \prod_{\ell=1}^{L} \mathcal{N}\left(\mathbf{v}_\ell^{(1)}; \mathbf{m}_\ell^{(1)}, \Sigma_\ell^{(1)}\right) \\
q(\mathbf{V}^{(2)}) &= \prod_{\ell=1}^{L} \mathcal{N}\left(\mathbf{v}_\ell^{(2)}; \mathbf{m}_\ell^{(2)}, \Sigma_\ell^{(2)}\right) \\
\gamma, \lambda, \phi, \psi &\sim \text{Gamma}(x; \alpha_x, \beta_x)
\end{aligned}
\tag{7}
$$

The posterior of output $\{\mathbf{y}_n\}_{n=1}^N$ and local hidden random variables $\{\mathbf{z}_{n\ell}\}$ are approximated with Gaussian variational distributions as:

$$
\begin{aligned}
q(\mathbf{y}_n) &= \mathcal{N}\left(\mathbf{y}_n; \mathbf{m}_{\mathbf{z}_{nL}}, \text{diag}(\mathbf{v}_{\mathbf{z}_{nL}})\right) \ , \\
q(\mathbf{z}_{n\ell}) &= \mathcal{N}\left(\mathbf{z}_{n\ell}; \mathbf{m}_{\mathbf{z}_{n\ell}}, \text{diag}(\mathbf{v}_{\mathbf{z}_{n\ell}})\right) \ ,
\end{aligned}
\tag{8}
$$

with $\mathbf{v}_{\mathbf{z}_{nL}} \in \mathbb{R}_+^{V_L}$. We will refer to $\mathbf{v}_{\mathbf{z}_{nL}}$ as variance vector in the following.

4.1 Online posterior inference with ADF

Assumed density filtering is an online inference algorithm that incrementally updates the variational distributions for model parameters after observing new evidence (data)[¶] (Minka, 2001). Generally, this is done by iteratively replacing one term from the true posterior (called a *factor*) with the corresponding term from the variational distribution, to form a new approximate posterior. The approximate posterior is then projected back to the variational distribution for an update.

Take the update on $(\mathbf{B}_\ell)_{ij}$ as an example. Let $q((\mathbf{B}_\ell)_{ij} \,|\, \mathbf{R}_{-ij\ell}^{(n-1)})$ be the approximate posterior (variational distribution) after seeing data $\{(\mathbf{x}_1, \mathbf{y}_1), \cdots, (\mathbf{x}_{n-1}, \mathbf{y}_{n-1})\}$, where $\mathbf{R}_{-ij\ell}^{(n-1)}$ denotes the current model parameters except $(\mathbf{B}_\ell)_{ij}$. When data $(\mathbf{x}_n, \mathbf{y}_n)$ comes, the posterior of $(\mathbf{B}_\ell)_{ij}$ is updated as:

$$
\tilde{q}((\mathbf{B}_\ell)_{ij}) = \frac{1}{Z} \tilde{f}((\mathbf{B}_\ell)_{ij}) q((\mathbf{B}_\ell)_{ij} \,|\, \mathbf{R}_{-ij\ell}^{(n-1)}) \ , \tag{9}
$$

where $\tilde{f}((\mathbf{B}_\ell)_{ij})$ is a factor[‖] related to $(\mathbf{B}_\ell)_{ij}$, either coming from a prior term or a likelihood term in (6), with Z the normalizer. In general, the updated posterior $\tilde{q}((\mathbf{B}_\ell)_{ij})$ does not have a simple parametric form. Thus it needs to be projected back to the proposed variational distribution $q((\mathbf{B}_\ell)_{ij} \,|\, \mathbf{R}_{-ij\ell}^{(n)})$. This is done by updating $q((\mathbf{B}_\ell)_{ij} \,|\, \mathbf{R}_{-ij\ell}^{(n)})$ such that it minimizes the KL divergence $\text{KL}\left[\tilde{q}((\mathbf{B}_\ell)_{ij}) \| q((\mathbf{B}_\ell)_{ij} \,|\, \mathbf{R}_{-ij\ell}^{(n)})\right]$.

