Journal of Machine Learning Research 22 (2021) 1-23 Submitted 9/20; Revised 6/21; Published 9/21

Tractable Approximate Gaussian Inference for Bayesian Neural Networks

James-A. Goulet* JAMES.GOULET@POLYMTL.CA
Luong Ha Nguyen* LUONG-HA.NGUYEN@POLYMTL.CA
Saeid Amiri
Department of Civil Engineering, Polytechnique Montréal, Montréal, Canada

Editor: Shakir Mohamed

Abstract

In this paper, we propose an analytical method for performing *tractable approximate Gaussian inference* (TAGI) in Bayesian neural networks. The method enables the analytical Gaussian inference of the posterior mean vector and diagonal covariance matrix for weights and biases. The method proposed has a computational complexity of $\mathcal{O}(n)$ with respect to the number of parameters n, and the tests performed on regression and classification benchmarks confirm that, for a same network architecture, it matches the performance of existing methods relying on gradient backpropagation.

Keywords: Bayesian, Neural Networks, TAGI, Gaussian Inference, Approximate Inference, Gaussian multiplicative approximation

1. Introduction

The estimation of weight and bias in neural networks is currently dominated by approaches employing point estimates when learning model parameters using gradient backpropagation (Rumelhart et al., 1986). Although these approaches allow for a state-of-the-art performance in many domains of applications, it is recognized that they fall short in situations where, for instance, datasets are small, when the task requires quantifying the uncertainty about the prediction made, and for continual learning (Ghahramani, 2015; Kendall and Gal, 2017; Farquhar and Gal, 2019). In general, the Bayesian approach for inferring the parameters' posterior is known to be theoretically better suited than a point estimate (Goodfellow et al., 2016; Murphy, 2012). This is in theory because applying exact Bayesian inference on large neural networks has been considered to be intractable (Goodfellow et al., 2016).

In this paper, we propose a *tractable approximate Gaussian inference* method (TAGI) for Bayesian neural networks. The approach, which does not rely on backpropagation, allows for an analytical treatment of uncertainty for the weight and bias parameters. TAGI relies on two key steps: The first one consists in the analytical forward uncertainty propagation through the network using moment generating functions and a local linearization of activation functions. This step allows computing the expected values, covariances, and cross covariances required for performing analytical inference. The second step leverages the Gaussian assumptions throughout the network in order to perform the analytical Gaussian inference for both the hidden units and parameters. The key to the computational tractability of TAGI is to take

∗. The first and second authors contributed equally to this work

advantage of the inherent conditional independence embedded in neural network in order to perform both the forward propagation of uncertainty as well as the Gaussian inference in a recursive layer-wise manner. This allows reaching a storage as well as the computational complexity that is linear with respect to the number of weights in the network.

The paper is organized as follows: Section 2 first introduces the *Gaussian multiplication approximation* (GMA) for propagating uncertainty in feedforward neural networks, and second, it presents how to perform tractable Gaussian inference for the posterior mean vector and diagonal covariance for the weight and bias parameters. Section 3, positions TAGI in the context of the related work which has already tackled the problem of approximate inference in Bayesian neural networks. Finally, Section 4 validates the performance of the approach on benchmark regression problems and on the MNIST classification problem.

2. Gaussian Approximation for BNN

This section first introduces the nomenclature for Gaussian feedforward neural network, then we present how to use matrix operations in order to propagate uncertainty through it, and finally, we present how to perform tractable approximate Gaussian inference.

2.1 Gaussian Feedforward Neural Network

Let us consider a vector of input covariates $\boldsymbol{X} = [X_1\ X_2\ \dots\ X_{\mathtt{X}}]^\intercal$ such that $\boldsymbol{x} \in \mathbb{R}^{\mathtt{X}}$ that are described by random variables in order to take into account errors potentially arising from observation uncertainties, and then suppose we have a vector of $\mathtt{Y}$ observed system responses $\boldsymbol{Y} = [Y_1\ Y_2\ \dots\ Y_{\mathtt{Y}}]^\intercal$ such that $\boldsymbol{y} \in \mathbb{R}^{\mathtt{Y}}$. Note that throughout the paper, the number of variables in vectors or sets are denoted by typewriter typefaces such as $\mathtt{X}$ and $\mathtt{Y}$. The details of the nomenclature for the feedforward neural network (FNN) employed in this paper is presented in Figure 9 from Appendix A. The relations between observed system responses and its covariates are described by the observation model

$$\boldsymbol{y} = \boldsymbol{z}^{(\mathtt{0})} + \boldsymbol{v}, \tag{1}$$

where the vector of hidden variables $\boldsymbol{z}^{(\mathtt{0})}$ corresponds to the output layer of a neural network on which observation errors $\boldsymbol{v}$ are added such that $\boldsymbol{V} \sim \mathcal{N}(\boldsymbol{v}; \boldsymbol{0}, \boldsymbol{\Sigma}_{\boldsymbol{V}})$. In common cases, $\boldsymbol{\Sigma}_{\boldsymbol{V}}$ is a diagonal covariance matrix assuming that observation errors are independent from each other, and further, we assume that $\boldsymbol{\Sigma}_{\boldsymbol{V}}$ is independent of $\boldsymbol{x}$. The treatment heteroschedastic problems with a closed-form analytical inference for $\sigma_V(x)$ is outside the scope of this paper and is not further treated. We can model the relations between covariates $\boldsymbol{x}$ and output hidden variables $\boldsymbol{z}^{(\mathtt{0})}$ using a feedforward neural network consisting of $\mathtt{L}$ hidden layers each having $\mathtt{A}$ activation units $a_i^{(j)}$ and hidden variables $z_i^{(j)}$, $\forall i = \{1, 2, \cdots, \mathtt{A}\}$, where an activation unit $a_i^{(j)}$ is a non-linear transformation of its associated hidden variable, $a_i^{(j)} = \sigma(z_i^{(j)})$. Note that for the sake of simplifying the notation and explanations, throughout the paper, we consider that all hidden layers have the same number of units $\mathtt{A}$. We go from the input layer containing the covariates $\boldsymbol{x}$, to the i^{th} hidden variable on the first hidden layer, using an affine function of $\boldsymbol{x}$ so that

$$z_i^{(1)} = w_{i,1}^{(\mathtt{0})} x_1 + w_{i,2}^{(\mathtt{0})} x_2 + \cdots + w_{i,\mathtt{X}}^{(\mathtt{0})} x_{\mathtt{X}} + b_i^{(\mathtt{0})}. \tag{2}$$

Equation 2 involves the product of weights $w_{i,j}^{(\emptyset)}$ and covariates x_j with an additive bias term $b_i^{(\emptyset)}$. In the context of a neural network, the process of learning consists in estimating these weights and biases. Here, we consider that weight and bias parameters are described by random variables so that our joint prior for $\{\boldsymbol{X}, \boldsymbol{W}^{(\emptyset)}, \boldsymbol{B}^{(\emptyset)}\}$ is a multivariate Gaussian. Although we know this distribution assumption may not be suited in all situations, we restrict ourself to the Gaussian setup.

In Equation 2, we note that the product of Gaussian random variables is not Gaussian; Despite this, we propose to employ moment generating functions in order to compute analytically its expected value, its variance, as well as the covariance between the product of Gaussian random variables and any other Gaussian random variable.

For instance, let $\boldsymbol{X} = [X_1 \ \dots \ X_4]^\intercal \sim \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ be a generic vector of Gaussian random variables, where $\boldsymbol{\mu}$ is the mean vector and $\boldsymbol{\Sigma}$ is the covariance matrix, then the following statements hold,

$$\mathbb{E}[X_1 X_2] = \mu_1\mu_2 + \text{cov}(X_1, X_2), \tag{3}$$

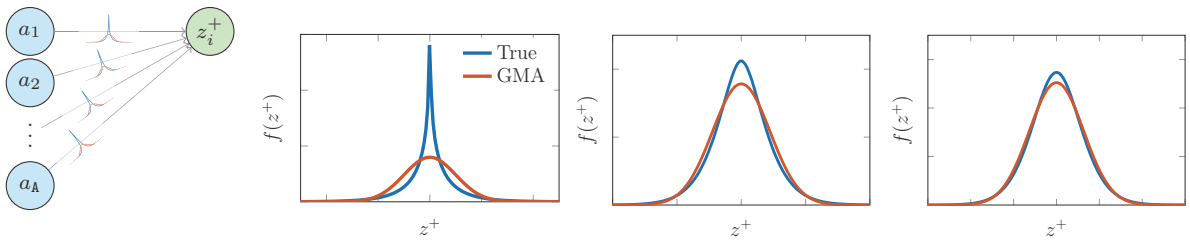
$$\text{cov}(X_3, X_1X_2) = \text{cov}(X_1,X_3)\mu_2 + \text{cov}(X_2,X_3)\mu_1, \tag{4}$$

$$\begin{aligned}\text{cov}(X_1X_2, X_3X_4) = &\text{cov}(X_1, X_3)\text{cov}(X_2, X_4) + \text{cov}(X_1, X_4)\text{cov}(X_2, X_3) \\ &+\text{cov}(X_1, X_3)\mu_2\mu_4 + \text{cov}(X_1, X_4)\mu_2\mu_3 \\ &+\text{cov}(X_2, X_3)\mu_1\mu_4 + \text{cov}(X_2, X_4)\mu_1\mu_3,\end{aligned} \tag{5}$$

$$\text{var}(X_1X_2) = \sigma_1^2\sigma_2^2 + \text{cov}(X_1, X_2)^2 + 2\text{cov}(X_1, X_2)\mu_1\mu_2 + \sigma_1^2\mu_2^2 + \sigma_2^2\mu_1^2. \tag{6}$$

The development of statements (3–6) is presented in the Appendix B. In this paper, we define the *Gaussian multiplication approximation* as the approximation of the probability density function (PDF) for any product term X_iX_j by a Gaussian whose first two moments are defined by Equations (3–6). With this approximation, we can now employ X_iX_j along with the random state vector $\boldsymbol{X}$ in affine functions. This allows propagating the uncertainty from the input covariates and prior knowledge on weight and bias parameter through a FNN.

