Noisy Natural Gradient as Variational Inference

Guodong Zhang [* 1 2]   **Shengyang Sun** [* 1 2]   **David Duvenaud** [1 2]   **Roger Grosse** [1 2]

Abstract

Variational Bayesian neural nets combine the flexibility of deep learning with Bayesian uncertainty estimation. Unfortunately, there is a tradeoff between cheap but simple variational families (e.g. fully factorized) or expensive and complicated inference procedures. We show that natural gradient ascent with adaptive weight noise implicitly fits a variational posterior to maximize the evidence lower bound (ELBO). This insight allows us to train full-covariance, fully factorized, or matrix-variate Gaussian variational posteriors using noisy versions of natural gradient, Adam, and K-FAC, respectively, making it possible to scale up to modern ConvNets. On standard regression benchmarks, our noisy K-FAC algorithm makes better predictions and matches Hamiltonian Monte Carlo's predictive variances better than existing methods. Its improved uncertainty estimates lead to more efficient exploration in active learning, and intrinsic motivation for reinforcement learning.

1. Introduction

Combining deep learning with Bayesian uncertainty estimation has the potential to fit flexible and scalable models that are resistant to overfitting (MacKay, 1992b; Neal, 1995; Hinton & Van Camp, 1993). Stochastic variational inference is especially appealing because it closely resembles ordinary backprop (Graves, 2011; Blundell et al., 2015), but such methods typically impose restrictive factorization assumptions on the approximate posterior, such as fully independent weights. There have been attempts to fit more expressive approximating distributions which capture correlations such as matrix-variate Gaussians (Louizos & Welling, 2016; Sun et al., 2017) or multiplicative normalizing flows (Louizos & Welling, 2017), but fitting such models can be expensive without further approximations.

---

[*]Equal contribution [1]University of Toronto [2]Vector Institute. Correspondence to: Guodong Zhang <gdzhang@cs.toronto.edu>, Shengyang Sun <ssy@cs.toronto.edu>.

In this work, we introduce and exploit a surprising connection between natural gradient descent (Amari, 1998) and variational inference. In particular, several approximate natural gradient optimizers have been proposed which fit tractable approximations to the Fisher matrix to gradients sampled during training (Kingma & Ba, 2014; Martens & Grosse, 2015). While these procedures were described as natural gradient descent on the weights using an *approximate* Fisher matrix, we reinterpret these algorithms as natural gradient on a variational posterior using the *exact* Fisher matrix. Both the weight updates and the Fisher matrix estimation can be seen as natural gradient ascent on a unified evidence lower bound (ELBO), analogously to how Neal and Hinton (Neal & Hinton, 1998) interpreted the E and M steps of Expectation-Maximization (E-M) as coordinate ascent on a single objective.

Using this insight, we give an alternative training method for variational Bayesian neural networks. For a factorial Gaussian posterior, it corresponds to a diagonal natural gradient method with weight noise, and matches the performance of Bayes By Backprop (Blundell et al., 2015), but converges faster. We also present noisy K-FAC, an efficient and GPU-friendly method for fitting a full matrix-variate Gaussian posterior, using a variant of Kronecker-Factored Approximate Curvature (K-FAC) (Martens & Grosse, 2015) with correlated weight noise.

2. Background

2.1. Variational Inference for Bayesian Neural Networks

Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)_{i=1}^n\}$, a Bayesian neural net (BNN) is defined in terms of a prior $p(\mathbf{w})$ on the weights, as well as the likelihood $p(\mathcal{D} \mid \mathbf{w})$. Variational Bayesian methods (Hinton & Van Camp, 1993; Graves, 2011; Blundell et al., 2015) attempt to fit an approximate posterior $q(\mathbf{w})$ to maximize the evidence lower bound (ELBO):

$$\mathcal{L}[q] = \mathbb{E}_q[\log p(\mathcal{D} \mid \mathbf{w})] - \lambda \mathrm{D}_{\mathrm{KL}}(q(\mathbf{w}) \,\|\, p(\mathbf{w})) \quad (1)$$

where λ is a regularization parameter and ϕ are the parameters of the variational posterior. Proper Bayesian inference corresponds to $\lambda = 1$, but other values may work better in practice on some problems.

(a) Fully factorized   (b) Matrix-variate   (c) Block tridiagonal   (d) Full covariance
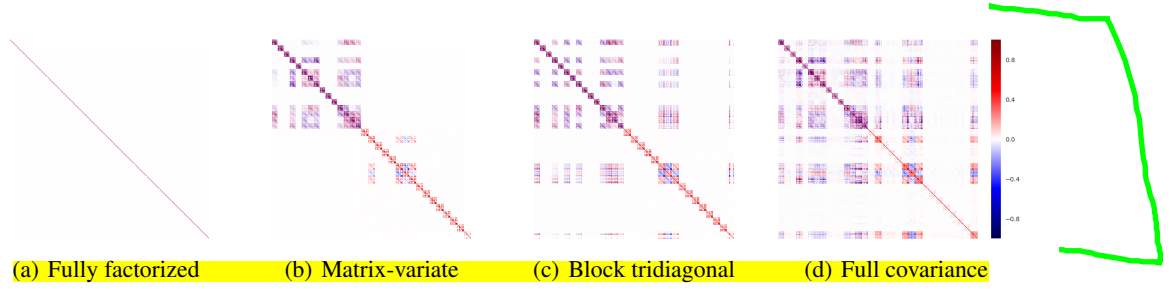
Figure 1: Normalized precision matrices for Gaussian variational posteriors trained using noisy natural gradient. We used a network with 2 hidden layers of 15 units each, trained on the Boston housing dataset.

2.2. Gradient Estimators for Gaussian Distribution

To optimize the ELBO, we must estimate the derivative of eq. (1) w.r.t. variational parameters ϕ. The standard approach uses the pathwise derivative estimator, also known as the reparameterization trick (Williams, 1992; Blundell et al., 2015; Kingma & Welling, 2013; Rezende et al., 2014). However, in the case of Gaussian distribution with parameters $\phi = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}\}$, there is another estimator given by Opper & Archambeau (2009):

$$\nabla_{\boldsymbol{\mu}} \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})}[f(\mathbf{w})] = \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})}[\nabla_{\mathbf{w}} f(\mathbf{w})]$$
$$\nabla_{\boldsymbol{\Sigma}} \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})}[f(\mathbf{w})] = \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})}[\nabla_{\mathbf{w}}^2 f(\mathbf{w})]$$

(2)

which are due to Bonnet (1964) and Price (1958), respectively. Both equations can be proved through integration by parts. In the case of Gaussian distribution, eq. (2) is equivalent to the pathwise derivative estimator for $\boldsymbol{\mu}$.

2.3. Natural Gradient

Natural gradient descent is a second-order optimization method originally proposed by Amari (1997). There are two variants of natural gradient commonly used in machine learning, which do not have standard names, but which we refer to as natural gradient for point estimation (NGPE) and natural gradient for variational inference (NGVI).

In natural gradient for point estimation (NGPE), we assume the neural network computes a predictive distribution $p(y|\mathbf{x}; \mathbf{w})$ and we wish to maximize a cost function $h(\mathbf{w})$, which may be the data log-likelihood. The natural gradient is the direction of steepest ascent in the Fisher information norm, and is given by $\tilde{\nabla}_{\mathbf{w}} h = \mathbf{F}^{-1} \nabla_{\mathbf{w}} h$, where $\mathbf{F} = \text{Cov}_{\mathbf{x} \sim p_{\mathcal{D}}, y \sim p(y|\mathbf{x}, \mathbf{w})}[\nabla_{\mathbf{w}} \log p(y|\mathbf{x}, \mathbf{w})]$, and the covariance is with respect to $\mathbf{x}$ sampled from the data distribution and y sampled from the model's predictions. NGPE is typically justified as a way to speed up optimization; see Martens (2014) for a comprehensive overview.

We now describe natural gradient for variational inference (NGVI) in the context of BNNs. We wish to fit the parameters of a variational posterior $q(\mathbf{w})$ to maximize the ELBO (eq. (1)). Analogously to the point estimation set-

ting, the natural gradient is defined as $\tilde{\nabla}_{\phi} \mathcal{L} = \mathbf{F}^{-1} \nabla_{\phi} \mathcal{L}$; but in this case, $\mathbf{F}$ is the Fisher matrix of q, i.e. $\mathbf{F} = \text{Cov}_{\mathbf{w} \sim q}[\nabla_{\phi} \log q(\mathbf{w}; \phi)]$. Note that in contrast with point estimation, $\mathbf{F}$ is a metric on ϕ, rather than $\mathbf{w}$, and its definition doesn't directly involve the data. Interestingly, because q is chosen to be tractable, the natural gradient can be computed exactly, and in many cases is even simpler than the ordinary gradient.

In general, NGPE and NGVI need not behave similarly; however, in Section 3, we show that in the case of Gaussian variational posteriors, the two are closely related.