In the following, we show how to update the global parameters $(\mathbf{B}, \mathbf{V}^{(1)}, \mathbf{V}^{(2)}, \gamma, \lambda, \phi, \psi)$ with ADF when the factor $\tilde{f}$ is chosen from the prior and the likelihood, respectively; we also show how to approximate the local parameters $\{\mathbf{z}_{n\ell}\}$, which is needed when $\tilde{f}$ is chosen from the likelihood for global parameter updates.

When $\tilde{f}$ is chosen from the prior We first derive update rules for $(\mathbf{B}, \mathbf{V}^{(1)}, \mathbf{V}^{(2)})$ with Gaussian variational distributions. The technique is similar to that in (Hernández-Lobato and Adams, 2015). Again we use $q((\mathbf{B}_\ell)_{ij} \,|\, \mathbf{R}_{-ij\ell}^{(n-1)})$ as an example, which is Gaussian with mean $m_{ij\ell}$ and variance $\lambda_{ij\ell}^{-1}$, as defined in (7). Based on (Minka, 2001), by minimizing the KL divergence between $q((\mathbf{B}_\ell)_{ij} \,|\, \mathbf{R}_{-ij\ell}^{(n-1)})$ and $\tilde{q}((\mathbf{B}_\ell)_{ij})$ defined above, we obtain the update equations for $m_{ij\ell}$ and

[§]Since now we have multiple training data, the local parameters (*e.g.*, $\mathbf{y}_n$) will be indexed by n in the following.

[¶]which means data is coming in a sequential way.

[‖]We have dropped out the dependency of the factor to other parameters for simplicity.

$\lambda_{ij\ell}$:

$$m_{ij\ell}^{(n)} = m_{ij\ell}^{(n-1)} + \lambda_{ij\ell}^{(n-1)} \nabla_{m_{ij\ell}} \log Z \tag{10}$$
$$\lambda_{ij\ell}^{(n)} = \lambda_{ij\ell}^{(n-1)} - (\lambda_{ij\ell}^{(n-1)})^2 \left[(\nabla_{m_{ij\ell}} \log Z)^2 - 2\nabla_{\lambda_{ij\ell}} \log Z \right]$$

where we have use the subscript "(n)" to denote the parameter after seeing up to n data points, and the normalizer Z can be computed approximately by using similar approach as in (Hernández-Lobato and Adams, 2015). Details and update equations for other parameters are provided in Appendix A.

For parameters $(\gamma, \lambda, \phi, \psi)$ with Gamma variational distributions, we must update the shape and rate parameters (α^x, β^x) in a Gamma distribution for $x \in \{\gamma, \lambda, \phi, \psi\}$. Following the supplementary material of Hernández-Lobato and Adams (2015), the update equations are:

$$\alpha_x^{(n)} = \left[Z Z_2 Z_1^{-2} \frac{\alpha_x^{(n-1)} + 1}{\alpha_x^{(n-1)}} - 1 \right]^{-1} \tag{11}$$
$$\beta_x^{(n)} = \left[Z_2 Z_1^{-1} \frac{\alpha_x^{(n-1)} + 1}{\beta_x^{(n-1)}} - Z_1 Z^{-1} \frac{\alpha_x^{(n-1)}}{\beta_x^{(n-1)}} \right] ,$$

where Z is the normalizer of $\tilde{q}(x)$, similar to (9); Z_1 and Z_2 are the values of the normalizer when α_x in the variational distribution is increased by 1 and 2, respectively.

When $\tilde{f}$ is chosen from the likelihood Again we use (10) and (11) to update the global Gaussian and Gamma parameters, as described above. When updating Gaussian distributions, the normalizer Z depends on the local parameter $\mathbf{z}_{nL}$. Specifically, Hernández-Lobato and Adams (2015) show that Z is approximated by (details shown in Appendix B):

$$Z \approx \mathcal{N}\left(\mathbf{y}_n; \mathbf{m}_{\mathbf{z}_{nL}}, \beta_\gamma/(\alpha_\gamma - 1) + \mathbf{v}_{\mathbf{z}_{nL}}\right) . \tag{12}$$

Consequently, the task is to estimate $\mathbf{m}_{\mathbf{z}_{nL}}$ and $\mathbf{v}_{\mathbf{z}_{nL}}$, the mean and variance vector for $\mathbf{y}_n$. Following (Hernández-Lobato and Adams, 2015), the idea is to start from the data layer, propagate the distributions forward through the neural network, and if necessary, approximate the distribution of a hidden layer with a Gaussian distribution. Note our technique here is different from that used in (Hernández-Lobato and Adams, 2015), because of the structured priors (MVG) instead of the unstructured priors (Gaussian) imposed on the weights.