Figure 1a illustrates the passage from the activation units $\boldsymbol{A}$, to a subsequent hidden unit $\boldsymbol{Z}_i^+$. Figures 1b–d compare the true theoretical PDFs with those obtained using the GMA for different numbers of activation units $\mathtt{A}$, under the assumption that both $A_k \sim \mathcal{N}(a_k; 0, 1)$, $W_{i,k} \sim \mathcal{N}(w_{i,k}; 0, 1)$ and $b_i = 0$. This shows that even if the GMA is a crude approximation for the true PDF resulting from the product of two Gaussians, when several of these



(a) $\boldsymbol{a} \to z_1^+$ (b) $\mathtt{A} = 1$ (c) $\mathtt{A} = 5$ (d) $\mathtt{A} = 10$

Figure 1: Illustration of the effect of the GMA on the PDF of a hidden unit Z_i^+ as a function of the number of activation units $\mathtt{A}$ on the preceding hidden layer. The blue curve represents the true PDF estimated using sampling and the red one, the PDF resulting from the GMA.

independent product terms are added, the result quickly tends to a Gaussian-like PDF. In order for the central limit theorem (CLT) to apply, all activation units A_k must be independent. We can demonstrate the validity of this hypothesis by revisiting the same example while computing the correlation between pairs of hidden units $\{Z_i^+, Z_j^+\}$ on the subsequent hidden layer. With this example, we want to show that starting from a hidden layer having $\mathtt{A}$ independent activation units, pairs of hidden units on the subsequent layer are also independent if $\mathtt{A}$ is sufficiently large, despite the fact that the hidden units $\{Z_i^+, Z_j^+\}$ depend on the same activation units on the previous layer. Figure 2 presents the expected value, and the one standard deviation confidence region for $\rho(Z_i^+, Z_j^+)$, obtained from 1000 weight value realizations. The wide confidence region associated with small numbers of units, e.g. $\mathtt{A} < 10$, confirms that for most realizations, the dependence on the same activation units will introduce a strong correlation between the hidden units of the subsequent layer. On the other hand, for larger $\mathtt{A}$ values, the narrow confidence region centred around 0 confirms that for any sets of weight realizations, the hidden units on the subsequent layer remain independent. This shows that starting from independent activation units, any pairs of hidden units on the following layer are also approximately independent. Because neural networks employ the same operations recursively over multiple layers, this independence assumption for the hidden units remains valid throughout the network. The theoretical example presented here is a simplified version of a real neural network as weights are randomly sampled; For real neural networks, Wu et al. (2019) have confirmed empirically that some form of CLT hold for the hidden units during training as the independence hypothesis remained approximately valid between layers. In Section 4.1 we further demonstrate on a real problem that if $\mathtt{A}$ is large enough, the approximate independence of the hidden units within layers remains valid.

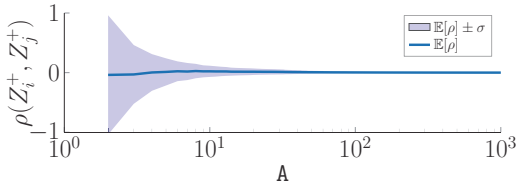


Figure 2: Evolution of the correlation $\rho(Z_i^+, Z_j^+)$ as a function of the number of activation units $\mathtt{A}$ on the preceding layer. The expected value and the confidence region corresponding to one standard deviation are presented for 1000 random realizations of weight values.

The passage from the hidden variables $\boldsymbol{Z}$, to their corresponding activation units $\boldsymbol{A}$ cannot be done analytically using non-linear activation functions. In order to work around this difficulty, we propose to employ functions that are locally linearized at $\mathbb{E}[\boldsymbol{Z}] = \boldsymbol{\mu_Z}$, analogously to what is done for the extended Kalman filter (Haykin, 2004). *Locally linearized activation functions* $\tilde{\sigma}(\cdot)$ allow calculating analytically the expected vector $\mathbb{E}[\boldsymbol{A}]$, the covariance $\mathrm{cov}(\boldsymbol{A})$, as well as the cross-covariance between activation units and the weight and bias $\mathrm{cov}(\boldsymbol{A}, \boldsymbol{\theta})$, where $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{B}\}$. Figure 3 presents an example for the linearization of a softplus activation function. Note that a locally linearized activation function is not equivalent to having a linear activation function because for each instance of input covariates $\boldsymbol{x}_i$, the linearization is done at a different value μ_z, which maintains the non-linear dependency between the input $\boldsymbol{x}_i$ and the output $\boldsymbol{y}_i$. This linearization procedure is an approximation of the *change of variable rule* (Murphy, 2012, §2.6.2) that would be required to obtain the true theoretical PDF for

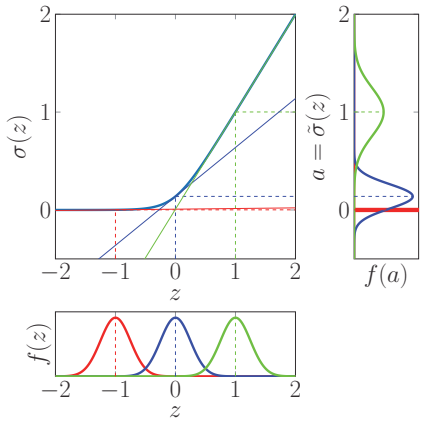


Figure 3: Examples of local linearization for a softplus activation function $\sigma(\cdot)$ at the expected value $\mathbb{E}[Z]$ for $Z \sim \mathcal{N}(z; \mu_Z, 0.25^2)$ where $\mu_Z = \{-1, 0, 1\}$.

the output $f(a)$ from the input $f(z)$. Although it is known that this approach may lead to poor approximation (Wan and Merwe, 2000) when there is a non-linearity in a region high probability density, we choose to employ this linearization procedure because of its minimal computational cost which still allows, as it will be shown in Section 4, matching the state-of-the-art performance on equivalent neural networks architectures using backpropagation. Finally, note that the linearization procedure is compatible with all the common activation functions such as the ReLU, tanh, logistic sigmoid, etc.

The transition from the knowledge of the j^{th} layer's activation units to an activation unit on the $j+1$ layer is defined by $a_i^{(j+1)} = \tilde{\sigma}(z_i^{(j+1)}) = \tilde{\sigma}(w_{i,1}^{(j)} a_1^{(j)} + w_{i,2}^{(j)} a_2^{(j)} + \cdots + w_{i,\mathtt{A}}^{(j)} a_{\mathtt{A}}^{(j)} + b_i^{(j)})$. Analogously, we go from the last hidden layer to the output layer by following

$$z_i^{(\mathtt{O})} = w_{i,1}^{(\mathtt{L})} a_1^{(\mathtt{L})} + w_{i,2}^{(\mathtt{L})} a_2^{(\mathtt{L})} + \cdots + w_{i,\mathtt{A}}^{(\mathtt{L})} a_{\mathtt{A}}^{(\mathtt{L})} + b_i^{(\mathtt{L})},$$

for all $i \in \{1, 2, \cdots, \mathtt{Y}\}$. All these steps define what we call the *approximate Gaussian feedforward neural network* (AG-FNN), which can be summarized by the input-output relation

$$\left\{ \boldsymbol{\mu}_{\boldsymbol{Z}}^{(\mathtt{O})}, \boldsymbol{\Sigma}_{\boldsymbol{Z}}^{(\mathtt{O})}, \boldsymbol{\Sigma}_{\boldsymbol{Z}\boldsymbol{\theta}}^{(\mathtt{O})} \right\} = \text{AG-FNN}\left(\boldsymbol{\mu}_{\boldsymbol{X}}, \boldsymbol{\Sigma}_{\boldsymbol{X}}, \boldsymbol{\mu}_{\boldsymbol{\theta}}, \boldsymbol{\Sigma}_{\boldsymbol{\theta}} \right). \tag{7}$$

For the *regression setup* where $\boldsymbol{y} \in \mathbb{R}^{\mathtt{Y}}$, the observed model output is directly defined by the last layer as described in Equation 1 so that $\mathcal{N}(\boldsymbol{y}; \boldsymbol{\mu}_{\boldsymbol{Y}}, \boldsymbol{\Sigma}_{\boldsymbol{Y}})$, where, $\boldsymbol{\mu}_{\boldsymbol{Y}} = \boldsymbol{\mu}_{\boldsymbol{Z}}^{(\mathtt{O})}$, and $\boldsymbol{\Sigma}_{\boldsymbol{Y}} = \boldsymbol{\Sigma}_{\boldsymbol{Z}}^{(\mathtt{O})} + \boldsymbol{\Sigma}_{\boldsymbol{V}}$.

For the *classification setup*, we need to convert the output $\boldsymbol{y} \in \mathbb{R}^{\mathtt{Y}}$ into a class observation $y^{(\mathtt{C})} \in \{1, 2, \cdots, \mathtt{K}\}$. Note that using the traditional *Softmax* output layer would not allow for a closed-form solution for propagating and marginalizing the uncertainty associated with the output $\boldsymbol{Y}$. Instead, we propose to employ a *hierarchical binary decomposition* similar to what was employed by Morin and Bengio (2005). The details regarding this formulation are presented in Appendix C.

2.2 Matrix operations for AG-FNN

In this section, we describe how the steps involved in the evaluation of Equation 7 can be performed using matrix operations. Our first hypothesis supposes that the knowledge for covariates, hidden units, as well as the weights and biases, is described by Gaussian random variables. We can then generalize the operations for going from the $\mathtt{A}$ activation units at a layer j to the subsequent $\mathtt{A}$ hidden units at layer $j+1$ using matrix operations so that

$$\underbrace{\begin{bmatrix} Z_1^{(j+1)} \\ Z_2^{(j+1)} \\ \vdots \\ Z_{\mathtt{A}}^{(j+1)} \end{bmatrix}}_{\boldsymbol{Z}^{(j+1)}} = \underbrace{\begin{bmatrix} W_{1,1}^{(j)} & W_{1,2}^{(j)} & \cdots & W_{1,\mathtt{A}}^{(j)} \\ W_{2,1}^{(j)} & W_{2,2}^{(j)} & \cdots & W_{2,\mathtt{A}}^{(j)} \\ \vdots & \vdots & \ddots & \vdots \\ W_{\mathtt{A},1}^{(j)} & W_{\mathtt{A},2}^{(j)} & \cdots & W_{\mathtt{A},\mathtt{A}}^{(j)} \end{bmatrix}}_{\boldsymbol{W}^{(j)}} \times \underbrace{\begin{bmatrix} A_1^{(j)} \\ A_2^{(j)} \\ \vdots \\ A_{\mathtt{A}}^{(j)} \end{bmatrix}}_{\boldsymbol{A}^{(j)}} + \underbrace{\begin{bmatrix} B_1^{(j)} \\ B_2^{(j)} \\ \vdots \\ B_{\mathtt{A}}^{(j)} \end{bmatrix}}_{\boldsymbol{B}^{(j)}}. \tag{8}$$

Our prior knowledge for activation units is described by $\boldsymbol{A}^{(j)} \sim \mathcal{N}(\boldsymbol{a}^{(j)}; \boldsymbol{\mu}_{\boldsymbol{A}}^{(j)}, \boldsymbol{\Sigma}_{\boldsymbol{A}}^{(j)})$ as well as by the cross-covariance $\mathrm{cov}(\boldsymbol{\theta}, \boldsymbol{A}^{(j)})$ between the activation units and the vector $\boldsymbol{\theta} \in \mathbb{R}^{\mathtt{P}}$ containing all the weight and bias parameters defined for all the layers in the network. We can re-write Equation 8 by breaking down the matrix-vector product $\boldsymbol{W} \times \boldsymbol{A}$, into an operation-wise equivalent vector $(\boldsymbol{WA}) \in \mathbb{R}^{\mathtt{A}^2 \times 1}$ for which the moments of the product terms can be pre-computed; We can employ Equation 3 in order to compute the expected vector $\boldsymbol{\mu}_{\boldsymbol{WA}}^{(j)} \equiv \mathbb{E}[(\boldsymbol{WA})^{(j)}]$, Equation 5–6 for the covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{WA}}^{(j)} \equiv \mathrm{cov}\left((\boldsymbol{WA})^{(j)}\right) \in \mathbb{R}^{\mathtt{A}^2 \times \mathtt{A}^2}$ associated with the product terms $(\boldsymbol{WA})^{(j)} \in \mathbb{R}^{\mathtt{A}^2}$, and Equation 4 for the cross-covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{WA\theta}}^{(j)} \equiv \mathrm{cov}\left((\boldsymbol{WA})^{(j)}, \boldsymbol{\theta}\right) \in \mathbb{R}^{\mathtt{A}^2 \times \mathtt{P}}$. We then introduce two new deterministic matrices $\mathbf{F}_{\boldsymbol{wa}}^{(j)} \in \{0,1\}^{\mathtt{A} \times \mathtt{A}^2}$ and $\mathbf{F}_{\boldsymbol{b}}^{(j)} \in \{0,1\}^{\mathtt{A} \times \mathtt{A}}$, which allow rewriting Equation 8 as a system of linear equations involving the product-terms vector $(\boldsymbol{WA})$,

$$\boldsymbol{Z}^{(j+1)} = \mathbf{F}_{\boldsymbol{wa}}^{(j)} (\boldsymbol{WA})^{(j)} + \mathbf{F}_{\boldsymbol{b}}^{(j)} \boldsymbol{B}^{(j)}. \tag{9}$$