2.4. Kronecker-Factored Approximate Curvature

As modern neural networks may contain millions of parameters, computing and storing the exact Fisher matrix and its inverse is impractical. Kronecker-factored approximate curvature (K-FAC) (Martens & Grosse, 2015) uses a Kronecker-factored approximation to the Fisher to perform efficient approximate natural gradient updates. Considering lth layer in the neural network whose input activations are $\mathbf{a}_l \in \mathbb{R}^{n_1}$, weight $\mathbf{W}_l \in \mathbb{R}^{n_1 \times n_2}$, and output $\mathbf{s}_l \in \mathbb{R}^{n_2}$, we have $\mathbf{s}_l = \mathbf{W}_l^T \mathbf{a}_l$. For simplicity, we define the following additional notation:

$$\mathcal{D}v = \nabla_v \log p(y|\mathbf{x}, \mathbf{w}) \text{ and } \mathbf{g}_l = \mathcal{D}\mathbf{s}_l$$

Therefore, the weight gradient is $\mathcal{D}\mathbf{W}_l = \mathbf{a}_l \mathbf{g}_l^T$. With this gradient formula, K-FAC decouples this layer's fisher matrix $\mathbf{F}_l$ by approximating $\mathbf{a}_l$ and $\mathbf{g}_l$ as independent:

$$\mathbf{F}_l = \mathbb{E}[\text{vec}\{\mathcal{D}\mathbf{W}_l\}\text{vec}\{\mathcal{D}\mathbf{W}_l\}^\top] = \mathbb{E}[\mathbf{g}_l \mathbf{g}_l^\top \otimes \mathbf{a}_l \mathbf{a}_l^\top]$$
$$\approx \mathbb{E}[\mathbf{g}_l \mathbf{g}_l^\top] \otimes \mathbb{E}[\mathbf{a}_l \mathbf{a}_l^\top] = \mathbf{S}_l \otimes \mathbf{A}_l = \tilde{\mathbf{F}}_l$$

(3)

Further, assuming between-layer independence, the whole fisher matrix $\tilde{\mathbf{F}}$ can be approximated as block diagonal consisting of layerwise fisher matrices $\tilde{\mathbf{F}}_l$. Decoupling $\tilde{\mathbf{F}}_l$ into $\mathbf{A}_l$ and $\mathbf{S}_l$ not only avoids the quadratic storage cost of the exact Fisher, but also enables tractable computation of the approximate natural gradient:

$$\tilde{\mathbf{F}}_l^{-1} \text{vec}\{\nabla_{\mathbf{W}_l} h\} = \mathbf{S}_l^{-1} \otimes \mathbf{A}_l^{-1} \text{vec}\{\nabla_{\mathbf{W}_l} h\}$$
$$= \text{vec}[\mathbf{A}_l^{-1} \nabla_{\mathbf{W}_l} h \mathbf{S}_l^{-1}]$$

(4)

As shown by eq. (4), computing natural gradient using K-FAC only consists of matrix transformations comparable to size of $\mathbf{W}_l$, making it very efficient.

3. Variational Inference using Noisy Natural Gradient

In this section, we draw a surprising relationship between natural gradient for point estimation (NGPE) of the weights of a neural net, and natural gradient for variational inference (NGVI) of a Gaussian posterior. (These terms are explained in Section 2.3.) In particular, we show that the NGVI updates can be approximated with a variant of NGPE with adaptive weight noise which we term *Noisy Natural Gradient (NNG)*. This insight allows us to train variational posteriors with a variety of structures using noisy versions of existing optimization algorithms (see Figure 1).

In NGVI, our goal is to maximize the ELBO $\mathcal{L}$ (eq. (1)) with respect to the parameters ϕ of a variational posterior distribution $q(\mathbf{w})$. We assume q is a multivariate Gaussian parameterized by $\phi = (\mu, \Sigma)$. Building on eq. (2), we determine the natural gradient of the ELBO with respect to μ and the precision matrix $\Lambda = \Sigma^{-1}$ (see supplement for details):

$$\tilde{\nabla}_{\mu}\mathcal{L} = \Lambda^{-1}\mathbb{E}_q\left[\nabla_{\mathbf{w}}\log p(\mathcal{D}\,|\,\mathbf{w}) + \lambda\nabla_{\mathbf{w}}\log p(\mathbf{w})\right] \quad (5)$$

$$\tilde{\nabla}_{\Lambda}\mathcal{L} = -\mathbb{E}_q\left[\nabla_{\mathbf{w}}^2\log p(\mathcal{D}\,|\,\mathbf{w}) + \lambda\nabla_{\mathbf{w}}^2\log p(\mathbf{w})\right] - \lambda\Lambda$$

We make several observations. First, the term inside the expectation in eq. (5) is the gradient for MAP estimation of $\mathbf{w}$. Second, the update for μ is preconditioned by Λ^{-1}, which encourages faster movement in directions of higher posterior uncertainty. Finally, the fixed point equation for Λ is given by

$$\Lambda = -\mathbb{E}_q\left[\frac{1}{\lambda}\nabla_{\mathbf{w}}^2\log p(\mathcal{D}\,|\,\mathbf{w}) + \nabla_{\mathbf{w}}^2\log p(\mathbf{w})\right] \quad (6)$$

Hence, if $\lambda = 1$, Λ will tend towards the expected Hessian of $-\log p(\mathbf{w}, \mathcal{D})$, so the update rule for μ will somewhat resemble a Newton-Raphson update. For simplicity, we further assume a spherical Gaussian prior $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \eta\mathbf{I})$, then $\nabla_{\mathbf{w}}^2\log p(\mathbf{w}) = -\eta^{-1}\mathbf{I}$. In each iteration, we sample $(\mathbf{x}, y) \sim p_{\mathcal{D}}$ and $\mathbf{w} \sim q$ and apply a stochastic natural gradient update based on eq. (5):

$$\mu \leftarrow \mu + \alpha\Lambda^{-1}\left[\mathcal{D}\mathbf{w} - \frac{\lambda}{N\eta}\mathbf{w}\right] \quad (7)$$

$$\Lambda \leftarrow \left(1 - \frac{\lambda\beta}{N}\right)\Lambda - \beta\left[\nabla_{\mathbf{w}}^2\log p(y|\mathbf{x}, \mathbf{w}) - \frac{\lambda}{N\eta}\mathbf{I}\right]$$

where α and β are separate learning rates for μ and Λ, and N is the number of training examples. Roughly speaking, the update rule for Λ corresponds to an exponential moving average of the Hessian, and the update rule for μ is a stochastic Newton step using Λ.

This update rule has two problems. First, the log-likelihood Hessian may be hard to compute, and is undefined at some points for neural nets which use not-everywhere-differentiable activation functions such as ReLU. Second, if the negative log-likelihood is non-convex (as is the case for neural networks), the Hessian could have negative eigenvalues. so the update may result in Λ which is not positive semidefinite. We circumvent both of these problems by approximating the negative log-likelihood Hessian with the NGPE Fisher matrix $\mathbf{F} = \mathrm{Cov}_{\mathbf{x}\sim p_{\mathcal{D}}, y\sim p(y|\mathbf{x}, \mathbf{w})}(\mathcal{D}\mathbf{w})$:

$$\Lambda \leftarrow \left(1 - \frac{\lambda\beta}{N}\right)\Lambda + \beta\left[\mathcal{D}\mathbf{w}\mathcal{D}\mathbf{w}^\top + \frac{\lambda}{N\eta}\mathbf{I}\right] \quad (8)$$

This approximation guarantees that Λ is positive semidefinite, and it allows for tractable approximations such as K-FAC (see below). In the context of BNNs, approximating the log-likelihood Hessian with the Fisher was first proposed by Graves (2011), so we refer to it as the *Graves approximation*. In the case where the output layer of the network represents the natural parameters of an exponential family distribution (as is typical in regression or classification), the Graves approximation can be justified in terms of the generalized Gauss-Newton approximation to the Hessian; see Martens (2014) for details.[1]

3.1. Simplifying the Update Rules

We have now derived a stochastic natural gradient update rule for Gaussian variational posteriors. In this section, we rewrite the update rules in order to disentangle hyperparameters and highlight relationships with NGPE. First, if the prior variance η is fixed[2], then Λ is a damped version of the moving average of the Fisher matrix and we can rewrite the update eq. (8):

$$\Lambda = \frac{N}{\lambda}\bar{\mathbf{F}} + \eta^{-1}\mathbf{I}$$
$$\bar{\mathbf{F}} \leftarrow (1 - \tilde{\beta})\bar{\mathbf{F}} + \tilde{\beta}\mathcal{D}\mathbf{w}\mathcal{D}\mathbf{w}^\top \quad (9)$$

In eq. (9), we avoid an awkward interaction between the KL weight λ and the learning rates α, β by writing the update rules in terms of alternative learning rates $\tilde{\alpha} = \alpha\lambda/N$ and $\tilde{\beta} = \beta\lambda/N$. We also rewrite the update rule for μ:

$$\mu \leftarrow \mu + \tilde{\alpha}\left(\bar{\mathbf{F}} + \frac{\lambda}{N\eta}\mathbf{I}\right)^{-1}\left[\mathcal{D}\mathbf{w} - \frac{\lambda}{N\eta}\mathbf{w}\right] \quad (10)$$

---

[1]eq. (8) leaves ambiguous what distribution the gradients are sampled from. Throughout our experiments, we sample the targets from the model's predictions, as done in K-FAC (Martens & Grosse, 2015). The resulting $\mathbf{F}$ is known as the *true Fisher*. The alternative is to use the SGD gradients, giving the *empirical Fisher*. The true Fisher is a better approximation to the Hessian (Martens, 2014).

[2]For simplicity, we assume the prior is a spherical Gaussian and its variance η is fixed. Otherwise, we can keep an exponential moving average of the prior Hessian.

Observe that if $\boldsymbol{\mu}$ is viewed as a point estimate of the weights, this update rule resembles NGPE with an exponential moving average of the Fisher matrix. The differences are that the Fisher matrix $\mathbf{F}$ is damped by adding $\frac{\lambda}{N\eta}\mathbf{I}$, and that the weights are sampled from q, which is a Gaussian with covariance $\boldsymbol{\Sigma} = (\frac{N}{\lambda}\bar{\mathbf{F}} + \eta^{-1}\mathbf{I})^{-1}$. Because our update rule so closely resembles NGPE with correlated weight noise, we refer to this method as Noisy Natural Gradient (NNG).

3.2. Damping

Interestingly, in second-order optimization, it is very common to dampen the updates by adding a multiple of the identity matrix to the curvature before inversion in order to compensate for errors in the quadratic approximation to the cost. NNG automatically achieves this effect, with the strength of the damping being $\lambda/N\eta$; we refer to this as *intrinsic damping*. In practice, it may be advantageous to add additional *extrinsic damping* for purposes of stability.