Specifically, assume the output $\mathbf{z}_{n(\ell-1)}$ of layer $\ell-1$ is a diagonal Gaussian distribution with mean $\mathbf{m}_{\mathbf{z}_{n(\ell-1)}}$ and variance vector $\mathbf{v}_{\mathbf{z}_{n(\ell-1)}}$ defined in (8), our task is to calculate $\mathbf{m}_{\mathbf{z}_{n(\ell)}}$ and $\mathbf{v}_{\mathbf{z}_{n(\ell)}}$ for the next layer. From the definition, the output of layer ℓ is:

$$\mathbf{z}_{n\ell} = \text{ReLU}\left(\mathbf{P}_\ell \mathbf{B}_\ell \mathbf{Q}_\ell^T \mathbf{z}_{n(\ell-1)} / \sqrt{V_{\ell-1} + 1} \right) .$$

To approximate the distribution of $\mathbf{z}_{n\ell}$, we decompose $\mathbf{z}_{n\ell} = \text{ReLU}\left(\mathbf{z}_{n\ell}^{(3)}\right)$, where $\mathbf{z}_{n\ell}^{(3)} \triangleq \mathbf{P}_\ell \mathbf{z}_{n\ell}^{(2)}$, $\mathbf{z}_{n\ell}^{(2)} \triangleq \mathbf{B}_\ell \mathbf{z}_{n\ell}^{(1)} / \sqrt{V_{l-1} + 1}$, and $\mathbf{z}_{n\ell}^{(1)} \triangleq \mathbf{Q}_\ell^T \mathbf{z}_{n(\ell-1)}$. As detailed in Appendix B, we show that

$$\mathbf{z}_{n\ell}^{(1)} \sim \mathcal{N}\left(\mathbf{m}_{\mathbf{z}_{n\ell}}^{(1)}, \text{diag}(\mathbf{v}_{\mathbf{z}_{n\ell}}^{(1)})\right), \ \mathbf{z}_{n\ell}^{(2)} \sim \mathcal{N}\left(\mathbf{m}_{\mathbf{z}_{n\ell}}^{(2)}, \text{diag}(\mathbf{v}_{\mathbf{z}_{n\ell}}^{(2)})\right)$$
$$\mathbf{z}_{n\ell}^{(3)} \sim \mathcal{N}\left(\mathbf{m}_{\mathbf{z}_{n\ell}}^{(3)}, \text{diag}(\mathbf{v}_{\mathbf{z}_{n\ell}}^{(3)})\right) , \tag{13}$$

where $\mathbf{m}_{\mathbf{z}_{n\ell}}^{(1)} = \left(\mathbf{I}_{V_{\ell-1}} - 2\mathbf{E}\frac{\mathbf{v}_\ell^{(2)}(\mathbf{v}_\ell^{(2)})^T}{(\mathbf{v}_\ell^{(2)})^T \mathbf{v}_\ell^{(2)}}\right) \mathbf{m}_{\mathbf{z}_{n(\ell-1)}}$; $\mathbf{v}_{\mathbf{z}_{n\ell}}^{(1)} = \mathbf{v}_{\mathbf{z}_{n(\ell-1)}}$; $\mathbf{m}_{\mathbf{z}_{n\ell}}^{(2)} = \mathbf{M}_\ell \mathbf{m}_{\mathbf{z}_{n\ell}}^{(1)} / \sqrt{V_{\ell-1} + 1}$; $\mathbf{v}_{\mathbf{z}_{n\ell}}^{(2)} = \left[(\mathbf{M}_\ell \circ \mathbf{M}_\ell) \mathbf{v}_{\mathbf{z}_{n\ell}}^{(1)} + \Sigma_\ell^{(2)} (\mathbf{m}_{\mathbf{z}_{n\ell}}^{(1)} \circ \mathbf{m}_{\mathbf{z}_{n\ell}}^{(1)}) + \Sigma_\ell^{(2)} \mathbf{v}_{\mathbf{z}_{n\ell}}^{(1)} \right] / \sqrt{V_{\ell-1} + 1}$; $\mathbf{m}_{\mathbf{z}_{n\ell}}^{(3)} = \left(\mathbf{I}_{V_\ell} - 2\mathbf{E}\frac{\mathbf{v}_\ell^{(1)}(\mathbf{v}_\ell^{(1)})^T}{(\mathbf{v}_\ell^{(1)})^T \mathbf{v}_\ell^{(1)}}\right) \mathbf{m}_{\mathbf{z}_{n\ell}}^{(2)}$; $\mathbf{v}_{\mathbf{z}_{n\ell}}^{(3)} = \mathbf{v}_{\mathbf{z}_{n\ell}}^{(2)}$. Here $\mathbf{M}_\ell$ is a matrix such that $(\mathbf{M}_\ell)_{ij} \triangleq m_{ij\ell}$, defined in (7). The expectations in the above equations can be approximated by Monte Carlo integration, as detailed in Appendix B, efficiently computed by incorporating the Cholesky decomposition.