Note that $\mathbf{F}_{\boldsymbol{wa}}^{(j)}$ and $\mathbf{F}_{\boldsymbol{b}}^{(j)}$ are non-unique as their specific definition depend on the ordering of variables in the problem. An example of structure for these matrices is presented in Appendix D. Using the properties for linear functions of Gaussian random variables, we obtain

$$\begin{aligned} \boldsymbol{\mu}_{\boldsymbol{Z}}^{(j+1)} &\equiv \quad \mathbb{E}[\boldsymbol{Z}^{(j+1)}] = \mathbf{F}_{\boldsymbol{wa}}^{(j)} \boldsymbol{\mu}_{\boldsymbol{WA}}^{(j)} + \mathbf{F}_{\boldsymbol{b}}^{(j)} \boldsymbol{\mu}_{\boldsymbol{B}}^{(j)}, \\ \boldsymbol{\Sigma}_{\boldsymbol{Z}}^{(j+1)} &\equiv \quad \mathrm{cov}(\boldsymbol{Z}^{(j+1)}) = \mathbf{F}_{\boldsymbol{wa}}^{(j)} \boldsymbol{\Sigma}_{\boldsymbol{WA}}^{(j)} \mathbf{F}_{\boldsymbol{wa}}^{(j)\intercal} + \mathbf{F}_{\boldsymbol{b}}^{(j)} \boldsymbol{\Sigma}_{\boldsymbol{B}}^{(j)} \mathbf{F}_{\boldsymbol{b}}^{(j)\intercal} + 2\mathbf{F}_{\boldsymbol{wa}}^{(j)} \mathrm{cov}(\boldsymbol{WA}^{(j)}, \boldsymbol{B}^{(j)}) \mathbf{F}_{\boldsymbol{b}}^{(j)\intercal}, \\ \boldsymbol{\Sigma}_{\boldsymbol{Z\theta}}^{(j+1)} &\equiv \mathrm{cov}(\boldsymbol{Z}^{(j+1)}, \boldsymbol{\theta}) = \mathbf{F}_{\boldsymbol{wa}}^{(j)} \boldsymbol{\Sigma}_{\boldsymbol{WA\theta}}^{(j)} + \boldsymbol{\Sigma}_{\boldsymbol{B\theta}}^{(j)}, \end{aligned} \tag{10}$$

where $\boldsymbol{\Sigma}_{\boldsymbol{B\theta}}^{(j)} \equiv \mathrm{cov}(\boldsymbol{B}^{(j)}, \boldsymbol{\theta}) \in \mathbb{R}^{\mathtt{A} \times \mathtt{P}}$ is the covariance between the bias parameters from the j^{th} layer and all the other parameters. In order to apply the locally linearized activation function $\boldsymbol{A}^{(j+1)} = \tilde{\sigma}(\boldsymbol{Z}^{(j+1)})$,

$$\boldsymbol{A}^{(j+1)} = \mathbf{J}^{(j+1)} \left(\boldsymbol{Z}^{(j+1)} - \boldsymbol{\mu}_{\boldsymbol{Z}}^{(j+1)} \right) + \sigma(\boldsymbol{\mu}_{\boldsymbol{Z}}^{(j+1)}), \tag{11}$$

we need to define the diagonal Jacobian matrix of the transformation evaluated at $\boldsymbol{\mu}_{\boldsymbol{Z}}^{(j+1)}$, $\mathbf{J}^{(j+1)} = \mathrm{diag}\left(\nabla_{\boldsymbol{z}} \sigma(\boldsymbol{\mu}_{\boldsymbol{Z}}^{(j+1)})\right)$. Using again the properties for linear functions of Gaussian

random variables, we obtain

$$\begin{aligned}
\boldsymbol{\mu}_{\boldsymbol{A}}^{(j+1)} &\equiv \mathbb{E}[\boldsymbol{A}^{(j+1)}] &&= \tilde{\sigma}(\boldsymbol{\mu}_{\boldsymbol{Z}}^{(j+1)}),\\
\boldsymbol{\Sigma}_{\boldsymbol{A}}^{(j+1)} &\equiv \operatorname{cov}(\boldsymbol{A}^{(j+1)}) &&= \mathbf{J}^{(j+1)}\boldsymbol{\Sigma}_{\boldsymbol{Z}}^{(j+1)}\mathbf{J}^{(j+1)\intercal},\\
\boldsymbol{\Sigma}_{\boldsymbol{A}\boldsymbol{\theta}}^{(j+1)} &\equiv \operatorname{cov}(\boldsymbol{A}^{(j+1)}, \boldsymbol{\theta}) &&= \mathbf{J}^{(j+1)}\boldsymbol{\Sigma}_{\boldsymbol{Z}\boldsymbol{\theta}}^{(j+1)}.
\end{aligned} \tag{12}$$

Equations 10 & 12 allow propagating the information about the covariance of activation units and its dependence on parameters through any pairs of successive layers. For the input layer, the steps described in Equations 8–12 remain the same except that the activation units $\boldsymbol{A}^{(j)}$ are replaced by the covariates $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_{\boldsymbol{X}}, \boldsymbol{\Sigma}_{\boldsymbol{X}})$.

2.3 Tractable Approximate Gaussian Inference (TAGI)

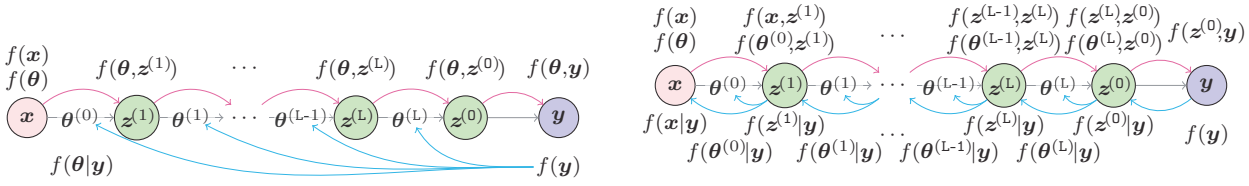
Let us assume we have a set of joint observations for covariates and system responses so that $\mathcal{D} = \{\mathcal{D}_x, \mathcal{D}_y\} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i), \forall i \in \{1 : \mathtt{D}\}\}$. Given that our prior knowledge for the neural network's parameter is $f(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}_{\boldsymbol{\theta}}, \boldsymbol{\Sigma}_{\boldsymbol{\theta}})$, the method presented in §2.1 supposes that the joint PDF $f(\boldsymbol{\theta}, \boldsymbol{y})$ for parameters $\boldsymbol{\theta}$ and observations $\boldsymbol{y}$ is Gaussian with mean vector and covariance

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_{\boldsymbol{\theta}} \\ \boldsymbol{\mu}_{\boldsymbol{Y}} \end{bmatrix}, \; \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{\boldsymbol{\theta}} & \boldsymbol{\Sigma}_{\boldsymbol{Y}\boldsymbol{\theta}}^{\intercal} \\ \boldsymbol{\Sigma}_{\boldsymbol{Y}\boldsymbol{\theta}} & \boldsymbol{\Sigma}_{\boldsymbol{Y}} \end{bmatrix}.$$

The Gaussian inference for the vector $\boldsymbol{\theta}$ given observations $\boldsymbol{Y} = \boldsymbol{y}$ is described by the Gaussian conditional equations $f(\boldsymbol{\theta}|\boldsymbol{y}) = \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}_{\boldsymbol{\theta}|\boldsymbol{y}}, \boldsymbol{\Sigma}_{\boldsymbol{\theta}|\boldsymbol{y}})$ defined by its conditional mean vector and covariance matrix,

$$\begin{aligned}
\boldsymbol{\mu}_{\boldsymbol{\theta}|\boldsymbol{y}} &= \boldsymbol{\mu}_{\boldsymbol{\theta}} + \boldsymbol{\Sigma}_{\boldsymbol{Y}\boldsymbol{\theta}}^{\intercal}\boldsymbol{\Sigma}_{\boldsymbol{Y}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu}_{\boldsymbol{Y}})\\
\boldsymbol{\Sigma}_{\boldsymbol{\theta}|\boldsymbol{y}} &= \boldsymbol{\Sigma}_{\boldsymbol{\theta}} - \boldsymbol{\Sigma}_{\boldsymbol{Y}\boldsymbol{\theta}}^{\intercal}\boldsymbol{\Sigma}_{\boldsymbol{Y}}^{-1}\boldsymbol{\Sigma}_{\boldsymbol{Y}\boldsymbol{\theta}}.
\end{aligned} \tag{13}$$

This 1-step network-wise Gaussian inference procedure is computationally prohibitive because the forward propagation of uncertainty depicted in Figure 4a involves large-sized densely populated matrices, and the inference using Equation 13 again involves full matrices.



(a) Intractable 1-step network-wise inference (Eq. 13) (b) Tractable recursive layer-wise inference (Eq. 14–16)

Figure 4: Representation of the *forward* propagation of uncertainty (magenta arrows) and *inference* procedures (cyan arrows).

The solution we propose to overcome these challenges is twofold: (1) employ a diagonal covariance structure for both the parameters $\boldsymbol{\theta}$, and hidden units $\boldsymbol{Z}^{(j)}$, and (2) use the inherent conditional independence of hidden units between layers, that is, $\boldsymbol{Z}^{(j-1)} \perp\!\!\!\perp \boldsymbol{Z}^{(j+1)}|\boldsymbol{z}^{(j)}$, in order to perform recursive layer-wise Gaussian inference. As depicted in Figures 4 & 9, in the

feedforward process, there is no direct information path between hidden layers $j-1$ and $j+1$ other than through the hidden layer j. Therefore, the knowledge of hidden units at layer j blocks the information from the layer $j-1$, so that layers $j+1$ and $j-1$ are conditionally independent under the assumptions that parameters $\boldsymbol{\theta}$ are independent between layers. As depicted in Figure 4b by the rightmost blue arrow from $\boldsymbol{y}$ to $\boldsymbol{z}^{(\mathtt{0})}$, the first step consists in inferring the posterior mean vector and diagonal covariance for the output layer following

$$\begin{aligned} f(\boldsymbol{z}^{(\mathtt{0})}|\boldsymbol{y}) &= \mathcal{N}(\boldsymbol{z}^{(\mathtt{0})}; \boldsymbol{\mu}_{\boldsymbol{Z}^{(\mathtt{0})}|\boldsymbol{y}}, \boldsymbol{\Sigma}_{\boldsymbol{Z}^{(\mathtt{0})}|\boldsymbol{y}}) \\ \boldsymbol{\mu}_{\boldsymbol{Z}^{(\mathtt{0})}|\boldsymbol{y}} &= \boldsymbol{\mu}_{\boldsymbol{Z}^{(\mathtt{0})}} + \boldsymbol{\Sigma}_{\boldsymbol{Y}\boldsymbol{Z}^{(\mathtt{0})}}^{\intercal} \boldsymbol{\Sigma}_{\boldsymbol{Y}}^{-1} (\boldsymbol{y} - \boldsymbol{\mu}_{\boldsymbol{Y}}) \\ \boldsymbol{\Sigma}_{\boldsymbol{Z}^{(\mathtt{0})}|\boldsymbol{y}} &= \boldsymbol{\Sigma}_{\boldsymbol{Z}^{(\mathtt{0})}} - \boldsymbol{\Sigma}_{\boldsymbol{Y}\boldsymbol{Z}^{(\mathtt{0})}}^{\intercal} \boldsymbol{\Sigma}_{\boldsymbol{Y}}^{-1} \boldsymbol{\Sigma}_{\boldsymbol{Y}\boldsymbol{Z}^{(\mathtt{0})}}. \end{aligned} \tag{14}$$

This step is analogous to performing message passing on the graph presented in Figure 9, where an observation is only available for the output node at the right end. Note that for classification problems, because of the hierarchical formulation described in Appendix C, even if there are $\mathtt{Y}$ classes, only $\mathtt{H} = \lceil \log_2(\mathtt{Y}) \rceil$ hidden units from the output layer are updated for each observation.