3.3. Fitting Fully Factorized Gaussian Posteriors with Noisy Adam

The discussion so far has concerned NGVI updates for a full covariance Gaussian posterior. Unfortunately, the number of parameters needed to represent a full covariance Gaussian is of order $(\dim \mathbf{w})^2$. Since it can be in the millions even for a relatively small network, representing a full covariance Gaussian is impractical. There has been much work on tractable approximations to second-order optimization. In the context of NNG, imposing structure on $\mathbf{F}$ also imposes structure on the form of the variational posterior. We now discuss two kinds of structure one can impose.

Perhaps the simplest approach is to approximate $\mathbf{F}$ with a diagonal matrix $\text{diag}(\mathbf{f})$, as done by Adagrad (Duchi et al., 2011) and Adam (Kingma & Ba, 2014). For our NNG approach, this yields the following updates:

$$\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} + \tilde{\alpha}\left[\mathcal{D}\mathbf{w} - \frac{\lambda}{N\eta}\mathbf{w}\right] \Big/ \left(\bar{\mathbf{f}} + \frac{\lambda}{N\eta}\right)$$
$$\bar{\mathbf{f}} \leftarrow (1-\tilde{\beta})\bar{\mathbf{f}} + \tilde{\beta}\mathcal{D}\mathbf{w}^2 \tag{11}$$

These update rules are similar in spirit to methods such as Adam, but with the addition of adaptive weight noise. We note that these update rules also differ from Adam in some details: (1) Adam keeps exponential moving averages of the gradients, which is equivalent to momentum, and (2) Adam applies the square root to the entries of $\mathbf{f}$ in the denominator. We define *noisy Adam* by adding momentum term to be consistent with Adam. We regard difference (2) inessential. The choice of squaring or divison may affect optimization performance, but they don't change the fixed points, i.e. they are fitting the same functional form of the variational posterior using the same variational objective. The full procedure is given in Alg. 1.

3.4. Fitting Matrix Variate Gaussian Posteriors with Noisy K-FAC

There has been much interest in fitting BNNs with matrix-variate Gaussian (MVG) posteriors[3] in order to compactly capture posterior correlations between different weights (Louizos & Welling, 2016; Sun et al., 2017). Let $\mathbf{W}_l$ denote the weights for one layer of a fully connected network. An MVG distribution is a Gaussian distribution whose covariance is a Kronecker product, i.e. $\mathcal{MN}(\mathbf{W}; \mathbf{M}, \boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2) = \mathcal{N}(\text{vec}(\mathbf{W}); \text{vec}(\mathbf{M}), \boldsymbol{\Sigma}_2 \otimes \boldsymbol{\Sigma}_1)$. MVGs are potentially powerful due to their compact representation[4] of posterior covariances between weights. However, fitting MVG posteriors is difficult, since computing the gradients and enforcing the positive semidefinite constraint for $\boldsymbol{\Sigma}_1$ and $\boldsymbol{\Sigma}_2$ typically requires expensive matrix operations such as inversion. Therefore, existing methods for fitting MVG posteriors typically impose additional structure such as diagonal covariance (Louizos & Welling, 2016) or products of Householder transformations (Sun et al., 2017) to ensure efficient updates.

We observe that K-FAC (Martens & Grosse, 2015) uses a Kronecker-factored approximation to the Fisher matrix for each layer's weights, as in eq. (3). By plugging this approximation in to eq. (9), we obtain an MVG posterior. In more detail, each block obeys the Kronecker factorization $\mathbf{S}_l \otimes \mathbf{A}_l$, where $\mathbf{A}_l$ and $\mathbf{S}_l$ are the covariance matrices of the activations and pre-activation gradients, respectively. K-FAC estimates $\mathbf{A}_l$ and $\mathbf{S}_l$ online using exponential moving averages which, conveniently for our purposes, are closely analogous to the exponential moving averages defining $\bar{\mathbf{F}}$ in eq. (9):

$$\bar{\mathbf{A}}_l \leftarrow (1-\tilde{\beta})\bar{\mathbf{A}}_l + \tilde{\beta}\mathbf{a}_l\mathbf{a}_l^\top$$
$$\bar{\mathbf{S}}_l \leftarrow (1-\tilde{\beta})\bar{\mathbf{S}}_l + \tilde{\beta}\mathcal{D}\mathbf{s}_l\mathcal{D}\mathbf{s}_l^\top \tag{12}$$

Conveniently, because these factors are estimated from the empirical covariances, they (and hence also $\boldsymbol{\Lambda}$) are automatically positive semidefinite.

Plugging the above formulas into eq. (9) does not quite yield an MVG posterior due to the addition of the prior Hessian. In general, there may be no compact representation of $\boldsymbol{\Lambda}$. However, for spherical Gaussian priors[5], we can approximate $\boldsymbol{\Sigma}$ using a trick proposed by Martens & Grosse (2015) in the context of damping. In this way, the covariance

---

[3]When we refer to a BNN with an "MVG posterior", we mean that the weights in different layers are independent, and the weights for each layer follow an MVG distribution.

[4]If $\mathbf{W}$ is of size $m \times n$, then the MVG covariance requires approximately $m^2/2 + n^2/2$ parameters to represent, in contrast with a full covariance matrix over $\mathbf{w}$, which would require $m^2n^2/2$.

[5]We consider spherical Gaussian priors for simplicity, but this trick can be extended to any prior whose Hessian is Kronecker-factored, such as group sparsity.

Algorithm 1 Noisy Adam. Differences from standard Adam are shown in blue.

Require: α: Stepsize
Require: β_1, β_2: Exponential decay rates for updating $\boldsymbol{\mu}$ and the Fisher $\mathbf{F}$
Require: $\lambda, \eta, \gamma_{\text{ex}}$: KL weighting, prior variance, extrinsic damping term
  $\mathbf{m} \leftarrow \mathbf{0}$
  Calculate the intrinsic damping term $\gamma_{\text{in}} = \frac{\lambda}{N\eta}$, total damping term $\gamma = \gamma_{\text{in}} + \gamma_{\text{ex}}$
  while stopping criterion not met **do**
    $\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}, \frac{\lambda}{N}\text{diag}(\mathbf{f} + \gamma_{\text{in}})^{-1})$
    $\mathbf{v} \leftarrow \nabla_{\mathbf{w}} \log p(y|\mathbf{x}, \mathbf{w}) - \gamma_{\text{in}} \cdot \mathbf{w}$
    $\mathbf{m} \leftarrow \beta_1 \cdot \mathbf{m} + (1 - \beta_1) \cdot \mathbf{v}$ (Update momentum)
    $\mathbf{f} \leftarrow \beta_2 \cdot \mathbf{f} + (1 - \beta_2) \cdot (\nabla_{\mathbf{w}} \log p(y|\mathbf{x}, \mathbf{w})^2$
    $\tilde{\mathbf{m}} \leftarrow \mathbf{m}/(1 - \beta_1^k)$
    $\hat{\mathbf{m}} \leftarrow \tilde{\mathbf{m}}/(\mathbf{f} + \gamma)$
    $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} + \alpha \cdot \hat{\mathbf{m}}$ (Update parameters)
  end while

$\boldsymbol{\Sigma}_l$ decomposes as the Kronecker product of two terms:

$$\boldsymbol{\Sigma}_l = \frac{\lambda}{N}[\mathbf{S}_l^\gamma]^{-1} \otimes [\mathbf{A}_l^\gamma]^{-1} \tag{13}$$

$$\triangleq \frac{\lambda}{N}\left(\bar{\mathbf{S}}_l + \frac{1}{\pi_l}\sqrt{\frac{\lambda}{N\eta}}\mathbf{I}\right)^{-1} \otimes \left(\bar{\mathbf{A}}_l + \pi_l\sqrt{\frac{\lambda}{N\eta}}\mathbf{I}\right)^{-1}$$

This factorization corresponds to a matrix-variate Gaussian posterior $\mathcal{MN}(\mathbf{W}_l; \mathbf{M}_l, \frac{\lambda}{N}[\mathbf{A}_l^\gamma]^{-1}, [\mathbf{S}_l^\gamma]^{-1})$, where the λ/N factor is arbitrarily assigned to the first factor. We refer to this BNN training method as *noisy K-FAC*. The full algorithm is given as Alg. 2.

3.5. Block Tridiagonal Covariance

Both the fully factorized and MVG posteriors assumed independence between layers. However, in practice the weights in different layers can be tightly coupled. To better capture these dependencies, we propose to approximate $\mathbf{F}$ using the block tridiagonal approximation from Martens & Grosse (2015). The resulting posterior covariance is block tridiagonal, so it accounts for dependencies between adjacent layers. The noisy version of block tridiagonal K-FAC is completely analogous to the block diagonal version, but since the approximation is rather complicated, we refer the reader to Martens & Grosse (2015) for details.

4. Related Work

Variational inference was first applied to neural networks by Peterson (1987) and Hinton & Van Camp (1993). More recently, Graves (2011) proposed a practical method for variational inference with fully factorized Gaussian posteriors which used a simple (but biased) gradient estimator.

Algorithm 2 Noisy K-FAC. Subscript l denotes layers, $\mathbf{w}_l = \text{vec}(\mathbf{W}_l)$, and $\boldsymbol{\mu}_l = \text{vec}(\mathbf{M}_l)$. We assume zero momentum for simplicity. Differences from standard K-FAC are shown in blue.