Finally, by applying the ReLU operator on $\mathbf{z}_{n\ell}^{(3)}$ and following Hernández-Lobato and Adams (2015), $\mathbf{z}_{n\ell}$ can be approximated by a Gaussian distribution, which is then propagated to the next layer. Details are provided in Appendix B.

4.2 Practical techniques

Computational complexity The computational cost of our model mainly relies on sampling Gaussian random variables, *e.g.*, sampling the normalizer Z in (12) to compute gradients with the reparameterization trick described in (Kingma and Welling, 2014), as well as sampling the hidden local parameters in (13) with Monte Carlo approximation. For the latter case, we adopt the Cholesky decomposition method (see Appendix B), which has complexity $O\left(D^3\right)$ with $D = \max_\ell V_\ell$. The naive implementation of the VMG model (without approximation when sampling an MVG distribution) (Louizos and Welling, 2016) shares the same computational cost as our algorithm. However, it is found in the experiments that our algorithm converges much faster than VMG for a given length of time while maintaining competitive predictive performance.

To speedup the sampling efficiency, we can either adopt the method in (Louizos and Welling, 2016) to approximate the covariance matrix of a Gaussian distribution with its diagonal elements, or treat $\{\mathbf{P}_\ell, \mathbf{Q}_\ell\}$ as hyperparameters (do not sample). These approaches lead to a reduced complexity of $O(D^2)$ but somewhat biased estimation.

Alternative learning methods In ADF, parameters are updated with every new data point, leading to potential underestimation of the variance. The problem can be alleviated by considering minibatches. Furthermore, the SEP method proposed in (Li et al., 2015) is also a good substitution for ADF, which is reserved for future work.

Extension for classification Our model can be naturally extended for classification by adding an additional softmax layer on top of $\mathbf{z}_{nL}$. The parameters in the lower layers are still updated with ADF. Given the input $\mathbf{z}_{nL}$'s for the softmax layer, the corresponding parameters can be learned by standard stochastic optimization algorithms, such as SGD and Adam (Kingma and Ba, 2015).

5 Related Work

The Laplace approximation has been proposed by MacKay (1992) and Bishop (2006) for learning of Bayesian neural networks. However, these methods have cubic computational cost with respect to the dataset size for computing the inversion of a Hessian matrix.

Hernández-Lobato and Adams (2015) propose to use ADF for the learning of a Bayesian neural network, forming the framework of probabilistic backpropagation. However, their work assumes independent Gaussian priors on the weights of a DNN. Nevertheless, their results show better prediction accuracies and faster learning speed compared to conventional variational inference (Graves, 2011) and backpropagation (Jylänki et al., 2014).

The variational Matrix Gaussian method (VMG) proposed by Louizos and Welling (2016) also introduces MVG priors in DNNs via the variational inference framework. Consequently they show better regression and classification performances than PBP and Dropout Bayesian neural network (Gal and Ghahramani, 2016). Although VMG establishes the connection to a deep Gaussian process and effectively avoids the high variance and memory requirements of simple variational inference, its convergence is found to be slow. While our work has the same computational complexity as VMG in each iteration, it converges faster in practice while maintaining competitive performance. Notably, with the reparameterization trick, instead of tackling a complex deep Gaussian process in VMG, our model can efficiently perform posterior inference like in a standard Bayesian neural network.