The layer-wise Gaussian inference for hidden units and parameters is again analogous to performing message passing on the graph in Figure 9, but this time, from the output layer to input. The backward passing of information depicted by blue arrows in Figure 4b, is done using the Rauch-Tung-Striebel recursive procedure that was developed in the context of state-space models (Rauch et al., 1965). For the RTS procedure, we define the short-hand notation $\{\boldsymbol{\theta}^{+}, \boldsymbol{Z}^{+}\} \equiv \{\boldsymbol{\theta}^{(j+1)}, \boldsymbol{Z}^{(j+1)}\}$ and $\{\boldsymbol{\theta}, \boldsymbol{Z}\} \equiv \{\boldsymbol{\theta}^{(j)}, \boldsymbol{Z}^{(j)}\}$ so that

$$\begin{aligned} f(\boldsymbol{z}|\boldsymbol{y}) &= \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_{\boldsymbol{Z}|\boldsymbol{y}}, \boldsymbol{\Sigma}_{\boldsymbol{Z}|\boldsymbol{y}}) \\ \boldsymbol{\mu}_{\boldsymbol{Z}|\boldsymbol{y}} &= \boldsymbol{\mu}_{\boldsymbol{Z}} + \mathbf{J}_{\boldsymbol{Z}} \left(\boldsymbol{\mu}_{\boldsymbol{Z}^{+}|\boldsymbol{y}} - \boldsymbol{\mu}_{\boldsymbol{Z}^{+}} \right) \\ \boldsymbol{\Sigma}_{\boldsymbol{Z}|\boldsymbol{y}} &= \boldsymbol{\Sigma}_{\boldsymbol{Z}} + \mathbf{J}_{\boldsymbol{Z}} \left(\boldsymbol{\Sigma}_{\boldsymbol{Z}^{+}|\boldsymbol{y}} - \boldsymbol{\Sigma}_{\boldsymbol{Z}^{+}} \right) \mathbf{J}_{\boldsymbol{Z}}^{\intercal} \\ \mathbf{J}_{\boldsymbol{Z}} &= \boldsymbol{\Sigma}_{\boldsymbol{Z}\boldsymbol{Z}^{+}} \boldsymbol{\Sigma}_{\boldsymbol{Z}^{+}}^{-1}, \end{aligned} \tag{15}$$

$$\begin{aligned} f(\boldsymbol{\theta}|\boldsymbol{y}) &= \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}_{\boldsymbol{\theta}|\boldsymbol{y}}, \boldsymbol{\Sigma}_{\boldsymbol{\theta}|\boldsymbol{y}}) \\ \boldsymbol{\mu}_{\boldsymbol{\theta}|\boldsymbol{y}} &= \boldsymbol{\mu}_{\boldsymbol{\theta}} + \mathbf{J}_{\boldsymbol{\theta}} \left(\boldsymbol{\mu}_{\boldsymbol{Z}^{+}|\boldsymbol{y}} - \boldsymbol{\mu}_{\boldsymbol{Z}^{+}} \right) \\ \boldsymbol{\Sigma}_{\boldsymbol{\theta}|\boldsymbol{y}} &= \boldsymbol{\Sigma}_{\boldsymbol{\theta}} + \mathbf{J}_{\boldsymbol{\theta}} \left(\boldsymbol{\Sigma}_{\boldsymbol{Z}^{+}|\boldsymbol{y}} - \boldsymbol{\Sigma}_{\boldsymbol{Z}^{+}} \right) \mathbf{J}_{\boldsymbol{\theta}}^{\intercal} \\ \mathbf{J}_{\boldsymbol{\theta}} &= \boldsymbol{\Sigma}_{\boldsymbol{\theta}\boldsymbol{Z}^{+}} \boldsymbol{\Sigma}_{\boldsymbol{Z}^{+}}^{-1}. \end{aligned} \tag{16}$$

The key aspect of this layer-wise approach is that in the forward step depicted in Figure 4b by magenta arrows, we only need to store the layer-wise mean vectors $\{\boldsymbol{\mu}_{\boldsymbol{\theta}}, \boldsymbol{\mu}_{\boldsymbol{Z}}\}$ and covariances $\{\boldsymbol{\Sigma}_{\boldsymbol{\theta}}, \boldsymbol{\Sigma}_{\boldsymbol{Z}}, \boldsymbol{\Sigma}_{\boldsymbol{\theta}\boldsymbol{Z}^{+}}, \boldsymbol{\Sigma}_{\boldsymbol{Z}\boldsymbol{Z}^{+}}\}$, where in addition to the relations already given in §2.2, the cross-covariance matrix for two successive layers is

$$\boldsymbol{\Sigma}_{\boldsymbol{Z}\boldsymbol{Z}^{+}} = \mathbf{F}_{\boldsymbol{w}\boldsymbol{a}}^{(j)} \mathrm{cov}(\boldsymbol{W}\boldsymbol{A}^{(j)}, \boldsymbol{Z}^{(j)}) + \mathbf{F}_{\boldsymbol{b}}^{(j)} \mathrm{cov}(\boldsymbol{B}^{(j)}, \boldsymbol{Z}^{(j)}).$$

With a diagonal covariance structure for both $\boldsymbol{Z}$ and $\boldsymbol{\theta}$, the covariance matrices defining each layer contain at most $\mathtt{A}^2 + \mathtt{A}$ non-zero terms, i.e., the number of weights ($\mathtt{A}^2$) and biases ($\mathtt{A}$) per layer; Because of the diagonal structures of covariance matrices, equations 14–16 have a computational complexity $\mathcal{O}(\mathtt{A}^2)$, which scales linearly with the number of hidden layers $\mathtt{L}$.

The inference of parameters $\boldsymbol{\theta}$ is done recursively so that after having seen either a single observation or a batch of them, the posterior becomes the prior for the next observations. In order to learn from batches consisting of multiple observations, one can perform the forward propagation of uncertainty for several observations, all sharing the same hyperparameters $\boldsymbol{\eta} = \{\boldsymbol{\mu}_{\boldsymbol{\theta}}, \boldsymbol{\Sigma}_{\boldsymbol{\theta}}\}$, and then do a single update for all of them. As it is the case for deterministic neural networks, the batch procedure allows leveraging parallel computing.

One aspect that needs to be considered is that the finals result of the inference will depend on the initialization of hyperparameters $\boldsymbol{\eta}$, as well as on data ordering. It happens because neural networks are inherently non-identifiable so that the true posterior for their parameters is multimodal. For example, we can imagine how any permutation of hidden units in a same layer that would not change the results but would lead to a different posterior. Therefore, TAGI depends upon hyperparameter initialization and data ordering because it approximates a multimodal posterior with an unimodal multivariate Gaussian.

2.4 Hyper-parameter estimation

There are typically tens of thousands, if not millions, of parameters in $\boldsymbol{\theta}$, for which we typically have little or no prior information for defining the hyper-parameters $\boldsymbol{\eta}^{(\emptyset)} = \{\boldsymbol{\mu}_{\boldsymbol{\theta}}^{(\emptyset)}, \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{(\emptyset)}\}$. In the case where we have small datasets, the weakly informative prior combined with limited data will lead to a weakly informative posterior. One solution to go around this difficulty while avoiding overfitting is to learn the model parameters over multiple epochs, $\mathtt{E} > 1$, using a training $\mathcal{D}_{\mathtt{T}}$ and validation set $\mathcal{D}_{\mathtt{V}}$. Here, we propose to employ the posterior's hyper-parameter values at the i^{th} iteration $\boldsymbol{\eta}^{(i)} = \{\boldsymbol{\mu}_{\boldsymbol{\theta}|\mathcal{D}_{\mathtt{T}}}^{(i)}, \boldsymbol{\Sigma}_{\boldsymbol{\theta}|\mathcal{D}_{\mathtt{T}}}^{(i)}\}$ and use them as the prior's hyper-parameters at the next iteration $i+1$. This recursive procedure is stopped when the marginal likelihood $f(\mathcal{D}_{y,\mathtt{V}}|\mathcal{D}_{x,\mathtt{V}}, \boldsymbol{\eta}^{(i)}) = \mathcal{N}\big(\mathcal{D}_{\mathtt{V}}; \boldsymbol{\mu}_{\boldsymbol{y}_{\mathtt{V}}|\mathcal{D}_{\mathtt{T}}}^{(i)}, \boldsymbol{\Sigma}_{\boldsymbol{y}_{\mathtt{V}}|\mathcal{D}_{\mathtt{T}}}^{(i)}\big)$ for the validation set $\mathcal{D}_{\mathtt{V}}$, has reached its maximal value. This procedure is analogous to the empirical Bayes approach (Efron, 2012) where the prior knowledge's hyper-parameters are learnt through the maximization problem

$$\hat{\boldsymbol{\eta}} = \arg\max_{\boldsymbol{\eta}} \int f(\mathcal{D}_{y,\mathtt{V}}|\mathcal{D}_{x,\mathtt{V}}, \boldsymbol{\theta}) \cdot f(\boldsymbol{\theta}|\boldsymbol{\eta}) d\boldsymbol{\theta}. \tag{17}$$

Note that unlike in Equation 17 where the maximization is explicit, in our case the maximization is implicitly performed by updating over multiple epochs. Even if the recursive approach proposed is not guaranteed to lead to the Type-2 maximum likelihood estimate solution sought by empirical Bayes, it is much more efficient than having to perform the explicit maximization for the expected value and variance parameters.

3. Related work

Many researchers have already proposed approximate inference methods for *Bayesian Neural Networks* (BNN). Early on, it was proposed to employ *message passing* methods (Murphy, 2012, §18.3) such as the *Extended* (Singhal and Wu, 1989; Puskorius and Feldkamp, 1991), *Unscented Kalman Filter* (Wan and Merwe, 2000), and the *RTS smoother* (Haykin, 2004) to leverage Gaussian inference in order to approximate the posterior. The main issue with these approaches is related to their computational complexity which is proportional to the square

of the number of weights (Wan and Merwe, 2000). More recently, this Gaussian inference framework was extended with the *Cubature filter* (Arasaratnam and Haykin, 2008), which, like the extended and unscented methods, is limited by its computational complexity. The main factor limiting the computational efficiency of these approaches is the usage of full covariance matrices for the network parameters. Plumer (1995) has proposed to employ either fully-decoupled or layer-decoupled variants, but the loss in performance prevents their recommendation as a default choice (Haykin, 2004).