Require: α: stepsize
Require: β: exponential moving average parameter
Require: $\lambda, \eta, \gamma_{\text{ex}}$: KL weighting, prior variance, extrinsic damping term
Require: stats and inverse update intervals T_{stats} and T_{inv}
  $k \leftarrow 0$ and initialize $\{\boldsymbol{\mu}_l\}_{l=1}^L, \{\mathbf{S}_l\}_{l=1}^L, \{\mathbf{A}_l\}_{l=1}^L$
  Calculate the intrinsic damping term $\gamma_{\text{in}} = \frac{\lambda}{N\eta}$, total damping term $\gamma = \gamma_{\text{in}} + \gamma_{\text{ex}}$
  while stopping criterion not met **do**
    $k \leftarrow k + 1$
    $\mathbf{W}_l \sim \mathcal{MN}(\mathbf{M}_l, \frac{\lambda}{N}[\mathbf{A}_l^{\gamma_{\text{in}}}]^{-1}, [\mathbf{S}_l^{\gamma_{\text{in}}}]^{-1})$
    if $k \equiv 0 \pmod{T_{\text{stats}}}$ **then**
      Update the factors $\{\mathbf{S}_l\}_{l=1}^L, \{\mathbf{A}_l\}_{l=0}^{L-1}$ using eq. (12)
    end if
    if $k \equiv 0 \pmod{T_{\text{inv}}}$ **then**
      Calculate the inverses $\{[\mathbf{S}_l^\gamma]^{-1}\}_{l=1}^L, \{[\mathbf{A}_l^\gamma]^{-1}\}_{l=0}^{L-1}$ using eq. (13).
    end if
    $\mathbf{V}_l = \nabla_{\mathbf{W}_l} \log p(y|\mathbf{x}, \mathbf{w}) - \gamma_{\text{in}} \cdot \mathbf{W}_l$
    $\mathbf{M}_l \leftarrow \mathbf{M}_l + \alpha[\mathbf{A}_l^\gamma]^{-1}\mathbf{V}_l[\mathbf{S}_l^\gamma]^{-1}$
  end while

Improving on that work, Blundell et al. (2015) proposed a unbiased gradient estimator using the reparameterization trick of Kingma & Welling (2013). Kingma et al. (2015) observed that variance of stochastic gradients can be significantly reduced by local reparameterization trick where global uncertainty in the weights is translated into local uncertainty in the activations.

There has also been much work on modeling the correlations between weights using more complex Gaussian variational posteriors. Louizos & Welling (2016) introduced the matrix variate Gaussian posterior as well as a Gaussian process approximation. Sun et al. (2017) decoupled the correlations of a matrix variate Gaussian posterior to unitary transformations and factorial Gaussian. Inspired by the idea of normalizing flows in latent variable models (Rezende & Mohamed, 2015), Louizos & Welling (2017) applied normalizing flows to auxiliary latent variables to produce more flexible approximate posteriors.

Since natural gradient was proposed by Amari (1998), there has been much work on tractable approximations. Hoffman et al. (2013) observed that for exponential family posteriors, the exact natural gradient could be tractably computed using stochastic versions of variational Bayes E-M updates. Martens & Grosse (2015) proposed K-FAC for performing efficient natural gradient optimization in deep neural networks. Following on that work, K-FAC has been adopted in

Table 1: Averaged test RMSE and log-likelihood for the regression benchmarks.

DATASET	TEST RMSE				TEST LOG-LIKELIHOOD			
	BBB	PBP	NNG-FFG	NNG-MVG	BBB	PBP	NNG-FFG	NNG-MVG
BOSTON	3.171±0.149	3.014±0.180	3.031±0.155	**2.742±0.125**	-2.602±0.031	-2.574±0.089	-2.558±0.032	**-2.446±0.029**
CONCRETE	5.678±0.087	5.667±0.093	5.613±0.113	**5.019±0.127**	-3.149±0.018	-3.161±0.019	-3.145±0.023	**-3.039±0.025**
ENERGY	0.565±0.018	1.804±0.048	0.839±0.046	**0.485±0.023**	-1.500±0.006	-2.042±0.019	-1.629±0.020	**-1.421±0.005**
KIN8NM	0.080±0.001	0.098±0.001	0.079±0.001	**0.076±0.001**	1.111±0.007	0.896±0.006	1.112±0.008	**1.148±0.007**
NAVAL	**0.000±0.000**	0.006±0.000	0.001±0.000	**0.000±0.000**	6.143±0.032	3.731±0.006	6.231±0.041	**7.079±0.034**
POW. PLANT	4.023±0.036	4.124±0.035	4.002±0.039	**3.886±0.041**	-2.807±0.010	-2.837±0.009	-2.803±0.010	**-2.776±0.011**
PROTEIN	4.321±0.017	4.732±0.013	4.380±0.016	**4.097±0.009**	-2.882±0.004	-2.973±0.003	-2.896±0.004	**-2.836±0.002**
WINE	0.643±0.012	**0.635±0.008**	0.644±0.011	0.637±0.011	-0.977±0.017	**-0.968±0.014**	-0.976±0.016	-0.969±0.014
YACHT	1.174±0.086	1.015±0.054	1.289±0.069	**0.979±0.077**	-2.408±0.007	**-1.634±0.016**	-2.412±0.006	-2.316±0.006
YEAR	9.076±NA	**8.879±NA**	9.071±NA	8.885±NA	-3.614±NA	-3.603±NA	-3.620±NA	**-3.595±NA**

many tasks (Grosse & Martens, 2016; Wu et al., 2017) to gain optimization benefits, and was shown to be amenable to distributed computation (Ba et al., 2017).

Khan et al. (2017a) independently derived a stochastic Newton update similar to eq. (5). Their focus was on variational optimization (VO) (Staines & Barber, 2012) which can be seen as a special case of variational inference by omitting KL term, and they only derived the version of diagonal approximation (see Section 3.3). Assuming the variational distribution is Gaussian distribution, we can extend NNG to VO by only modifying the update rule of Fisher to keep a running sum of individual Fisher.

Concurrently, Khan et al. (2017b) also found the relationship between natural gradient and variational inference and derived Vprop by adding weight noise to RMSprop, which essentially resembles noisy Adam.

5. Experiments

In this section, we conducted a series of experiments to investigate the following questions: (1) How does noisy natural gradient (NNG) compare with existing methods in terms of prediction performance? (2) Is NNG able to scale to large dataset and modern-size convolutional neural network? (3) Can NNG achieve better uncertainty estimates? (4) Does it enable more efficient exploration in active learning and reinforcement learning?

Our method with a full-covariance multivariate Gaussian, a fully-factorized Gaussian, a matrix-variate Gaussian and block-tridiagonal posterior are denoted as NNG-full, NNG-FFG (noise Adam), NNG-MVG (noisy K-FAC) and NNG-BlkTri, respectively.

5.1. Regression

We first experimented with regression datasets from the UCI collection (Asuncion & Newman, 2007). All experiments used networks with one hidden layer unless stated otherwise. We compared our method with Bayes By Backprop

Table 2: Classification accuracy on CIFAR10 with modified VGG16. **[D]** denotes data augmentation including horizontal flip and random crop while **[B]** denotes Batch Normalization. We leave **[N/A]** for BBB and noisy Adam with BN since they are extremely unstable and work only with a very small λ.

METHOD	NETWORK	TEST ACCURACY			
			D	B	D + B
SGD	VGG16	81.79	88.35	85.75	91.39
KFAC	VGG16	82.39	88.89	86.86	**92.13**
BBB	VGG16	82.82	88.31	N/A	N/A
NOISY-ADAM	VGG16	82.68	88.23	N/A	N/A
NOISY-KFAC	VGG16	**85.52**	**89.35**	**88.22**	92.01

(BBB) (Blundell et al., 2015), probabilistic backpropagation (PBP) with a factorial gaussian posterior (Hernández-Lobato & Adams, 2015). The results for PBP_MV (Sun et al., 2017) and VMG (Louizos & Welling, 2016) can be found in supplement.

Following previous work (Hernández-Lobato & Adams, 2015; Louizos & Welling, 2016), we report the standard metrics including root mean square error (RMSE) and test log-likelihood. The results are summarized in Table 1. As we can see from the results, our NNG-FFG performed similarly to BBB (Blundell et al., 2015), indicating that the Graves approximation did not cause a performance hit. Our NNG-MVG method achieved substantially better RMSE and log-likelihoods than BBB and PBP due to the more flexible posterior. Moreover, NNG-MVG outperforms PBP_MV (Sun et al., 2017) on all datasets other than Yacht and Year, though PBP_MV also uses matrix variate Gaussian posterior.

5.2. Classification

To evaluate the scalability of our method to large networks, we applied noisy K-FAC to a modified version of the VGG16[6] network (Simonyan & Zisserman, 2014) and

---

[6]The detailed network architecture is 32-32-M-64-64-M-128-128-128-M-256-256-256-M-256-256-256-M-FC10, where each number represents the number of filters in a convolutional layer, and M denotes max-pooling.

Table 3: Average test RMSE in active learning.

DATASET	PBP_R	PBP_A	NNG-FFG_R	NNG-FFG_A	NNG-MVG_R	NNG-MVG_A	HMC_R	HMC_A
BOSTON	6.716±0.500	5.480±0.175	5.911±0.250	5.435±0.132	5.831±0.177	5.220±0.132	5.750±0.222	**5.156±0.150**
CONCRETE	12.417±0.392	11.894±0.254	12.583±0.168	12.563±0.142	12.301±0.203	11.671±0.175	**10.564±0.198**	11.484±0.191
ENERGY	3.743±0.121	3.399±0.064	4.011±0.087	3.761±0.068	3.635±0.084	3.211±0.076	3.264±0.067	**3.118±0.062**
KIN8NM	0.259±0.006	0.254±0.005	0.246±0.004	0.252±0.003	0.243±0.003	0.244±0.003	0.226±0.004	**0.223±0.003**
NAVAL	0.015±0.000	0.016±0.000	0.013±0.000	0.013±0.000	0.010±0.000	**0.009±0.000**	0.013±0.000	0.012±0.000
POW. PLANT	5.312±0.108	5.068±0.082	5.812±0.119	5.423±0.111	5.377±0.133	4.974±0.078	5.229±0.097	**4.800±0.074**
WINE	0.945±0.044	0.809±0.011	0.730±0.011	0.748±0.008	0.752±0.014	0.746±0.009	**0.740±0.011**	0.749±0.010
YACHT	5.388±0.339	4.508±0.158	7.381±0.309	6.583±0.264	7.192±0.280	6.371±0.204	4.644±0.237	**3.211±0.120**

tested it on CIFAR10 benchmark (Krizhevsky, 2009). It is straightforward to incorporate noisy K-FAC into convolutional layers by considering them using Kronecker Factors for Convolution Grosse & Martens (2016). We compared our method to SGD with momentum, K-FAC and BBB in terms of test accuracy. Results are shown in Table 2. Noisy K-FAC achieves the highest accuracy on all configurations except where both data augmentation and Batch Normalization (BN) (Ioffe & Szegedy, 2015) are used. When no extra regularization used, noisy K-FAC shows a gain of 3% (85.52% versus 82.39%).