Ghosh et al. (2016) generalizes PBP (Hernández-Lobato and Adams, 2015) for classification and count regression by defining general output probability distributions. However, the priors for weights are still independent Gaussian, limiting modeling capacity. Note that similar techniques can be adaptive to our model for classification and count regression.

Another line of research for scalable Bayesian learning of a DNN employs stochastic gradient MCMC methods (SG-MCMC) (Welling and Teh, 2011; Ahn et al., 2012; Chen et al., 2014; Ding et al., 2014; Balan et al., 2015). However, the computational cost in testing is typically high because of the requirement to do model averaging. Our method mitigates this problem by adopting the ADF framework.

6 Experiments

Since our model is trained within the PBP framework, we denote it as PBP_MV. To verify the effectiveness of PBP_MV, we conduct several experiments on both synthetic and real datasets, and compare it with PBP (Hernández-Lobato and Adams, 2015) and VMG (Louizos and Welling, 2016), the-state-of-the-art method to model structured weight uncertainty with MVG priors. Specifically, we use two simple synthetic datasets to verify the estimation quality of different methods for nonlinear regression and classification, respectively; then a comparison of PBP_MV with PBP and VMG for regression is performed on 10 real datasets. In Appendix C.2, we show some preliminary results for classification on MNIST dataset. To compared with VMG, we use the code from the authors, which has been fairly well optimized for efficiency.

6.1 Synthetic experiments

For non-linear regression, we follow the experiment setup in the appendix of (Louizos and Welling, 2016). We randomly generate 12 data points from Uniform$(0, 0.6)$ and 8 from Uniform$(0.8, 1)$. The output y_n for input x_n is modeled as $y_n = x_n + \epsilon_n + \sin(4(x_n + \epsilon_n)) + \sin(13(x_n + \epsilon_n))$, where $\epsilon_n \sim \mathcal{N}(\epsilon_n; 0, 0.0009)$. We fit a neural network with two layers, each with 50 neurons. In PBP_MV, 100 samples are used to approximate the expectation in (13). Both PBP_MV and PBP are run for 120 epochs, while VMG is run for 1000 epochs because of its slow convergence speed. The results are plotted in 2, which clearly shows that PBP_MV obtains a better density estimation in that it captures the uncertainty within its variance; whereas other two methods fail to capture one side parameter uncertainty.

For classification, we follow the binary classification toy example in Ghosh et al. (2016), which uniformly samples ten $2D$ data points in two separable regions: $[-3, -1] \times [-3, -1]$ and $[1, 3] \times [1, 3]$, respectively. We use a network with one hidden layer of size 10 and a softmax layer on the top. We use 10 epochs for both

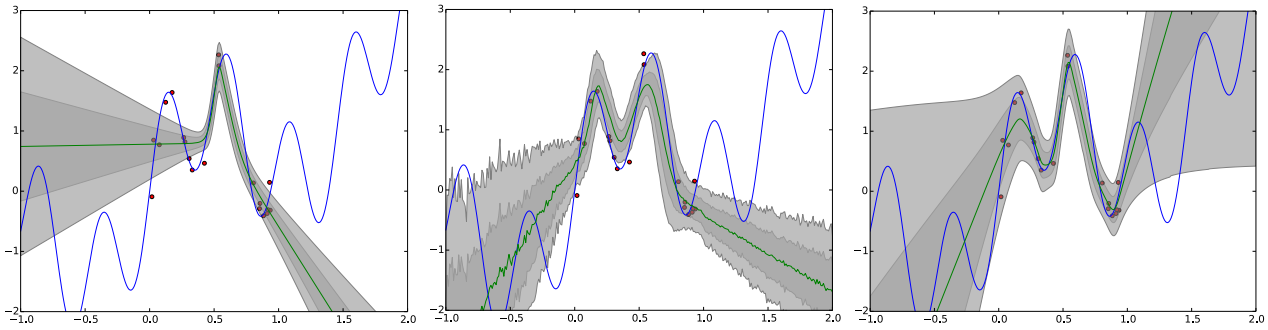

Figure 2: Comparison of PBP (left), VMG (middle) and PBP_MV (right) on a toy data for regression. The observation are shown as black dots. The blue line represents the true data generating function and the mean predictions are shown as green line. The light gray shaded area is the ± 3 standard derivation confidence interval.