In another early approach, MacKay (1992) employed the *Laplace approximation* to describe the posterior covariance of parameters. The author experimented with neural networks having a small number of hidden units per layer (i.e., 5–20) and noted that "*diagonal approximation are no good because of the strong posterior correlation*". Later, Neal (1995) explored the potential of BNN using the Hamiltonian Monte Carlo (HMC) method. Despite its lack of practical scalability in the context of large neural networks applied to image classification tasks, natural language processing, or reinforcement learning, HMC is seen as a classic reference method for Bayesian inference (Gelman et al., 2014). For instance, Farquhar et al. (2020) have employed HMC and other structured-covariance methods to demonstrate empirically that there is no significant difference in performance when using a diagonal or a full covariance structure for the weight parameters of deep neural networks.

In parallel, several researchers have applied *variational inference* for estimating the posterior distribution of neural networks' parameters (Hinton and Van Camp, 1993; Barber and Bishop, 1998). The development of moment matching and variational approaches for BNN is still nowadays an active research area (Kingma et al., 2015; Hernández-Lobato and Adams, 2015; Blundell et al., 2015; Louizos and Welling, 2016; Osawa et al., 2019; Wu et al., 2019). Recently, the technique of using Dropout as a Bayesian approximation has received a lot of attention in the community (Gal and Ghahramani, 2016). All the recent methods that are either based on moment matching, variational approaches, or dropout, share a common aspect; the inference of parameters is still treated as an optimization problem relying on gradient backpropagation. Although some approaches such as the one by Wu et al. (2019) have exploited approximate analytically tractable mean and variance propagation under Gaussian assumptions, it still relies on variational inference, as none of the approaches currently available allow for analytically tractable inference in neural networks, while reaching a competitive level of performance and efficiency.

4. Experiments

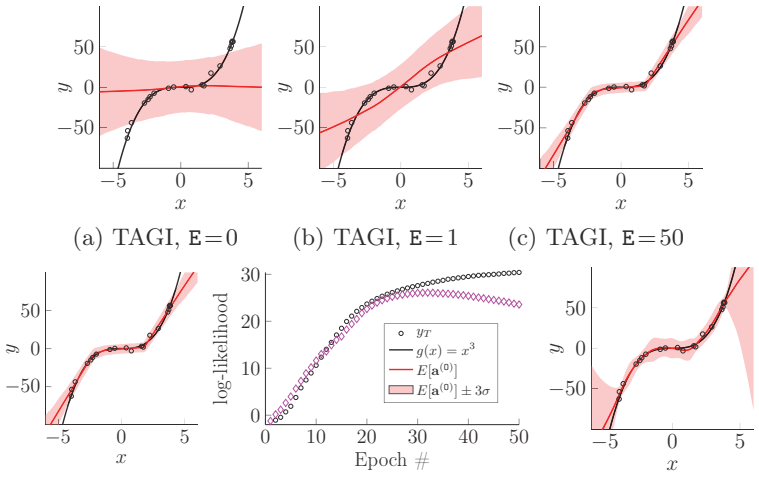
In this section, we perform experiments using the TAGI method for a 1D toy problem, for a set of benchmark regression problems, and for the MNIST classification dataset.

4.1 1D toy problem

We apply TAGI to the 1D regression problem $y = x^3 + v$, such that $V \sim \mathcal{N}(0, 9)$, as taken from (Hernández-Lobato and Adams, 2015), using an AG-FNN having a single hidden layer with 100 units, and ReLU activation functions. The objective of this case study is to showcase how TAGI can be applied on small datasets ($\mathtt{D} = 20$ points), and to compare the results obtained by considering either diagonal or full covariance matrices. In this example, the inference is performed using one observation at a time, where both the covariates x and

observations y were normalized in the range $[-1, 1]$. The optimal number of epochs is identified from a validation set $\mathcal{D}_{\mathtt{V}}$ consisting of 20 additional points. The prior covariance for bias is initialized to $\boldsymbol{\Sigma}_{\boldsymbol{B}}^{\emptyset} = 0.01 \cdot \mathbf{I}$, and for weights $\boldsymbol{\Sigma}_{\boldsymbol{W}}^{\emptyset}$, by using the Xaviers's approach (Glorot and Bengio, 2010). The prior mean vector is randomly sampled from $\boldsymbol{\mu}_{\boldsymbol{\theta}}^{\emptyset} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{\emptyset})$ in order to break the initial symmetry in the network, as starting with zero expected values for weights adversely affects learning.

Figure 5 compares the true function employed to generate the data, with the AG-FNN (with diagonal covariances) predictions described by their expected values and $\pm 3\sigma$ confidence regions. We can see in (a) that the prior predictive obtained before updating with observations ($\mathtt{E} = 0$) is weakly informative, and that the posterior predictive obtained after the first epoch (b, $\mathtt{E} = 1$) is still a poor approximation of the true function. The log-likelihood reported in (d) for the validation set allows identifying that the optimal number of epochs is here $\mathtt{E} = 31$. The log-likelihood values reported in Figure 5d confirm that employing a training set to identify the number of epochs would not be able to prevent overfitting as depicted in (c). We can see in (e) the same results processed with Hamiltonian Monte-Carlo (HMC) using the implementation by Cobb et al. (2019). One aspect we observe is that TAGI is not able to correctly extrapolate the widening confidence region outside the training data, as correctly displayed by HMC. The main reason behind this behaviour is that TAGI only provide an unimodal posterior; In the case of HMC, the multiple posterior modes all coincide to similar predictions when constrained by data but not during extrapolation.



(a) TAGI, $\mathtt{E}=0$ (b) TAGI, $\mathtt{E}=1$ (c) TAGI, $\mathtt{E}=50$

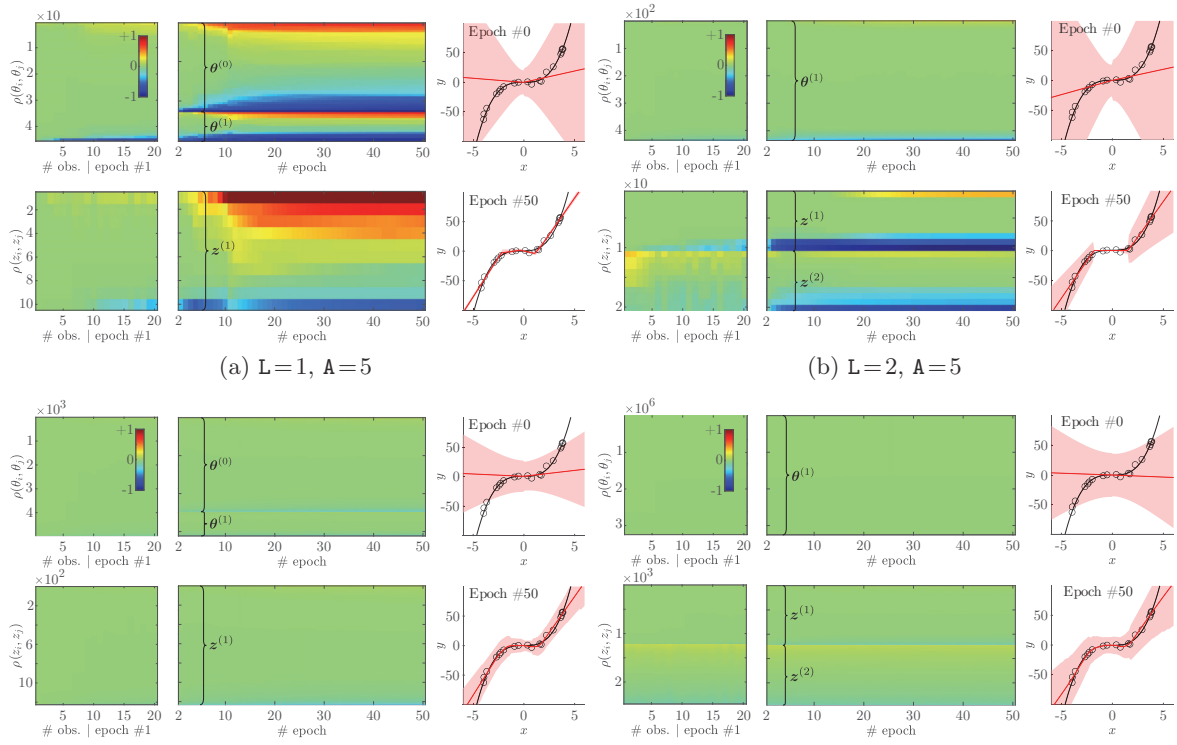
(d) TAGI, $\mathtt{E}=31$ (e) HMC

Figure 5: Application of AG-FNN with diagonal covariances to a toy problem where (a–c) describe the evolution of the predictive distribution with respect to the number of epochs $\mathtt{E}$. In (d), we compare the training and validation log-likelihood in order to identify the optimal number of epochs, i.e., $\mathtt{E} = 31$. In (e) we compare with the results obtained using HMC.

In a second experiment, we now apply TAGI to the same dataset while considering the full covariance matrices for $\boldsymbol{\theta}^{(j)}$ and $\boldsymbol{z}^{(j)}$. We study networks with either $\mathtt{L}=1$ or 2 hidden layers, each comprising either $\mathtt{A} = 5$ or 50 hidden units. Figure 6 displays colour maps where each pixel represents the layer-wise sorted correlation coefficients extracted from the

upper-triangular portion of the posterior covariance matrices, for either the parameters $\boldsymbol{\theta}^{(j)}$ (top), or the hidden units $\boldsymbol{z}^{(j)}$ (bottom). Note that in the cases (b) and (d), results are only presented for the fully connected layer $\boldsymbol{\theta}^{(1)}$ because of the disproportionate number of weights in it in comparison with the input and output layers. The results displayed on the left colour map from each subfigure corresponds to each observation from the first epoch, and the right colour maps are for the last observation from subsequent epochs.

In Figure 6a,b for hidden layers of $\mathtt{A} = 5$ units, we notice the presence of extensive large positive and negative correlations in the posteriors. In Figure 6c,d for hidden layers of $\mathtt{A} = 50$ units, non-zero correlations, i.e., those with a colour other than green, are restricted to a small fraction of the covariance matrix. This can be explained by the theoretical results presented in Figure 2, and by further looking at the simplified context for the sum of several independent inputs $Z_i \sim \mathcal{N}(z; 0, 1)$ such that the observation model is $y = \sum_{i=1}^{\mathtt{A}} z_i$. If we observe a constant $y \in \mathbb{R}$, we can then exactly infer the Gaussian posterior for $\boldsymbol{Z}$, for which the correlations $\rho(Z_i, Z_j),\ \forall i \neq j$ are directly impacted by the number of variables $\mathtt{A}$ as depicted in Figure 7. For small numbers of hidden units, i.e. < 10, the posterior correlation is non-negligible, whereas, for large numbers, i.e. > 100, it is almost zero. This confirms what



(a) $\mathtt{L}=1$, $\mathtt{A}=5$ (b) $\mathtt{L}=2$, $\mathtt{A}=5$

(c) $\mathtt{L}=1$, $\mathtt{A}=50$ (d) $\mathtt{L}=2$, $\mathtt{A}=50$

Figure 6: Representation of the sorted correlation coefficients extracted from upper-triangular posterior covariance matrices for the parameters $\boldsymbol{\theta}^{(j)}$ and hidden units $\boldsymbol{z}^{(j)}$. The left-most graphs present the correlations for each of the 20 observations within the first epoch, and the center graphs present the correlations at the end of each epoch.

was shown theoretically in Figure 2, that if the number of units per layer $\mathtt{A}$ is sufficient, the hidden units among a layer are independent, and thus considering only diagonal covariance matrices may lead to results that are comparable to those obtained while considering the full covariance structure.