We observed that point estimates tend to make poorly calibrated predictions, as shown in Figure 2. By contrast, models trained with noisy K-FAC are well-calibrated (i.e. the bars align roughly along the diagonal), which benefits interpretability.

We note that noisy K-FAC imposes a weight decay term intrinsically. To check that this by itself doesn't explain the performance gains, we modified K-FAC to use weight decay of the same magnitude. K-FAC with this weight decay setting achieves 83.51% accuracy. However, as shown in Table 2, noisy K-FAC achieves 85.52%, demonstrating the importance of adaptive weight noise.

5.3. Active Learning

One particularly promising application of uncertainty estimation is to guiding an agent's exploration towards part of a space which it's most unfamiliar with. We have evaluated our BNN algorithms in two instances of this general approach: active learning, and intrinsic motivation for reinforcement learning. The next two sections present experiments in these two domains, respectively.

In the simplest active learning setting (Settles, 2010), an algorithm is given a set of unlabeled examples and, in each round, chooses one unlabeled example to have labeled. A classic Bayesian approach to active learning is the information gain criterion (MacKay, 1992a), which in each step attempts to achieve the maximum reduction in posterior entropy. Under the assumption of i.i.d. Gaussian noise, this is equivalent to choosing the unlabeled example with the largest predictive variance. We first investigated how ac-
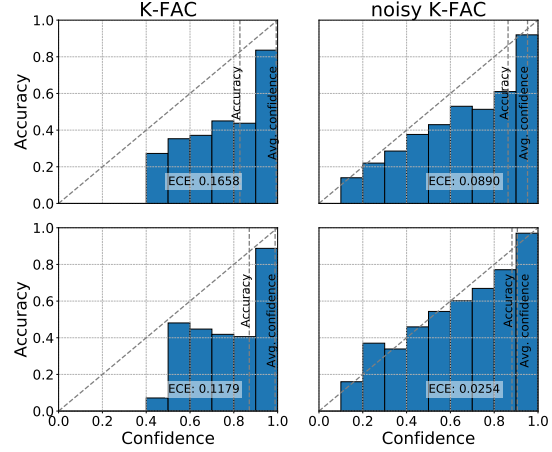

Figure 2: Reliability diagrams (Niculescu-Mizil & Caruana, 2005; Guo et al., 2017) for K-FAC (left) and noisy K-FAC (right) on CIFAR10. Reliability diagrams show accuracy as a function of confidence. Models trained without BN (top) and with BN (bottom). ECE = Expected Calibration Error (Guo et al., 2017); smaller is better.

curately each of the algorithms could estimate predictive variances. In each trial, we randomly selected 20 labeled training examples and 100 unlabeled examples; we then computed each algorithm's posterior predictive variances for the unlabeled examples. 10 independent trials were run. As is common practice, we treated the predictive variance of HMC as the "ground truth" predictive variance. Table 4 reports the average and standard error of Pearson correlations between the predictive variances of each algorithm and those of HMC. In all of the datasets, our two methods NNG-MVG and NNG-BlkTri match the HMC predictive variances significantly better than the other approaches, and NNG-BlkTri consistently matches them slightly better than NNG-MVG due to the more flexible variational posterior.

Next, we evaluated the performance of all methods on active learning, following the protocol of Hernández-Lobato & Adams (2015). As a control, we evaluated each algorithm with labeled examples selected uniformly at random; this is denoted with the _R suffix. Active learning results are denoted with the _A suffix. The average test RMSE for
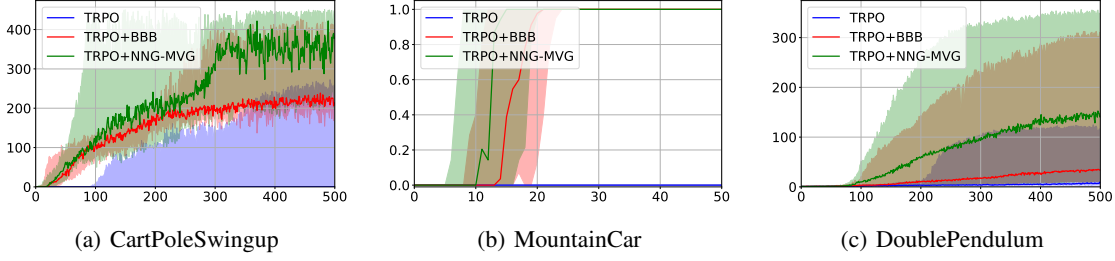
(a) CartPoleSwingup

(b) MountainCar

(c) DoublePendulum

Figure 3: Performance of **[TRPO]** TRPO baseline with Gaussian control noise, **[TRPO+BBB]** VIME baseline with BBB dynamics network, and **[TRPO+NNG-MVG]** VIME with NNG-MVG dynamics network (ours). The darker-colored lines represent the median performance in 10 different random seeds while the shaded area show the interquartile range.

Table 4: Pearson correlation of each algorithm's predictive variances with those of HMC.

DATASET	PBP	NNG-FFG	NNG-MVG	NNG-BLKTRI
BOSTON	0.761±0.032	0.718±0.035	**0.891±0.021**	0.889±0.024
CONCRETE	0.817±0.028	0.811±0.028	0.913±0.010	**0.922±0.006**
ENERGY	0.471±0.076	0.438±0.075	0.617±0.087	**0.646±0.088**
KIN8NM	0.587±0.021	0.659±0.015	0.731±0.021	**0.759±0.023**
NAVAL	0.270±0.098	0.321±0.087	0.596±0.073	**0.598±0.070**
POW. PLANT	0.509±0.068	0.618±0.050	0.829±0.020	**0.853±0.020**
WINE	0.883±0.042	0.918±0.014	0.957±0.009	**0.964±0.006**
YACHT	0.620±0.053	0.597±0.063	0.717±0.072	**0.727±0.070**

all methods is reported in Table 3. These results shows that NNG-MVG_A performs better than NNG-MVG_R in most datasets and is closer to HMC_A compared to PBP_A and NNG-FFG_A. However, we note that better predictive variance estimates do not reliably yield better active learning results, and in fact, active learning methods sometimes perform worse than random. Therefore, while information gain is a useful criterion for benchmarking purposes, it is important to explore other uncertainty-based active learning criteria.

5.4. Reinforcement Learning

We next experimented with using uncertainty to provide intrinsic motivation in reinforcement learning. Houthooft et al. (2016) proposed Variational Information Maximizing Exploration (VIME), which encouraged the agent to seek novelty through an information gain criterion. VIME involves training a separate BNN to predict the dynamics, i.e. learn to model the distribution $p(s_{t+1}|s_t, a_t; \theta)$. With the idea that surprising states lead to larger updates to the dynamics network, the reward function was augmented with an "intrinsic term" corresponding to the information gain for the BNN. If the history of the agent up until time step t is denoted as $\xi = \{s_1, a_1, ..., s_t\}$, then the modified reward can be written in the following form:

$$r^*(s_t, a_t, s_{t+1}) = r(s_t, a_t)$$
$$+ \eta D_{KL}(p(\theta|\xi_t, a_t, s_{t+1}) \| p(\theta|\xi_t)) \quad (14)$$

In the above formulation, the true posterior is generally intractable. Houthooft et al. (2016) approximated it using Bayes by Backprop (BBB) (Blundell et al., 2015). We experimented with replacing the fully factorized posterior with our NNG-MVG model.

Following the experimental setup of Houthooft et al. (2016), we tested our method in three continuous control tasks and sparsified the rewards (see supplement for details). We compared our NNG-MVG dynamics model with a Gaussian noise baseline, as well as the original VIME formulation using BBB. All experiments used TRPO to optimize the policy itself (Schulman et al., 2015).

Performance is measured by the average return (under the original MDP's rewards, not including the intrinsic term) at each iteration. Figure 3 shows the performance results in three tasks. Consistently with Houthooft et al. (2016), we observed that the Gaussian noise baseline completely breaks down and rarely achieves the goal, VIME significantly improved the performance. However, replacing the dynamics network with NNG-MVG considerably improved the exploration efficiency on all three tasks. Since the policy search algorithm was shared between all three conditions, we attribute this improvement to the improved uncertainty modeling by the dynamics network.

6. Conclusion

We drew a surprising connection between natural gradient ascent for point estimation and for variational inference. We exploited this connection to derive surprisingly simple variational BNN training procedures which can be instantiated as noisy versions of widely used optimization algorithms for point estimation. This let us efficiently fit MVG variational posteriors, which capture correlations between different weights. Our variational BNNs with MVG posteriors matched the predictive variances of HMC much better than fully factorized posteriors, and led to more efficient exploration in the settings of active learning and reinforcement learning with intrinsic motivation.

Acknowledgements

References

Amari, Shun-ichi. Neural learning in structured parameter spaces-natural riemannian gradient. In *Advances in neural information processing systems*, pp. 127–133, 1997.