PBP and PBP_MV, 80 epochs with 5 pseudo data and 5 batch size for VMG. Following Ghosh et al. (2016), the ground true is obtained by running the No-U-Turn sampler of Hoffman and Gelman (2014). The posterior classification density is plotted in 3, from which it is seen that compared to PBP and VMG, PBP_MV obtains a more accurate density estimation.

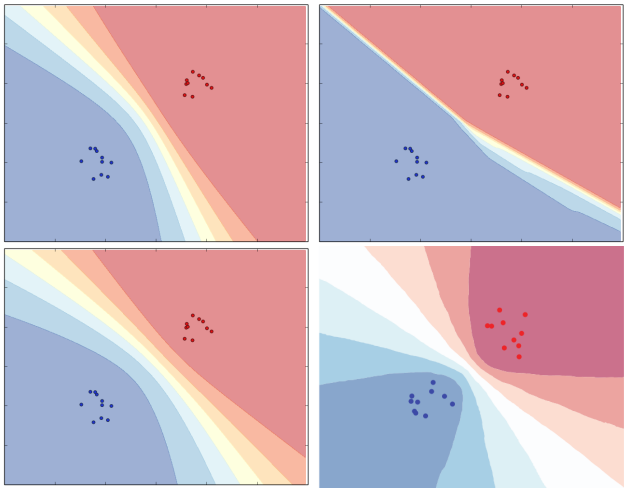

Figure 3: A synthetic binary classification experiment with PBP (top-left), VMG (top-right), PBP_MV (bottom-left) and ground true (bottom right). The blue and red dots are data points for training. The posterior density varies with colors, where a deeper color means more confident to belong to some class.

6.2 Bayesian DNN for nonlinear regression

Following PBP, we preform nonlinear regression on ten publicly available datasets, with names listed in Table 1. We use a network with two hidden layers. For the two big datasets, *YearPredict* and *Protein*, the number of neurons in both layers is set to 100, while it is set to 50 for the other 8 small datasets. Since PBP has shown superior performance to variational inference (VI) and backpropagation (BP) Hernández-Lobato and Adams (2015), but is generally worse than VMG (Louizos and Welling, 2016), it suffices to compare our model with VMG.

We randomly split the datasets into training and test sets, with 90% of the data for training and the remaining for testing. We repeat for 20 times to calculate the average predictive performance on the 8 small datasets, 5 times for the larger *YearPredict* dataset, and once for the largest dataset *Protein*. As data magnitudes differ significantly, we normalize each dataset so that the training data has zero mean and unit variance. All models are run until convergence**, which roughly results in 70 epochs for VMG and 50 epochs for PBP_MV on the small datasets: *Boston_Housing*, *Concrete*, *Winequality* and *Yacht*; On the slightly larger datasets, *Energy*, *Kin8nm*, *Naval* and *CCPP*, 180 epochs are used for both methods; for the two biggest datasets, *Protein* and *YearPredict*, we run long enough until converged. The expectations in (13) are evaluated based on 100 and 300 samples for training and testing, respectively.

Following Hernández-Lobato and Adams (2015), we use root mean squared error (RMSE) and test log-likelihood to evaluate model performance. In addition, to show the efficiency of our model compared to VMG, we also report the total running time as a metric. Because the VMG code from the authors has a big data overhead, it runs much slower in GPU model than in CPU mode, we thus report running time for both methods based on the CPU mode. Results are reported in Table 1, from which we find that in most case, our model outperforms VMG, especially in terms of test log-likelihood. Notably, we observe that our model obtains a much faster total running time than VMG (more precise results on convergence speed will be shown below). In addition, we note that in