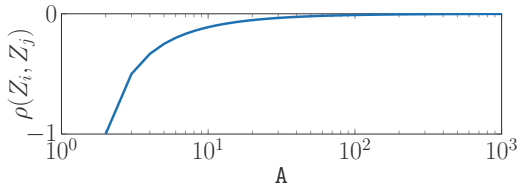


Figure 7: Evolution of the posterior correlation $\rho(Z_i, Z_j)$ as a function of the number of variables $\mathtt{A}$ for an observation model $y = \sum_{i=1}^{\mathtt{A}} z_i$, where all $Z_i \sim \mathcal{N}(z, 0, 1)$ are independent.

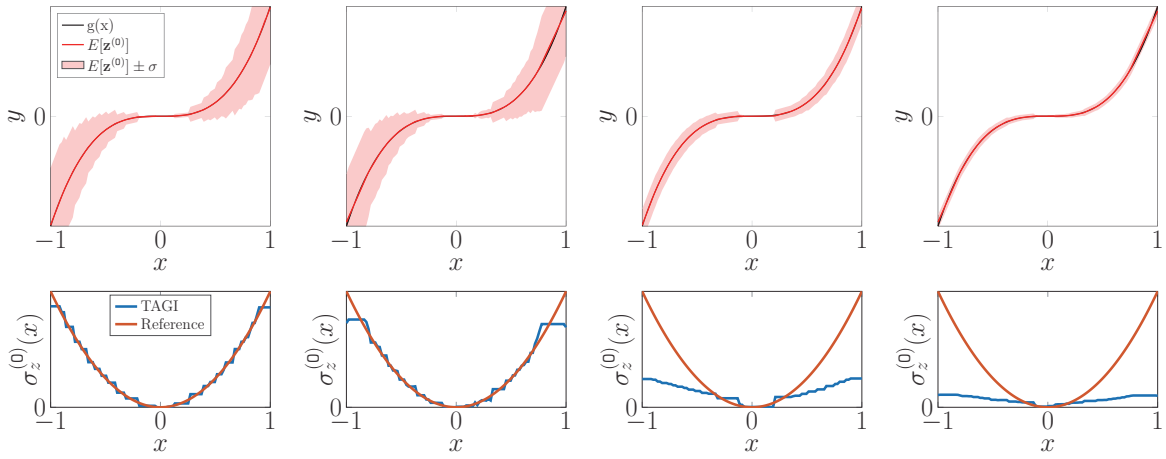
The third and last experiment exposes the limitations of using diagonal covariance matrices for hidden units, in the case where we care about the correct propagation of the input-layer uncertainty through a neural network. We use again the function $g(x) = x^3$, but this time in the range $x \in [-1, 1]$, and with infinitely many exact observations so that $\boldsymbol{\Sigma}_{\boldsymbol{\theta}} \to 0$ and the parameters can be considered as deterministic. Once we have learnt the parameters exactly, we want to test the capacity to propagate the input-layer uncertainty for any value x subject to an observation uncertainty such that $\tilde{X} \sim \mathcal{N}(x; 0, \sigma_X^2)$, where $\sigma_X = \frac{2}{10}$. As depicted in Figure 8, because we use a ReLU activation function, the neural network's output is a piecewise-linear function and the reference conditional output standard deviation for such a locally linearized function can be computed using the 1^{st} order Taylor expansion so that

$$\sigma_Z^{(\mathtt{O})}(x) = \sqrt{\text{var}[g(\tilde{X})|x]} = \left|\frac{dg(x)}{dx}\right| \cdot \sigma_X = \frac{6x^2}{10}. \tag{18}$$

The results in Figure 8 are for both, full or diagonal hidden variable covariances $\boldsymbol{\Sigma}_{\boldsymbol{Z}}$, and for one or two hidden layers. We can see that with a full $\boldsymbol{\Sigma}_{\boldsymbol{Z}}$, the neural network's output uncertainty closely follow the reference values for $\sigma_Z^{(\mathtt{O})}(x)$. When using diagonal covariances for hidden layers, $\boldsymbol{\Sigma}_{\boldsymbol{Z}} = \text{diag}(\boldsymbol{\sigma}_{\boldsymbol{Z}})$, we are unable to retrieve the same reference values as defined in Equation 18. Therefore, despite having attested in the previous experiments that TAGI can learn the parameters while using diagonal covariance matrices, it remains unable to correctly propagate uncertainty from the input layer to the output. For applications where propagating the input-layer uncertainty is critical, e.g. in partially observable reinforcement learning (Sutton and Barto, 2018; Jin et al., 2018), one may choose to sacrifice the linear computational complexity of TAGI (see §2.3) by including full $\boldsymbol{\Sigma}_{\boldsymbol{Z}}$ matrices for each layer during the forward propagation of uncertainty.

4.2 Benchmark regression problems

The performance of TAGI is now compared with PBP (Hernández-Lobato and Adams, 2015), VMG (Louizos and Welling, 2016), and MC-dropout (Gal and Ghahramani, 2016) for benchmark regression datasets. For the purpose of comparison with existing results taken from the literature, all the datasets are analyzed for a fixed number of epochs, that is $\mathtt{E} = 40$.



(a) L = 1, A = 50, $\boldsymbol{\Sigma}_{\boldsymbol{Z}}$: full (b) L = 2, A = 50, $\boldsymbol{\Sigma}_{\boldsymbol{Z}}$: full (c) L = 1, A = 50, $\boldsymbol{\Sigma}_{\boldsymbol{Z}}$: diag (d) L = 2, A = 50, $\boldsymbol{\Sigma}_{\boldsymbol{Z}}$: diag

Figure 8: Comparison of the capacity to propagate uncertainty from the input to the output of a neural network while using either a full or a diagonal covariance $\boldsymbol{\Sigma}_{\boldsymbol{Z}}$. The top graphs represent the neural network output with confidence interval where the bottom graphs compare the reference values with the empirical ones.

For all cases, the data is normalized, the activation function is a ReLU, and the batch size is B = 10; The prior covariance for biases is initialized to $\boldsymbol{\Sigma}_{\boldsymbol{B}}^{0} = 0.01 \cdot \mathbf{I}$, and for weights $\boldsymbol{\Sigma}_{\boldsymbol{W}}^{0}$, by using the Xaviers's approach (Glorot and Bengio, 2010); The initial value for the observation error's standard deviation is set to $\sigma_V = 1$, and this value is optimized using a 5-folds cross-validation setup.

The results reported in Table 1 indicate that TAGI matches the performance of existing methods in terms of root mean square error (RMSE) and log-likelihood (LL). Even though VMG displays the best predictive performance, its computational time is two orders of magnitude greater than TAGI, PBP and MC-Dropout (Sun et al., 2017). The timing details reported in Appendix E show that the current TAGI's implementation has a computational

Table 1: Comparison TAGI's results with those available in the literature for PBP (Hernández-Lobato and Adams, 2015), VMG (Louizos and Welling, 2016), and MC-dropout (Gal and Ghahramani, 2016) (Rank legend: **first**, **second**). The $\pm\sigma$ represent the standard deviation over the 20 test-folds.

Datasets	Root mean square error (RMSE)				Average log-likelihood (LL)			
	PBP	VMG	MC-Dropout	**TAGI**	PBP	VMG	MC-Dropout	**TAGI**
Boston	3.01±0.18	**2.70±0.13**	**2.97±0.85**	2.98±0.86	**-2.56±0.12**	**-2.46±0.09**	**-2.46±0.25**	-2.58±0.45
Concrete	5.67±0.09	**4.89±0.12**	**5.23±0.53**	5.72±0.52	-3.14±0.11	**-3.01±0.03**	**-3.04±0.09**	-3.17±0.09
Energy	1.80±0.05	**0.54±0.02**	1.66±0.19	**1.46±0.22**	-2.04±0.02	**-1.06±0.03**	-1.99±0.09	**-1.81±0.14**
Kin8nm	0.10±0.00	0.10±0.00	0.10±0.00	0.10±1E-3	0.90±0.01	**1.10±0.01**	**0.95±0.03**	0.88±0.04
Naval	0.01±0.00	**0.00±0.00**	0.01±0.00	0.01±6E-3	**3.73±0.01**	2.46±0.12	**3.80±0.05**	2.10±0.57
Power	4.12±0.03	**4.04±0.04**	**4.02±0.18**	4.12±0.16	-2.84±0.01	**-2.82±0.01**	**-2.80±0.05**	-2.83±0.04
Protein	4.73±0.01	**4.13±0.02**	**4.36±0.04**	4.70±0.02	-2.97±0.00	**-2.84±0.00**	**-2.89±0.01**	-2.97±4E-3
Wine	0.64±0.01	**0.63±0.01**	**0.62±0.01**	**0.63±0.04**	-0.97±0.01	**-0.95±0.01**	**-0.93±0.06**	-0.96±0.06
Yacht	1.02±0.05	**0.71±0.05**	1.11±0.38	**1.02±0.42**	-1.63±0.02	**-1.30±0.02**	-1.55±0.12	**-1.49±0.45**

time that is more than four times faster than PBP and MC-Dropout (Gal and Ghahramani, 2016). Moreover, because TAGI had to be implemented from scratch, it is not yet fully optimized for computational efficiency. The same is true for the optimization of hyper-parameters such as σ_V, which is assumed to be constant over the covariate domain, and which is currently treated as an hyperparameter to be estimated separately from the analytical inference for weights and biases. The fact that TAGI is currently limited to the case of homoscedastic variance reduces its performance in comparison with the other methods presented, and also inevitably lead to poorly calibrated predictive uncertainties. Note that as it was the case for the other methods reported in Table 1, the number of epochs employed was not optimized.

4.3 Application on MNIST

We apply TAGI to the MNIST classification problem (LeCun et al., 1998) consisting of $\texttt{D} = 70\,000$ (28×28) greyscale images for $\texttt{K} = 10$ classes (60 000 training and 10 000 test). Here, we compare the performance of two AG-FNN configurations, each having $\texttt{L} = 2$ hidden layers with a number of hidden units equal to $\texttt{A} \in \{100, 800\}$. Each AG-FNN has the same structure for the input ($\texttt{X} = 784$ nodes) and the output layer ($\texttt{Y} = 11$ nodes). The ReLU activation function is used for the two hidden layers. For each digit, the vector of covariates $\mathbf{x}_i \in (0, 1)^{784}$ is assumed to be deterministic so that $\boldsymbol{\mu}_{\boldsymbol{X}_i} = \mathbf{x}_i$ and $\boldsymbol{\Sigma}_{\boldsymbol{X}_i} = \mathbf{0}$. The prior covariance for bias is initialized to $\Sigma^{\emptyset}_{\boldsymbol{B}} = 0.01 \cdot \mathbf{I}$, and by using the Xaviers's initialization approach (Glorot and Bengio, 2010) for weights $\Sigma^{\emptyset}_{\boldsymbol{W}}$. The prior mean vector is randomly sampled from $\boldsymbol{\mu}^{\emptyset}_{\boldsymbol{\theta}} \sim \mathcal{N}(\mathbf{0}, \Sigma^{0}_{\boldsymbol{\theta}})$. The hyper-parameter associated with the output layer is set to $\alpha = 1/3$. The posterior mean vector as well as the main diagonal of the posterior covariance are learnt using two setups: (1) a single observation per batch, that is, $\texttt{B} = 1$, and (2) $\texttt{B} = 10$ observations per batch. Each network is evaluated for $\sigma_V = \{0.1, 0.2, 0.3, 0.4\}$ and the optimal value for σ_V is selected using a randomly selected validation set corresponding to 5% of the training set. The optimal number of epochs $\texttt{E}$ is identified using an early-stop procedure evaluated on the validation set.