Amari, Shun-Ichi. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

Asuncion, Arthur and Newman, David. Uci machine learning repository, 2007.

Ba, Jimmy, Grosse, Roger, and Martens, James. Distributed second-order optimization using kronecker-factored approximations. 2016.

Ba, Jimmy, Martens, James, and Grosse, Roger. Distributed second-order optimization using kronecker-factored approximations. In *International Conference on Learning Representations*, 2017.

Blundell, Charles, Cornebise, Julien, Kavukcuoglu, Koray, and Wierstra, Daan. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

Bonnet, Georges. Transformations des signaux aléatoires a travers les systemes non linéaires sans mémoire. *Annals of Telecommunications*, 19(9):203–220, 1964.

Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12: 2121–2159, 2011.

Graves, Alex. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pp. 2348–2356, 2011.

Grosse, Roger and Martens, James. A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, pp. 573–582, 2016.

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

Hernández-Lobato, José Miguel and Adams, Ryan. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, pp. 1861–1869, 2015.

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

Hoffman, Matthew D, Blei, David M, Wang, Chong, and Paisley, John. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

Houthooft, Rein, Chen, Xi, Duan, Yan, Schulman, John, De Turck, Filip, and Abbeel, Pieter. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pp. 1109–1117, 2016.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456, 2015.

Khan, Mohammad Emtiyaz, Lin, Wu, Tangkaratt, Voot, Liu, Zouzhu, and Nielsen, Didrik. Variational adaptive-Newton method for explorative learning. *arXiv preprint arXiv:1711.05560*, 2017a.

Khan, Mohammad Emtiyaz, Liu, Zuozhu, Tangkaratt, Voot, and Gal, Yarin. Vprop: Variational inference using rmsprop. *arXiv preprint arXiv:1712.01038*, 2017b.

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

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

Kingma, Diederik P, Salimans, Tim, and Welling, Max. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pp. 2575–2583, 2015.

Krizhevsky, Alex. Learning multiple layers of features from tiny images. 2009.

Louizos, Christos and Welling, Max. Structured and efficient variational deep learning with matrix gaussian posteriors. In *International Conference on Machine Learning*, pp. 1708–1716, 2016.

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

MacKay, David JC. Information-based objective functions for active data selection. *Neural computation*, 4(4):590–604, 1992a.

MacKay, David JC. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4:448–472, 1992b.

Martens, James. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.

Martens, James and Grosse, Roger. Optimizing neural networks with kronecker-factored approximate curvature. In *International Conference on Machine Learning*, pp. 2408–2417, 2015.

Neal, Radford M. *BAYESIAN LEARNING FOR NEURAL NETWORKS*. PhD thesis, University of Toronto, 1995.

Neal, Radford M and Hinton, Geoffrey E. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pp. 355–368. Springer, 1998.

Neal, Radford M et al. Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2(11), 2011.

Niculescu-Mizil, Alexandru and Caruana, Rich. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning*, pp. 625–632. ACM, 2005.

Opper, Manfred and Archambeau, Cédric. The variational gaussian approximation revisited. *Neural computation*, 21(3):786–792, 2009.

Peterson, Carsten. A mean field theory learning algorithm for neural networks. *Complex systems*, 1:995–1019, 1987.

Price, Robert. A useful theorem for nonlinear devices having gaussian inputs. *IRE Transactions on Information Theory*, 4(2):69–72, 1958.

Rezende, Danilo Jimenez and Mohamed, Shakir. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

Rezende, Danilo Jimenez, Mohamed, Shakir, and Wierstra, Daan. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Schulman, John, Levine, Sergey, Abbeel, Pieter, Jordan, Michael, and Moritz, Philipp. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 1889–1897, 2015.

Settles, Burr. Active learning literature survey. *University of Wisconsin, Madison*, 52(55-66):11, 2010.

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

Staines, Joe and Barber, David. Variational optimization. *arXiv preprint arXiv:1212.4507*, 2012.

Sun, Shengyang, Chen, Changyou, and Carin, Lawrence. Learning structured weight uncertainty in bayesian neural networks. In *Artificial Intelligence and Statistics*, pp. 1283–1292, 2017.

Williams, Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Wu, Yuhuai, Mansimov, Elman, Liao, Shun, Grosse, Roger, and Ba, Jimmy. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *arXiv preprint arXiv:1708.05144*, 2017.

A. Natural Gradient for Multivariate Gaussian

Suppose we have a model parameterized by $\boldsymbol{\theta}$ which lives in a subspace $\mathcal{S}$ (such as the set of symmetric matrices). The natural gradient $\tilde{\nabla}_{\boldsymbol{\theta}} h$ is motivated in terms of a trust region optimization problem, that finding the optimal θ in a neighborhood of θ_0 defined with KL divergence,

$$\arg\min_{\boldsymbol{\theta} \in \mathcal{S}} \alpha(\nabla_{\boldsymbol{\theta}} h)^\top \boldsymbol{\theta} + \mathrm{D_{KL}}(p_{\boldsymbol{\theta}} \| p_{\boldsymbol{\theta}_0})$$

$$\approx \arg\min_{\boldsymbol{\theta} \in \mathcal{S}} \alpha(\nabla_{\boldsymbol{\theta}} h)^\top \boldsymbol{\theta} + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \mathbf{F}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

Then the optimal solution to this optimization problem is given by $\boldsymbol{\theta} - \alpha\mathbf{F}^{-1}\nabla_{\boldsymbol{\theta}} h$. Here $\mathbf{F} = \nabla_{\boldsymbol{\theta}}^2 \mathrm{D_{KL}}(p_{\boldsymbol{\theta}} \| p_{\boldsymbol{\theta}_0}))$ is the Fisher matrix and α is the learning rate. Note that $h(\boldsymbol{\theta})$ and $\mathrm{D_{KL}}(p_{\boldsymbol{\theta}} \| p_{\boldsymbol{\theta}_0})$ are defined only for $\boldsymbol{\theta}, \boldsymbol{\theta}_0 \in \mathcal{S}$, but these can be extended to the full space however we wish without changing the optimal solution.

Now let assume the model is parameterized by multivariate Gaussian $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. The KL-divergence between $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\mathcal{N}_0(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$ are:

$$\mathrm{D_{KL}}(\mathcal{N} \| \mathcal{N}_0) = \frac{1}{2}\left[\log\frac{|\boldsymbol{\Sigma}_0|}{|\boldsymbol{\Sigma}|} - d + \mathrm{tr}(\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\Sigma})\right] \tag{15}$$
$$+ (\boldsymbol{\mu} - \boldsymbol{\mu}_0)^\top \boldsymbol{\Sigma}_0^{-1}(\boldsymbol{\mu} - \boldsymbol{\mu}_0)$$

Hence, the Fisher matrix w.r.t $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are

$$\mathbf{F}_{\boldsymbol{\mu}} = \nabla_{\boldsymbol{\mu}}^2 \mathrm{D_{KL}} = \boldsymbol{\Sigma}_0^{-1} \approx \boldsymbol{\Sigma}^{-1}$$
$$\mathbf{F}_{\boldsymbol{\Sigma}} = \nabla_{\boldsymbol{\Sigma}}^2 \mathrm{D_{KL}} = \frac{1}{2}\boldsymbol{\Sigma}^{-1} \otimes \boldsymbol{\Sigma}^{-1} \tag{16}$$

Then, by the property of vec-operator $(\mathbf{B}^\top \otimes \mathbf{A})\mathrm{vec}(\mathbf{X}) = \mathrm{vec}(\mathbf{AXB})$, we get the natural gradient updates

$$\tilde{\nabla}_{\boldsymbol{\mu}} h = \boldsymbol{\Sigma}\nabla_{\boldsymbol{\mu}} h$$
$$\tilde{\nabla}_{\boldsymbol{\Sigma}} h = 2\boldsymbol{\Sigma}\nabla_{\boldsymbol{\Sigma}} h\boldsymbol{\Sigma} \tag{17}$$

An analogous derivation gives us $\tilde{\nabla}_{\boldsymbol{\Lambda}} h = 2\boldsymbol{\Lambda}\nabla_{\boldsymbol{\Lambda}} h\boldsymbol{\Lambda}$. Considering $\boldsymbol{\Sigma} = \boldsymbol{\Lambda}^{-1}$, we have $\mathrm{d}\boldsymbol{\Sigma} = -\boldsymbol{\Sigma}\mathrm{d}\boldsymbol{\Lambda}\boldsymbol{\Sigma}$, which gives us the convenient formulas

$$\tilde{\nabla}_{\boldsymbol{\Sigma}} h = -2\nabla_{\boldsymbol{\Lambda}} h$$
$$\tilde{\nabla}_{\boldsymbol{\Lambda}} h = -2\nabla_{\boldsymbol{\Sigma}} h \tag{18}$$

Recall in variational inference, the gradient of ELBO $\mathcal{L}$ towards $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are given as

$$\nabla_{\boldsymbol{\mu}}\mathcal{L} = \mathbb{E}\left[\nabla_{\mathbf{w}}\log p(\mathcal{D}\,|\,\mathbf{w}) + \lambda\nabla_{\mathbf{w}}\log p(\mathbf{w})\right]$$
$$\nabla_{\boldsymbol{\Sigma}}\mathcal{L} = \frac{1}{2}\mathbb{E}\left[\nabla_{\mathbf{w}}^2\log p(\mathcal{D}\,|\,\mathbf{w}) + \lambda\nabla_{\mathbf{w}}^2\log p(\mathbf{w})\right] + \frac{\lambda}{2}\boldsymbol{\Sigma}^{-1} \tag{19}$$