**Though there are no theory to guarantee the convergence of PBP_MV, it usually does converge in practice.

	Test RMSE		Test Log likelihood		Avg. Time (second)	
Dataset	VMG	PBP_MV	VMG	PBP_MV	VMG	PBP_MV
Boston_Housing	3.18±0.19	**3.11**±0.15	−2.71±0.12	**−2.54**±0.08	633	**109**
Concrete	5.18±0.16	**5.08**±0.14	−3.07±0.04	**−3.04**±0.03	1287	**92**
Energy	0.48±0.01	**0.45**±0.01	**−0.91**±0.01	−1.01±0.01	934	**620**
Kin8nm	**0.07**±0.00	**0.07**±0.00	1.24±0.00	**1.28**±0.01	8461	**2277**
Naval	**0.00**±0.00	**0.00**±0.00	2.47±0.00	**4.85**±0.06	12457	**3434**
CCPP	**3.87**±0.05	3.91±0.04	**−2.78**±0.01	**−2.78**±0.01	8131	**2711**
Protein	**3.90**±0.02	3.94±0.02	−2.78±0.01	**−2.77**±0.01	45119	**27846**
Winequality	**0.64**±0.01	**0.64**±0.01	−0.99±0.02	**−0.97**±0.01	1713	**280**
Yacht	0.87±0.08	**0.81**±0.06	**−1.46**±0.02	−1.64±0.02	332	**79**
YearPredict	**8.64**±NA	8.72±NA	−3.57±NA	**−3.33**±NA	71007	**42596**

Table 1: Averaged predictions with standard deviations in terms of RMSE, log-likelihood and running time on test sets.

PBP_MV, the hyperparameters are updated in close forms, bypassing the need for cross validation.

To compare the convergence speed, we further plot the learning curves of *RMSE/log-likelihood vs time* in Figure 4 on *Concrete*, *Kin8nm*, *Yacht* and *Energy* datasets. Plots for other datasets are provide in Appendix C. It is clear from the figure that in most cases PBP_MV converges much faster than VMG. One exception is the *likelihood vs time* on the *Energy* dataset, where PBP_MV seems to be slower at the beginning, but quickly surpasses VMG to reach a better likelihood. For VMG, the test performance varies more significantly along time, making it hard to diagnose the convergence behavior. Consequently, it would lead to a higher estimation variance.

7 Conclusion

We introduce the MVG prior to model structured weight uncertainty in Bayesian neural networks. By leveraging a reparameterization technique for the MVG distribution, the MVG-based DNN model is transformed to a simple DNN with isotropic Gaussian priors on the weights. Consequently, an online learning algorithm based on the PBP framework is readily derived. Experiments on several synthetic and real datasets demonstrate the superiority of our method, obtaining competitive predictive performance yet faster convergence speed, compared to other Bayesian DNN models.

There are several future research directions. One is to extend the proposed Bayesian DNN with convolutional layers to model more complex data such as images. Another direction is to improve the current online learning algorithms, possibly by stochastic EP (Li et al., 2015). Finally, incorporating MVG into the variational auto-encoder framework (Kingma and Welling, 2014) for more flexible modeling is also interesting.

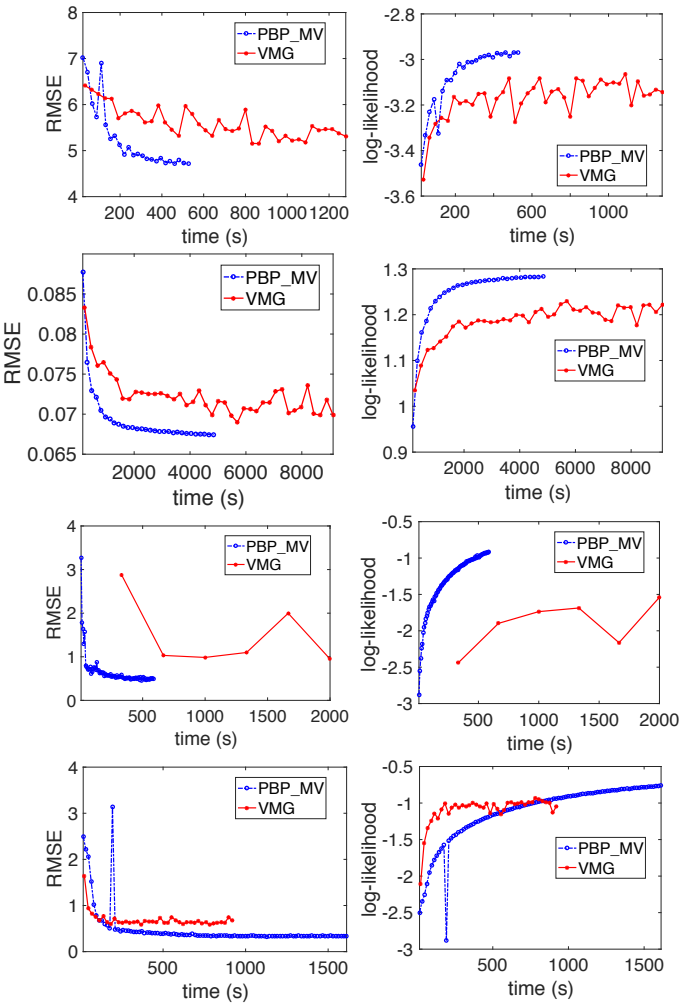

Figure 4: Example learning curves in terms of *RMSE vs running time* (left column) and *log-likelihood vs. running time* (right column) on 4 datasets: *Concrete* (1st row), *Kin8nm* (2nd row), *Yacht* (3rd row) and *Energy* (4th row).