Table 2 presents the average test error evaluated on 10 000 images for the different AG-FNN configurations for $\texttt{E} = 1$ epoch, and for the optimal number of epochs found using early stopping. In order to factor in the effect of random weight initialization, the results reported are the average and standard deviations from five runs. The performance achieved

Table 2: MNIST test-set average classification error [%] for the first and last epoch. A: number of hidden units on each layer; B: number of observations per batch; E: number of epochs; e: optimal number of epoch found using early-stop. The results reported are the average of five runs along with $\pm$ one standard deviation.

	B = 1				B = 10			
A	E = 1	E = e	e	σ_V	E = 1	E = e	e	σ_V
100	3.39±0.17	2.29±0.06	21±12	0.2	3.56±0.2	2.45±0.20	20±12	0.4
800	3.15±0.2	1.54±0.07	14±9	0.1	3.10±0.2	1.53±0.07	15±6	0.4

with respect to the average classification errors matches the reported state-of-the-art accuracy

of approximately 1.6% for FNNs having a same architecture with 2 layers and 800 hidden units and trained using gradient backpropagation (Simard et al., 2003; Wan et al., 2013). Some Bayesian methods for neural networks have shown to be able to outperform their deterministic counterparts. For example, Blundell et al. (2015) showed that they can reach an error rate of 1.34% using Bayes by Backprop with scale mixture, while only reaching 1.99% when using Gaussians. Similarly Louizos and Welling (2016) showed that VMG could reach a performance of 1.15% with even smaller networks. The results in Table 2 indicate that TAGI's classification accuracy is not significantly affected by the usage of batch sizes greater than one. Nevertheless, we noticed though our experiments that using large batch sizes makes the learning phase sensitive to the network initialization as well as the observation noise parameter σ_V.

5. Conclusion

The tractable approximate Gaussian inference method proposed in this paper allows for the analytical inference of the posterior mean vector and diagonal covariance matrix for the parameters of Bayesian neural networks. The applications on the regression and classification datasets validate that the approach matches the performance of existing methods with respect to computational efficiency and accuracy. TAGI's performance and its linear complexity with respect to the number of parameters makes it a viable alternative to gradient backpropagation. By allowing the treatment of uncertainty and by being inherently suited for online inference, we foresee that the approach will enable transformative developments in supervised, unsupervised, and reinforcement learning.

Acknowledgements

The second author was financially supported by research grants from Hydro-Quebec/IREQ, and the Natural Sciences and Engineering Research Council of Canada (NSERC). The third author was supported by a research grant from the Institute for Data Valorization (IVADO).

References

I. Arasaratnam and S. Haykin. Nonlinear Bayesian filters for training recurrent neural networks. In *Mexican International Conference on Artificial Intelligence*, pages 12–33. Springer, 2008.

D. Barber and C. M. Bishop. Ensemble learning in Bayesian neural networks. *NATO ASI Series F Computer and Systems Sciences*, 168:215–238, 1998.

C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural network. In *International Conference on Machine Learning*, pages 1613–1622, 2015.

A.D. Cobb, Atılım G. Baydin, A. Markham, and S.J. Roberts. Introducing an explicit symplectic integration scheme for Riemannian manifold Hamiltonian Monte Carlo. *arXiv preprint arXiv:1910.06243*, 2019.

B. Efron. *Large-scale inference: empirical Bayes methods for estimation, testing, and prediction*, volume 1. Cambridge University Press, 2012.

S. Farquhar and Y. Gal. A unifying Bayesian view of continual learning. *arXiv preprint arXiv:1902.06494*, 2019.

S. Farquhar, L. Smith, and Y. Gal. Liberty or depth: Deep bayesian neural nets do not need complex weight posterior approximations. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 4346–4357, 2020.

Y. Gal and Z. Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, pages 1050–1059, 2016.

A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian data analysis*. CRC Press, 3rd edition, 2014.

Z. Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553): 452, 2015.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT Press, 2016.

S. Haykin. *Kalman filtering and neural networks*, volume 47. John Wiley & Sons, 2004.

J. M. Hernández-Lobato and R. Adams. Probabilistic backpropagation for scalable learning of Bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869, 2015.

G. E. Hinton and D. Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13. ACM, 1993.

P. Jin, K. Keutzer, and S. Levine. Regret minimization for partially observable deep reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2342–2351, 10–15 Jul 2018.

A. Kendall and Y. Gal. What uncertainties do we need in Bayesian deep learning for computer vision? In *Advances in neural information processing systems*, pages 5574–5584, 2017.

D. P. Kingma, T. Salimans, and M. Welling. Variational dropout and the local reparameterization trick. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28, 2015.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

C. Louizos and M. Welling. Structured and efficient variational deep learning with matrix Gaussian posteriors. In *International Conference on Machine Learning*, pages 1708–1716, 2016.

D.J.C. MacKay. A practical Bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

F. Morin and Y. Bengio. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252. Citeseer, 2005.

K. P. Murphy. *Machine learning: a probabilistic perspective*. The MIT Press, 2012.

R. M. Neal. *Bayesian learning for neural networks*. PhD thesis, University of Toronto, 1995.

K. Osawa, S. Swaroop, A. Jain, R. Eschenhagen, R. E. Turner, R. Yokota, and M. E. Khan. Practical deep learning with Bayesian principles. In *Advances in Neural Information Processing Systems*, 2019.

E. S. Plumer. Training neural networks using sequential extended Kalman filtering. In *1995 world congress on neural networks*. Los Alamos National Lab., NM (United States), 1995.

G. V. Puskorius and L. A. Feldkamp. Decoupled extended Kalman filter training of feedforward layered networks. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume 1, pages 771–777. IEEE, 1991.

C. E. Rasmussen and C. K. Williams. *Gaussian processes for machine learning*. the MIT Press, 2006.

H. E. Rauch, C. T. Striebel, and F. Tung. Maximum likelihood estimates of linear dynamic systems. *AIAA journal*, 3(8):1445–1450, 1965.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

P. Y Simard, D. Steinkraus, and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Proceedings of Seventh International Conference on Document Analysis and Recognition*, 2003.

S. Singhal and L. Wu. Training multilayer perceptrons with the extended Kalman algorithm. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 133–140. Morgan-Kaufmann, 1989.

S. Sun, C. Chen, and L. Carin. Learning structured weight uncertainty in Bayesian neural networks. In *Artificial Intelligence and Statistics*, pages 1283–1292, 2017.

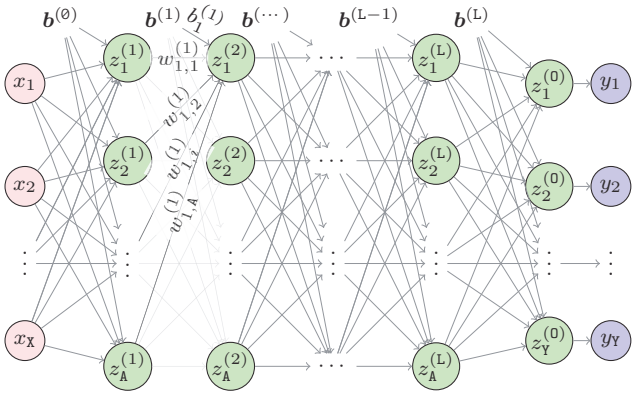
R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT Press, 2nd edition, 2018.

E. A. Wan and R. v. d. Merwe. The unscented Kalman filter for nonlinear estimation. In *Adaptive Systems for Signal Processing, Communications, and Control Symposium*, pages 153–158. IEEE, 2000.

L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus. Regularization of neural networks using DropConnect. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1058–1066, 2013.

A. Wu, S. Nowozin, E. Meeds, R. E. Turner, J. M. Hernández-Lobato, and A. L. Gaunt. Deterministic variational inference for robust Bayesian neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

Appendix A. Feedforward neural network nomenclature



(a) Expanded representation

(b) Compact representation where $\boldsymbol{\theta} = \{\boldsymbol{w}, \boldsymbol{b}\}$

Figure 9: Expanded and compact representations of the variable nomenclature associated with feedforward neural networks.

Appendix B. Derivation for the Gaussian multiplicative approximation

The proof of statement (3) can be obtained directly from the definition of covariance. To prove the statement (4), one needs the moments of variables and their products. Because the underlying distribution is Gaussian and its moment generating function are known, $M_{\boldsymbol{X}}(\mathbf{t}^\intercal) = \mathbb{E}[e^{\mathbf{t}^\intercal \boldsymbol{X}}] = e^{\mathbf{t}^\intercal \boldsymbol{\mu} + \frac{1}{2}\mathbf{t}^\intercal \boldsymbol{\Sigma} \mathbf{t}}, \mathbf{t} = [t_1 \ \dots \ t_4]^\intercal$, all moments can be extracted using the moment-generating function. Using the derivatives of the moment-generating function,

$$
\begin{aligned}
\mathbb{E}[X_1 X_2 X_3] &= \frac{\partial^3}{\partial t_1 \partial t_2 \partial t_3} \mathbb{E}[e^{\mathbf{t}^\intercal \boldsymbol{x}}]|_{t_1=t_2=t_3=t_4=0} \\
&= \frac{\partial^3}{\partial t_1 \partial t_2 \partial t_3} e^{\mathbf{t}^\intercal \boldsymbol{\mu} + \frac{1}{2}\mathbf{t}^\intercal \boldsymbol{\Sigma} \mathbf{t}}|_{t_1=t_2=t_3=t_4=0} \\
&= \frac{\partial^3}{\partial t_1 \partial t_2 \partial t_3} e^{\sum_{i=1}^4 t_i \mu_i + \frac{1}{2}\sum_{i,j=1}^4 t_i t_j \mathrm{cov}(X_i, X_j)}|_{t_{1,2,3,4}=0} \\
&= \mathrm{cov}(X_1, X_2)\mu_3 + \mathrm{cov}(X_1, X_3)\mu_2 \\
&\quad + \mathrm{cov}(X_2, X_3)\mu_1 + \mu_1 \mu_2 \mu_3. \qquad (19)
\end{aligned}
$$

Using the definition of covariance and substituting (19), the statement (4) can be established:

$$
\begin{aligned}
\mathrm{cov}(X_3, X_1 X_2) &= \mathbb{E}[X_1 X_2 X_3] - \mathbb{E}(X_1 X_3)\mathbb{E}(X_2) \\
&= \mathrm{cov}(X_1, X_2)\mu_3 + \mathrm{cov}(X_1, X_3)\mu_2 \\
&\quad + \mathrm{cov}(X_2, X_3)\mu_1 + \mu_1 \mu_2 \mu_3 \\
&\quad - \mu_3\big(\mu_1 \mu_2 + \mathrm{cov}(X_1, X_2)\big). \qquad (20)
\end{aligned}
$$