Based on eq. (19) and eq. (18), the natural gradient is given by:

$$\tilde{\nabla}_{\boldsymbol{\mu}}\mathcal{L} = \boldsymbol{\Lambda}^{-1}\mathbb{E}\left[\nabla_{\mathbf{w}}\log p(\mathcal{D}\,|\,\mathbf{w}) + \lambda\nabla_{\mathbf{w}}\log p(\mathbf{w})\right]$$
$$\tilde{\nabla}_{\boldsymbol{\Lambda}}\mathcal{L} = -\mathbb{E}\left[\nabla_{\mathbf{w}}^2\log p(\mathcal{D}\,|\,\mathbf{w}) + \lambda\nabla_{\mathbf{w}}^2\log p(\mathbf{w})\right] - \lambda\boldsymbol{\Lambda} \tag{20}$$

B. Matrix Variate Gaussian

Recently Matrix Variate Gaussian (MVG) distribution are also used in Bayesian neural networks (Louizos & Welling, 2016; Sun et al., 2017). A matrix variate Gaussian distributions models a Gaussian distribution for a matrix $\mathbf{W} \in \mathbb{R}^{n\times p}$,

$$p(\mathbf{W}|\mathbf{M}, \mathbf{U}, \mathbf{V})$$
$$= \frac{\exp(\frac{1}{2}\,\mathrm{tr}[\mathbf{V}^{-1}(\mathbf{W} - \mathbf{M})^\top\mathbf{U}^{-1}(\mathbf{W} - \mathbf{M})])}{(2\pi)^{np/2}|\mathbf{V}|^{n/2}|\mathbf{U}|^{p/2}} \tag{21}$$

In which $\mathbf{M} \in \mathbb{R}^{n\times p}$ is the mean, $\mathbf{U} \in \mathbb{R}^{n\times n}$ is the covariance matrix among rows and $\mathbf{V} \in \mathbb{R}^{p\times p}$ is the covariance matrix among columns. Both $\mathbf{U}$ and $\mathbf{V}$ are positive definite matrices to be a covariance matrix. Connected with Gaussian distribution, vectorization of $\mathbf{W}$ confines a multivariate Gaussian distribution whose covariance matrix is Kronecker product of $\mathbf{V}$ and $\mathbf{U}$.

$$\mathrm{vec}(\mathbf{W}) \sim \mathcal{N}(\mathrm{vec}(\mathbf{M}), \mathbf{V} \otimes \mathbf{U}) \tag{22}$$

C. Implementation Details

C.1. Regression Implementation Details

The datasets were randomly splitted into training and test sets, with 90% of the data for training and the remaining for testing. To reduce the randomness, we repeated the splitting process for 20 times (except two largest datasets, i.e., Year and Protein, where we repeated 5 times and 1 times, respectively.) For all datasets except two largest ones, we used neural networks with 50 hidden units. For two largest datasets, we used 100 hidden units. Besides, we also introduced a Gamma prior, $p(\tau) = Gam(a_0 = 6, b_0 = 6)$ for the precision of the Gaussian likelihood and included the posterior $q(\tau) = Gam(\alpha^\tau, \beta^\tau)$ into variational objective. The variational posterior we used is $q(\mathbf{w}, \lambda) = q(\mathbf{w})q(\tau), q(\tau) = \mathrm{Gamma}(\alpha^\tau, \beta^\tau)$, then the expected likelihood $\mathcal{L}_r$ can be computed as

$$\mathcal{L}_r = \mathbb{E}_{q(\mathbf{w})}\mathbb{E}_{q(\tau)}\log p(y|\mathbf{x}, \mathbf{w}, \tau)$$
$$= \mathbb{E}_{q(\mathbf{w})}\mathbb{E}_{q(\tau)}\log \mathcal{N}(y|\hat{y}(\mathbf{x}, \mathbf{w}), \frac{1}{\tau})$$
$$= \frac{1}{2}\mathbb{E}_{q(\mathbf{w})}\mathbb{E}_{q(\tau)}[\log\tau - \tau(y - \hat{y}(\mathbf{x}, \mathbf{w}))^2 - \log 2\pi]$$
$$= \mathbb{E}_{q(\mathbf{w})}[\psi(\alpha^\tau) - \log\beta^\tau - \frac{\alpha^\tau}{\beta^\tau}(y - \hat{y}(\mathbf{x}, \mathbf{w}))^2 - \log 2\pi] \tag{23}$$

Where ψ represents digamma function. Therefore, ELBO can be computed with

$$\mathcal{L} = \mathcal{L}_r - \mathrm{D_{KL}}(q(\mathbf{w})\|p(\mathbf{w})) - \mathrm{D_{KL}}(q(\tau)\|p(\tau)) \tag{24}$$

With ELBO as above, we can directly compute the gradients towards variational parameters α, β using automatic differentiation.

In training, the input features and training targets were normalized to be zero mean and unit variance. We removed the normalization on the targets in test time. For each dataset, we set $\tilde{\alpha} = 0.01$ and $\tilde{\beta} = 0.001$ unless state otherwise. We set batch size 10 for 5 small datasets with less than 2000 data points, 500 for Year and 100 for other fours. Besides, we decay the learning rate by 0.1 in second half epochs.

C.2. Classification Implementation Details

Throughout classification experiments, we used VGG16 architecture but reduced the number of filters in each convolutional layer by half.

In training, we adopted learning rate selection strategy adopted by Ba et al. (2016). In particular, given a parameter update vector $\mathbf{v}$, the KL divergence between the predictive distributions before and after the update is given by the Fisher norm:

$$D_{KL}(q \parallel p) \approx \frac{1}{2}\mathbf{v}^\top \mathbf{F} \mathbf{v} \qquad (25)$$

Observe that choosing a step size of $\tilde{\alpha}$ will produce an update with squared Fisher norm $\tilde{\alpha}^2 \mathbf{v}^\top \mathbf{F} \mathbf{v}$. Motivated by the idea of trust region, we chose α in each iteration such that the squared Fisher norm is at most some value c:

$$\tilde{\alpha} = \min\left(\tilde{\alpha}_{\max}, \sqrt{\frac{c}{\mathbf{v}^\top \mathbf{F} \mathbf{v}}}\right) \qquad (26)$$

We used an exponential decay schedule $c_k = c_0 \zeta^k$, where c_0 and ζ were tunable parameters (c_0 is 0.001 or 0.01 for noisy K-FAC in our CIFAR-10 experiments when models trained without/with Batch Normalization (Ioffe & Szegedy, 2015), ζ is 0.95; c_0 is 0.0001 for noisy Adam), and k was incremented periodically (every epoch in our CIFAR-10 experiments). In practice, computing $\mathbf{v}^\top \mathbf{F} \mathbf{v}$ involves curvature-vector products after each update which introduces significant computational overhead, so we instead used the approximate Fisher $\tilde{\mathbf{F}}$ that we used to compute natural gradient. The maximum step size $\tilde{\alpha}_{\max}$ was set to be 0.01.

To reduce computational overhead of K-FAC (also noisy K-FAC) introduced by updating approximate Fisher matrix $\tilde{\mathbf{F}}$ and inverting it, we set $T_{\text{stats}} = 10$ and $T_{\text{inv}} = 200$. That means our curvature statistics are somewhat more stale, but we found that it didn't significantly affect per-iteration optimization performance. $\tilde{\beta}$ was set to 0.01 and 0.003 for noisy K-FAC and noisy Adam, respectively.

We noticed that it was favorable to tune regularization parameter λ and prior variance η. We used a small regularization parameter λ when data augmentation was adopted. E.g., we set $\lambda = 0.1$ when models were trained with data augmentation while $\lambda = 0.5$ otherwise. We speculate that using data augmentation leads to more training examples

(larger N), so it's reasonable to use a smaller λ. Moreover, we set η to 0.1 when models were trained without Batch Normalization.

C.3. Active Learning Implementation Details

Following the experimental protocol in PBP (Hernández-Lobato & Adams, 2015), we splited each dataset into training and test sets with 20 and 100 data points. All remaining data were included in pool sets. In all experiments, we used a neural network with one hidden layer and 10 hidden units.

After fitting our model in training data, we evaluated the performance in test data and further added one data point from pool set into training set. The selection was based on the method described by which was equivalent to choose the one with highest predictive variance. This process repeated 10 times, that is, we collected 9 points from pool sets. For each iteration, we re-trained the whole model from scratch.

Beyond that, as uncertainty estimation is of fundamental importance in active learning, we also performed experiments to evaluate the uncertainty estimation of our method directly, which was measured according to the Pearson's correlation of predictive variance compared to HMC (Neal et al., 2011). Recall Pearson's correlation,

$$\rho(X, Y) = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y} \qquad (27)$$

is a measure of linear correlation between two variables X and Y. Pearson's corrleation ranges from 0 to 1, with bigger value representing stronger correlations. We compared several algorithms, including PBP, NNG-FFG, NNG-MVG, NNG-BlkTri.

We trained NNG-FFG, NNG-MVG and NNG-BlkTri for 20000 epochs, PBP for 40 epochs and HMC with 20 chains, 100000 iterations. For all the models, we used 1000 sampled weights for predicting on the testing set, thus we could compute the model's predicative variance for every data point in the test set. Finally, we computed the Pearson's correlation between different models and HMC in terms of predicative variance. In all experiments, we used $\tilde{\alpha} = 0.01, \tilde{\beta} = 0.01$ and no extra damping term.

C.4. Reinforcement Learning Implementation Details

In all three tasks, CartPoleSwingup, MountainCar and DoublePendulum, we used one-layer Bayesian Neural Network with 32 hidden units for both BBB and NNG-MVG. And we used rectified linear unit (RELU) as our activation function. The number of samples drawn from variational posterior was fixed to 10 in the training process. For TRPO, the batch size was set to be 5000 and the replay pool has a fixed number of 100,000 samples. In both BBB and NNG-MVG, the dynamic model was updated in each epoch with 500 itera-

tions and 10 batch size. For the policy network, one-layer network with 32 tanh units was used.

