Noisy Natural Gradient as Variational Inference

Guodong Zhang[*] **Shengyang Sun**[*] **David Duvenaud** **Roger Grosse**
University of Toronto
Vector Institute
{gdzhang, ssy, duvenaud, rgrosse}@cs.toronto.edu

Abstract

Combining the flexibility of deep learning with Bayesian uncertainty estimation has long been a goal in our field, and many modern approaches are based on variational Bayes. Unfortunately, one is forced to choose between overly simplistic variational families (e.g. fully factorized) or expensive and complicated inference procedures. We show that natural gradient ascent with adaptive weight noise can be interpreted as fitting a variational posterior to maximize the evidence lower bound (ELBO). This insight allows us to train full covariance, fully factorized, and matrix variate Gaussian variational posteriors using noisy versions of natural gradient, Adam, and K-FAC, respectively. On standard regression benchmarks, our noisy K-FAC algorithm makes better predictions and matches HMC's predictive variances better than existing methods. Its improved uncertainty estimates lead to more efficient exploration in the settings of 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 [21, 24, 11]. Stochastic variational inference is especially appealing because it closely resembles ordinary backprop [8, 5], 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 [18, 31] or multiplicative normalizing flows [19], but fitting such models can be expensive without further approximations.

In this work, we introduce and exploit a surprising connection between natural gradient descent [2] 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 [15, 23]. 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 [25] 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 [5], 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) [23] with correlated weight noise.

[*]Equal contribution.

2 Background

2.1 Variational Inference for Bayesian Neural Networks.

Assume we are given a dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$ and a neural network architecture with a set of parameters $\mathbf{w}$. A Bayesian neural network (BNN) is defined in terms of a prior $p(\mathbf{w})$ on the weights, as well as the likelihood $p(\mathcal{D}|\mathbf{w})$. Performing inference on the BNN requires integrating over the intractable posterior distribution $p(\mathbf{w}\,|\,\mathcal{D})$. Variational Bayesian methods [11, 8, 5] attempt to fit an approximate posterior $q(\mathbf{w})$ to maximize the evidence lower bound (ELBO):

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

where λ is a regularization parameter and $\boldsymbol{\phi}$ 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.)

The most commonly used variational BNN training method is Bayes By Backprop (BBB) [5], which uses a fully factorized Gaussian approximation to the posterior, i.e. $q(\mathbf{w}) = \mathcal{N}(\mathbf{w}; \boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$. The variational parameters $\boldsymbol{\phi} = (\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$ are adapted using stochastic gradients of $\mathcal{L}$ obtained using the reparameterization trick [17]. The resulting updates for $\boldsymbol{\mu}$ look like ordinary backpropagation updates for the weights of a non-Bayesian neural network, except that $\mathbf{w}$ is sampled from the variational posterior. Another interpretation of BBB regards $\boldsymbol{\mu}$ as a point estimate of the weights and $\boldsymbol{\sigma}$ as the standard deviations of Gaussian noise added independently for each training example. This similarity to ordinary neural net training is a big part of the appeal of BBB and other variational BNN approaches.

2.2 Natural Gradient

Natural gradient descent is a second-order optimization method originally proposed by Amari [1]. 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). While both methods are broadly applicable, we limit the present discussion to neural networks for simplicity.

In natural gradient for point estimation (NGPE), we assume the neural network computes a predictive distribution $r(y|\mathbf{x};\mathbf{w})$ 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 $\mathbf{F} = \mathrm{Cov}_{\mathbf{x}\sim p_{\mathrm{data}}, y\sim r(y|\mathbf{x};\mathbf{w})}(\nabla_{\mathbf{w}} \log r(y|\mathbf{x};\mathbf{w}))$, 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; the details are inessential to this paper, but see [22] for a comprehensive overview. Because the dimension of $\mathbf{F}$ is the number of parameters, which can be in the tens of millions for modern neural networks, exact computation of the natural gradient is typically infeasible, and approximations are required (see below).

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 (Eqn. 1). Analogously to the point estimation setting, the natural gradient is defined as $\tilde{\nabla}_{\boldsymbol{\phi}}\mathcal{L} = \mathbf{F}^{-1}\nabla_{\boldsymbol{\phi}}\mathcal{L}$; but in this case, $\mathbf{F}$ is the Fisher matrix of q, i.e. $\mathbf{F} = \mathrm{Cov}_{\mathbf{w}\sim q}(\nabla_{\boldsymbol{\phi}} \log q(\mathbf{w};\boldsymbol{\phi}))$. Note that in contrast with point estimation, $\mathbf{F}$ is a metric on $\boldsymbol{\phi}$, 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. As an important example, Hoffman et al. [12] used natural gradient to scale up variational Bayes for latent variable models such as LDA, and found that the natural gradient ascent updates closely resemble the EM-like updates used in variational Bayes.