The expansion of the right side for the last statement leads to Equation (5) so that

$$\mathrm{cov}(X_1X_2, X_3X_4) = \mathbb{E}[X_1X_2X_3X_4] - \mathbb{E}[X_1X_2]\mathbb{E}[X_3X_4],$$

where the expected value of the product two random variables is given in (3) and the expectation for the product of four random variables is a generalization of (19) that can be obtained using the derivatives of the moment-generating function:

$$\begin{aligned}
\mathbb{E}[X_1X_2X_3X_4] &= \frac{\partial^4}{\partial t_1\partial t_2\partial t_3\partial t_4}\mathbb{E}[e^{\mathbf{t}^\intercal \boldsymbol{X}}]|_{t_{1,2,3,4}=0} \\
&= \frac{\partial^4}{\partial t_1\partial t_2\partial t_3\partial t_4} e^{\mathbf{t}^\intercal\boldsymbol{\mu}+\frac{1}{2}\mathbf{t}^\intercal\boldsymbol{\Sigma}\mathbf{t}}|_{t_{1,2,3,4}=0} \\
&= \mathrm{cov}(X_1X_2)\big(\mathrm{cov}(X_3,X_4)+\mu_3\mu_4\big) \\
&+\mathrm{cov}(X_1X_3)\big(\mathrm{cov}(X_2,X_4)+\mu_2\mu_4\big) \\
&+\mathrm{cov}(X_2X_3)\big(\mathrm{cov}(X_1,X_4) \\
&+\mu_1\mu_4\big) + \mathrm{cov}(X_1,X_4)\mu_2\mu_3 + \mathrm{cov}(X_2,X_4)\mu_1\mu_3 \\
&+\mathrm{cov}(X_3,X_4)\mu_1\mu_2 + \mu_1\mu_2\mu_3\mu_4.
\end{aligned}$$

Using the definition of variance

$$\mathrm{var}(X_1X_2) = \mathbb{E}[(X_1X_2)^2] - \mathbb{E}[X_1X_2]^2 \tag{21}$$

The elements of variance can be expended as below

$$\begin{aligned}
\mathbb{E}[(X_1X_2)^2] &= \frac{\partial^4}{\partial t_1^2\partial t_2^2}\mathbb{E}[e^{\mathbf{t}^\intercal \boldsymbol{X}}]|_{t_1=t_2=t_3=t_4=0} \\
&= \frac{\partial^4}{\partial t_1^2\partial t_2^2} e^{\mathbf{t}^\intercal\boldsymbol{\mu}+\frac{1}{2}\mathbf{t}^\intercal\boldsymbol{\Sigma}\mathbf{t}}|_{t_1=t_2=t_3=t_4=0} \\
&= \sigma_1^2\sigma_2^2 + 2\mathrm{cov}(X_1,X_2)^2 \\
&\quad +4\mathrm{cov}(X_1,X_2)\mu_1\mu_2 \\
&\quad +\sigma_1^2\mu_2^2 + \sigma_2^2\mu_1^2 + \mu_1^2\mu_2^2.
\end{aligned} \tag{22}$$

$$\begin{aligned}
\mathbb{E}[X_1X_2]^2 &= \big(\mathrm{cov}(X_1,X_2) + \mathbb{E}(X_1)\mathbb{E}(X_2)\big)^2 \\
&= \mathrm{cov}(X_1,X_2)^2 + 2\mathrm{cov}(X_1,X_2)\mu_1\mu_2 \\
&\quad +\mu_1^2\mu_2^2.
\end{aligned} \tag{23}$$

Substituting (22) and (23) in (21) establishes (6).

Appendix C. Formulation for the binary tree hierarchical decomposition

For classification problem, each class is encoded in a binary tree with $\mathtt{H} = \lceil \log_2(\mathtt{K}) \rceil$ layers, and which is defined by $\mathtt{Y} = \mathtt{K} - 1$ hidden states when $\log_2(\mathtt{K}) \in \mathbb{Z}^+$. Figure 10 depicts the hierarchical decomposition for $\mathtt{K} = 8$ classes and $\mathtt{H} = 3$ layers, where a given class $y_{\mathcal{C}}^{(\mathtt{C})}$ such that $\mathcal{C} = \{j, k, l\} \in \{0, 1\}^3$ is uniquely described by a set of $\mathtt{H}$ indices. In a binary

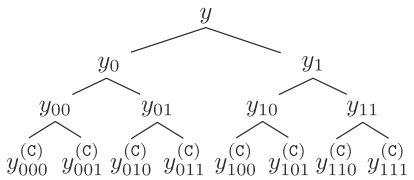


Figure 10: Representation of a 3-layers hierarchical binary decomposition of classes $y_{ijk}^{(\mathtt{C})} \in \{1, 2, \cdots, 8\}$ using the output layer variables $\boldsymbol{y} = [y\, y_0\, y_1\, y_{00}\, y_{01}\, y_{10}\, y_{11}]^\intercal \in \mathbb{R}^7$.

context where $\mathtt{H} = 1$, we can transform a regression problem into a probability for a class $y_i^{\mathtt{C}},\ i \in \{0, 1\}$ by using

$$p(y_i^{\mathtt{C}}|y) = \Phi\left((-1)^i \frac{y}{\alpha}\right),$$

where, $\Phi(\cdot)$ denotes the standard normal CDF, and $\alpha \in \mathbb{R}^+$ is a scaling factor to account for the fact that the standard normal CDF reaches values close to 0 or 1 for inputs close to -3 and 3. Therefore, if the data is initially normalized in range -1 to 1, we need the scaling factor $\alpha \approx 1/3$ in order to accommodate for this discrepancy. In the general case with $\mathtt{K}$-classes, the conditional probability of a class given the output values $\boldsymbol{y}$ is defined by

$$p(y_{\mathcal{C}}^{(\mathtt{C})}|\boldsymbol{y}) = \prod_{h=1}^{\mathtt{H}} \Phi\left((-1)^{\mathcal{C}_h} \frac{[\boldsymbol{y}_{\mathcal{C}}]_h}{\alpha}\right),$$

where $\boldsymbol{y}_{\mathcal{C}} = [y\, y_j\, y_{jk}\, y_{jkl} \cdots]^\intercal \in \mathbb{R}^{\mathtt{H}}$. For the example in Figure 10 where $\mathtt{H} = 3$ layers, it simplifies to $\boldsymbol{y}_{\mathcal{C}} = [y\, y_j\, y_{jk}]^\intercal$, so that

$$p(y_{\{ijk\}}^{(\mathtt{C})}|\boldsymbol{y}) = \Phi\left((-1)^i \frac{y}{\alpha}\right) \cdot \Phi\left((-1)^j \frac{y_j}{\alpha}\right) \cdot \Phi\left((-1)^k \frac{y_{jk}}{\alpha}\right).$$

For the special case where $\boldsymbol{Y}_{\mathcal{C}}|\mathcal{D} \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{Y}_{\mathcal{C}}}, \mathrm{diag}(\boldsymbol{\sigma}^2_{\boldsymbol{Y}_{\mathcal{C}}}))$ follows a Gaussian distribution with diagonal covariance, we can employ the development found in Rasmussen and Williams (2006) in order to obtain a closed-form solution to marginalize the output layer's uncertainty,

$$\begin{aligned} p(y_{\mathcal{C}}^{(\mathtt{C})}|\mathcal{D}) &= \int p(y_{\mathcal{C}}^{(\mathtt{C})}|\boldsymbol{y}_{\mathcal{C}}) \cdot f(\boldsymbol{y}_{\mathcal{C}}|\mathcal{D}) d\boldsymbol{y} \\ &= \prod_{h=1}^{\mathtt{H}} \Phi\left((-1)^{\mathcal{C}_h} \frac{[\boldsymbol{\mu}_{\boldsymbol{Y}_{\mathcal{C}}}]_h}{\sqrt{\alpha^2 + [\boldsymbol{\sigma}^2_{\boldsymbol{Y}_{\mathcal{C}}}]_h}}\right) \end{aligned} \tag{24}$$

Note that in the case where the number of classes $\mathtt{K}$ does not correspond to an integer to the power 2, it is required to normalize the marginal probabilities obtained in Equation 24 in order to account for the unused leaves from the binary tree. During the training phase where we infer the network's parameters from observations $y^{(\mathtt{C})} \in \{1, 2, \cdots, \mathtt{K}\}$, we convert each class into a $\mathtt{H}$-component vector $\boldsymbol{y}_{\mathcal{C}} \in \{-1, 1\}^{\mathtt{H}}$ so that $[\boldsymbol{y}_{\mathcal{C}}]_i = (-1)^{\mathcal{C}_i}$.

Appendix D. Example of F matrices for product-terms indices

Figure 11 presents an example of two successive hidden layers each comprising only two hidden units. The formulation of the the $\mathbf{F}_{\boldsymbol{wa}}^{(j)}$ and in $\mathbf{F}_{\boldsymbol{b}}^{(j)}$ matrices corresponding to the

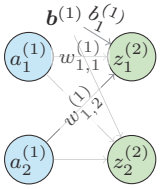


Figure 11: Example of trivial network configuration employed to illustrate the configuration for the matrices $\mathbf{F}_{\boldsymbol{wa}}^{(j)}$ and $\mathbf{F}_{\boldsymbol{b}}^{(j)}$.

network in Figure 11 is

$$\underbrace{\begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \end{bmatrix}}_{\boldsymbol{z}^{(j+1)}} = \underbrace{\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}}_{\mathbf{F}_{\boldsymbol{wa}}^{(j)}} \times \underbrace{\begin{bmatrix} w_{1,1}^{(1)} a_1^{(1)} \\ w_{1,2}^{(1)} a_2^{(1)} \\ w_{2,1}^{(1)} a_1^{(1)} \\ w_{2,2}^{(1)} a_2^{(1)} \end{bmatrix}}_{(\boldsymbol{wa})^{(j)}} + \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{\mathbf{F}_{\boldsymbol{b}}^{(j)}} \times \underbrace{\begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix}}_{\boldsymbol{b}^{(j)}}.$$

Note that the structure of $\mathbf{F}_{\boldsymbol{wa}}^{(j)}$ depends on the ordering of variables.

Appendix E. Experiment configurations and supplementary results for benchmark regression datasets

Table 3 presents the details for the experiments conducted for the benchmark regression datasets. Note that the times presented in the last columns are the average parameter (i.e. weights and biases) inference time in second per folds and the average hyper parameter (i.e. σ_V) optimization time in second. All these experiments were conducted using CPU.

Table 3: Experiment details for the benchmark regression datasets using BLNN. `X`: number of covariates, `L`: number of layers, `A`: number of activation units per layer, `F` number of random training/test folds.

Datasets	`X`	Train #obs.	Test #obs.	`L` × `A`	`F`	Average inference ($\boldsymbol{\theta}$) time per fold (s)	Average optimization (σ_V) time per fold (s)	Average σ_V (20 folds)
Boston	13	455	51	1×50	20	0.6	1.5	0.28
Concrete	8	927	103	1×50	20	0.9	2.3	0.32
Energy	8	691	77	1×50	20	0.6	2.7	0.15
Kin8nm	8	7373	819	1×50	20	6.7	16.7	0.36
Naval	16	11934	1193	1×50	20	13.7	32.6	0.31
Power	4	8611	957	1×50	20	6.6	17.7	0.24
Protein	9	41157	4373	1×100	5	39	53.5	0.74
Wine	11	1439	160	1×50	20	1.4	2.1	0.72
Yacht	6	277	31	1×50	20	0.2	1.4	0.07