For all three tasks, we sparsified the rewards in the following way. A reward of $+1$ is given in CartPoleSwingup when $cos(\theta) > 0.8$, with θ the pole angle; when the car escapes the valley in MountainCar; and when $D < 0.1$, with D the distance from the target in DoublePendulum.

To derive the intrinsic reward in Houthooft et al. (2016), we just need to analyze a single layer since we assume layer-wise independence in NNG-MVG. The intrinsic reward for each layer is given by (Note: $\mathcal{L}$ below is the ELBO with $q(\phi)$ as prior.)

$$
\mathrm{D_{KL}}\left(q(\phi') \,\|\, q(\phi)\right) =
$$
$$
\frac{1}{2}\tilde{\alpha}^2 \begin{bmatrix} \mathrm{vec}\{\nabla_{\boldsymbol{\mu}}\mathcal{L}\} \\ \mathrm{vec}\{\nabla_{\boldsymbol{\Sigma}}\mathcal{L}\} \end{bmatrix}^T \begin{bmatrix} \mathbf{F}_{\boldsymbol{\mu}}^{-1} & \\ & \mathbf{F}_{\boldsymbol{\Sigma}}^{-1} \end{bmatrix} \begin{bmatrix} \mathrm{vec}\{\nabla_{\boldsymbol{\mu}}\mathcal{L}\} \\ \mathrm{vec}\{\nabla_{\boldsymbol{\Sigma}}\mathcal{L}\} \end{bmatrix} \quad (28)
$$

Where $\tilde{\alpha}$ is the step-size. As shown in eq. (16), the Fisher matrix for $\boldsymbol{\mu}$ is given by $\mathbf{F}_{\boldsymbol{\mu}} = \boldsymbol{\Sigma}^{-1}$, thus the first term in eq. (28) is easy to get by exploiting Kronecker structure. However, $\mathbf{F}_{\boldsymbol{\Sigma}} = \frac{1}{2}\boldsymbol{\Sigma}^{-1} \otimes \boldsymbol{\Sigma}^{-1}$ where $\boldsymbol{\Sigma}$ itself is a gigantic matrix which makes computation of the second term intractable. Fortunately, the approximate variational posterior is a matrix variate Gaussian whose covariance is a Kronecker product, i.e. $\mathcal{MN}(\mathbf{W}; \mathbf{M}, \boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2) = \mathcal{N}(\mathrm{vec}(\mathbf{W}); \mathrm{vec}(\mathbf{M}), \boldsymbol{\Sigma}_2 \otimes \boldsymbol{\Sigma}_1)$, where $\mathbf{W}$ is of size $m \times n$ and $\boldsymbol{\mu} = \mathrm{vec}(\mathbf{M})$.

Using $\nabla_{\boldsymbol{\Sigma}}\mathcal{L} = \nabla_{\boldsymbol{\Sigma}_2}\mathcal{L} \otimes \boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2 \otimes \nabla_{\boldsymbol{\Sigma}_1}\mathcal{L}$ and substitute $\mathbf{F}_{\boldsymbol{\Sigma}}$ with $\frac{1}{2}\boldsymbol{\Sigma}^{-1} \otimes \boldsymbol{\Sigma}^{-1}$, we get the following identity

$$
\mathrm{vec}\{\nabla_{\boldsymbol{\Sigma}}\mathcal{L}\}^T \mathbf{F}_{\boldsymbol{\Sigma}} \, \mathrm{vec}\{\nabla_{\boldsymbol{\Sigma}}\mathcal{L}\} =
$$
$$
\begin{bmatrix} \mathrm{vec}\{\nabla_{\boldsymbol{\Sigma}_1}\mathcal{L}\} \\ \mathrm{vec}\{\nabla_{\boldsymbol{\Sigma}_2}\mathcal{L}\} \end{bmatrix}^T \begin{bmatrix} \mathbf{F}_{\boldsymbol{\Sigma}_1} & \mathbf{F} \\ \mathbf{F}^T & \mathbf{F}_{\boldsymbol{\Sigma}_2} \end{bmatrix} \begin{bmatrix} \mathrm{vec}\{\nabla_{\boldsymbol{\Sigma}_1}\mathcal{L}\} \\ \mathrm{vec}\{\nabla_{\boldsymbol{\Sigma}_2}\mathcal{L}\} \end{bmatrix} \quad (29)
$$

where Fisher matrices $\mathbf{F}_{\boldsymbol{\Sigma}_1}$ and $\mathbf{F}_{\boldsymbol{\Sigma}_2}$

$$
\mathbf{F}_{\boldsymbol{\Sigma}_1} = \frac{n}{2}\left(\boldsymbol{\Sigma}_1^{-1} \otimes \boldsymbol{\Sigma}_1^{-1}\right)
$$
$$
\mathbf{F}_{\boldsymbol{\Sigma}_2} = \frac{m}{2}\left(\boldsymbol{\Sigma}_2^{-1} \otimes \boldsymbol{\Sigma}_2^{-1}\right) \quad (30)
$$

By further ignoring off-diagonal block $\mathbf{F}$ in eq. (29), we can decompose $\mathrm{vec}\{\nabla_{\boldsymbol{\Sigma}}\mathcal{L}\}^T \mathbf{F}_{\boldsymbol{\Sigma}}^{-1} \mathrm{vec}\{\nabla_{\boldsymbol{\Sigma}}\mathcal{L}\}$ into two terms,

$$
\mathrm{vec}\{\nabla_{\boldsymbol{\Sigma}_1}\mathcal{L}\}^T \mathbf{F}_{\boldsymbol{\Sigma}_1}^{-1} \mathrm{vec}\{\nabla_{\boldsymbol{\Sigma}_1}\mathcal{L}\}
$$
$$
= \frac{2}{n} \mathrm{vec}\{\nabla_{\boldsymbol{\Sigma}_1}\mathcal{L}\}^T \mathrm{vec}\left(\boldsymbol{\Sigma}_1^{-1}\nabla_{\boldsymbol{\Sigma}_1}\mathcal{L}\boldsymbol{\Sigma}_1^{-1}\right) \quad (31)
$$

and

$$
\mathrm{vec}\{\nabla_{\boldsymbol{\Sigma}_2}\mathcal{L}\}^T \mathbf{F}_{\boldsymbol{\Sigma}_2}^{-1} \mathrm{vec}\{\nabla_{\boldsymbol{\Sigma}_2}\mathcal{L}\}
$$
$$
= \frac{2}{m} \mathrm{vec}\{\nabla_{\boldsymbol{\Sigma}_2}\mathcal{L}\}^T \mathrm{vec}\left(\boldsymbol{\Sigma}_2^{-1}\nabla_{\boldsymbol{\Sigma}_2}\mathcal{L}\boldsymbol{\Sigma}_2^{-1}\right) \quad (32)
$$

Now, each term can be computed efficient since $\boldsymbol{\Sigma}_1$ and $\boldsymbol{\Sigma}_2$ are small matrices.
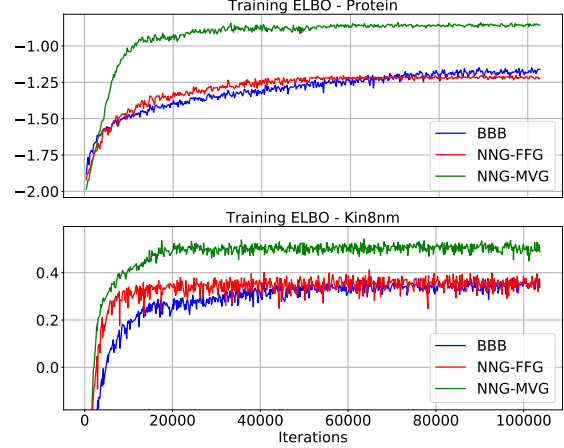
D. Additional Results

Figure 4: Training curves for all three methods. For each method, we tuned the learning rate for updating the posterior mean. Note that BBB and NNG-FFG use the same form of q, while NNG-MVG uses a more flexible q distribution.

We also run PBP_MV (Sun et al., 2017) and VMG (Louizos & Welling, 2016) on regression datasets from UCI collection (Asuncion & Newman, 2007). Results are shown in Table 5. Note that VMG introduced pseudo input-output pairs to enhance the flexibility of posterior distribution.

Table 5: Averaged test RMSE and log-likelihood for the regression benchmarks.

	TEST RMSE		TEST LOG-LIKELIHOOD	
DATASET	PBP_MV	VMG	PBP_MV	VMG
BOSTON	3.137±0.155	2.810±0.110	-2.666±0.081	-2.540±0.080
CONCRETE	5.397±0.130	4.700±0.140	-3.059±0.029	-2.980±0.030
ENERGY	0.556±0.016	1.160±0.030	-1.151±0.016	-1.450±0.030
KIN8NM	0.088±0.001	0.080±0.001	1.053±0.012	1.140±0.010
NAVAL	0.002±0.000	0.000±0.000	4.935±0.051	5.840±0.000
POW. PLANT	4.030±0.036	3.880±0.030	-2.830±0.008	-2.780±0.010
PROTEIN	4.490±0.012	4.140±0.010	-2.917±0.003	-2.840±0.000
WINE	0.641±0.006	0.610±0.010	-0.969±0.013	-0.930±0.020
YACHT	0.676±0.054	0.770±0.060	-1.024±0.025	-1.290±0.020
YEAR	9.450±NA	8.780±NA	-3.392±NA	-3.589±NA

While optimization was not the primary focus of this work, we compared NNG with the baseline BBB (Blundell et al., 2015) in terms of convergence. Training curves for two regression datasets are shown in Figure 4. We found that NNG-FFG trained in fewer iterations than BBB, while leveling off to similar ELBO values, even though our BBB implementation used Adam, and hence itself exploited diagonal curvature. Furthermore, despite the increased flexibility and larger number of parameters, NNG-MVG took roughly 2 times fewer iterations to converge, while at the same time surpassing BBB by a significant margin in terms of the ELBO.