Acknowledgements

This research was supported in part by ARO, DARPA, DOE, NGA, ONR and NSF.

References

S. Ahn, A. K. Balan, and M. Welling. Bayesian posterior sampling via stochastic gradient Fisher scoring. In *ICML*, 2012.

A. K. Balan, V. Rathod, K. P. Murphy, and M. Welling. Bayesian dark knowledge. In *NIPS*, 2015.

C. M. Bishop. Pattern recognition. *Machine Learning*, 128, 2006.

C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural networks. In *ICML*, 2015.

L. Bottou, editor. *Online algorithms and stochastic approximations*. Cambridge University Press, 1998.

T. Chen, E. B. Fox, and C. Guestrin. Stochastic gradient Hamiltonian Monte Carlo. In *ICML*, 2014.

N. Ding, Y. Fang, R. Babbush, C. Chen, R. D. Skeel, and H. Neven. Bayesian sampling using stochastic gradient thermostats. In *NIPS*, 2014.

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

S. Ghosh, F. M. Delle Fave, and J. Yedidia. Assumed density filtering methods for learning bayesian neural networks. In *AAAI*, 2016.

X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *AISTATS*, 2011.

G. H. Golub and C. F. Van Loan, editors. *Matrix Computations*. Johns Hopkins University Press, 1996.

A. Graves. Practical variational inference for neural networks. In *NIPS*, 2011.

A. K. Gupta and D. K. Nagar. *Matrix Variate Distributions*. CRC Press, 1999.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

H. V. Henderson and S. R. Searle. The vec-permutation matrix, the vec operator and kronecker products: A review. *Linear and Multilinear Algebra*, 9(4):271–288, 1980.

J. M. Hernández-Lobato and R. P. Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, 2015.

M. D. Hoffman and A. Gelman. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15(1):1593–1623, 2014.

P. Jylänki, A. Nummenmaa, and A. Vehtari. Expectation propagation for neural networks with sparsity-promoting priors. *Journal of Machine Learning Research*, 2014.

D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

D. Kingma and M. Welling. Auto-encoding variational Bayes. In *ICLR*, 2014.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

C. Li, C. Chen, D. Carlson, and L. Carin. Preconditioned stochastic gradient Langevin dynamics for deep neural networks. In *AAAI*, 2016.

Y. Li, J. M. Hernández-Lobato, and R. E. Turner. Stochastic expectation propagation. In *NIPS*, 2015.

C. Louizos and M. Welling. Structured and efficient variational deep learning with matrix gaussian posteriors. *arXiv preprint arXiv:1603.04733*, 2016.

X. Lu, V. Perrone, L. Hasenclever, Y. W. Teh, and S. J. Vollmer. Relativistic Monte Carlo. In *arXiv:1609.04388*, 2016.

D. J. C. MacKay. A practical bayesian framework for backpropagaion networks. *Neural Computation*, 4: 448–472, 1992.

T. P. Minka. A family of algorithms for approximate bayesian inference. Technical report, MIT, USA, 2001.

M. Opper. *A Bayesian approach to on-line learning*. On-line learning in neural networks, Cambridge University Pres, 1998.

Y. Pu, Z. Gan, R. Henao, X. Yuan, C. Li, A. Stevens, and L. Carin. Variational autoencoder for deep learning of images, labels and captions. In *NIPS*, 2016.

D. Silver et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

Q. Su, X. Liao, C. Chen, and L. Carin. Nonlinear statistical learning with truncated gaussian graphical models. In *ICML*, 2016.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *ICML*, 2011.

Y. Wu et al. Google's neural machine translation system: Bridging the gap between human and machine translation. In *arXiv:1609.08144*, 2016.

K. Xu, J. L. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, 2015.