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.

2.3 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)

[23] 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 $\mathbf{s}_l = \mathbf{W}_l^T \mathbf{a}_l$. Therefore, weight gradient is $\nabla_{\mathbf{W}_l} \mathcal{L} = \mathbf{a}_l (\nabla_{\mathbf{s}_l} \mathcal{L})^T$. With this gradient formula, K-FAC decouples this layer's fisher matrix $\mathbf{F}_l$ using mild approximations,

$$\begin{aligned} \mathbf{F}_l &= \mathbb{E}[\mathrm{vec}\{\nabla_{\mathbf{W}_l}\mathcal{L}\}\mathrm{vec}\{\nabla_{\mathbf{W}_l}\mathcal{L}\}^T] = \mathbb{E}[\{\nabla_{\mathbf{s}_l}\mathcal{L}\}\{\nabla_{\mathbf{s}_l}\mathcal{L}\}^T \otimes \mathbf{a}_l\mathbf{a}_l^T] \\ &\approx \mathbb{E}[\{\nabla_{\mathbf{s}_l}\mathcal{L}\}\{\nabla_{\mathbf{s}_l}\mathcal{L}\}^T] \otimes \mathbb{E}[\mathbf{a}_l\mathbf{a}_l^T] = \mathbf{S}_l \otimes \mathbf{A}_l \end{aligned} \tag{2}$$

Where $\mathbf{A}_l = \mathbb{E}[\mathbf{a}\mathbf{a}^T]$ and $\mathbf{S}_l = \mathbb{E}[\{\nabla_{\mathbf{s}}\mathcal{L}\}\{\nabla_{\mathbf{s}}\mathcal{L}\}^T]$. The approximation above assumes independence between $\mathbf{a}$ and $\mathbf{s}$, which proves to be accurate in practice. Further, assuming between-layer independence, the whole fisher matrix $\mathbf{F}$ can be approximated as block diagonal consisting of layerwise fisher matrices $\mathbf{F}_l$. Decoupling $\mathbf{F}_l$ into $\mathbf{A}_l$ and $\mathbf{S}_l$ not only avoids the memory issue saving $\mathbf{F}_l$, but also provides efficient natural gradient computation.

$$\mathbf{F}_l^{-1}\mathrm{vec}\{\nabla_{\mathbf{W}_l}\mathcal{L}\} = \mathbf{S}_l^{-1} \otimes \mathbf{A}_l^{-1}\mathrm{vec}\{\nabla_{\mathbf{W}_l}\mathcal{L}\} = \mathrm{vec}[\mathbf{A}_l^{-1}\nabla_{\mathbf{W}_l}\mathcal{L}\mathbf{S}_l^{-1}] \tag{3}$$

As shown by Eqn. 3, 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 2.2.) 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).*

In NGVI, our goal is to maximize the ELBO $\mathcal{L}$ (Eqn. 1) with respect to the parameters $\boldsymbol{\phi}$ of a variational posterior distribution $q(\mathbf{w})$. We assume q is a multivariate Gaussian parameterized by $\boldsymbol{\phi} = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Building on the analysis of [26], we determine the natural gradient of the ELBO with respect to $\boldsymbol{\mu}$ and the precision matrix $\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1}$ (see Appendix A for details):

$$\tilde{\nabla}_{\boldsymbol{\mu}}\mathcal{L} = \boldsymbol{\Lambda}^{-1}\mathbb{E}_{\mathbf{w}\sim q}\left[\nabla_{\mathbf{w}}\log p(\mathcal{D}\,|\,\mathbf{w}) + \lambda\nabla_{\mathbf{w}}\log p(\mathbf{w})\right] \tag{4}$$

$$\tilde{\nabla}_{\boldsymbol{\Lambda}}\mathcal{L} = -\mathbb{E}_{\mathbf{w}\sim q}\left[\nabla^2_{\mathbf{w}}\log p(\mathcal{D}\,|\,\mathbf{w}) + \lambda\nabla^2_{\mathbf{w}}\log p(\mathbf{w})\right] - \lambda\boldsymbol{\Lambda} \tag{5}$$

Here, λ is the weighting of the KL term in Eqn. 1. We make several observations. First, the term inside the expectation in Eqn. 4 is the gradient for MAP estimation of $\mathbf{w}$. Second, the update for $\boldsymbol{\mu}$ is preconditioned by $\boldsymbol{\Lambda}^{-1}$, which encourages faster movement in directions of higher posterior uncertainty. Finally, the fixed point equation for $\boldsymbol{\Lambda}$ is given by

$$\boldsymbol{\Lambda} = -\mathbb{E}\left[\frac{1}{\lambda}\nabla^2_{\mathbf{w}}\log p(\mathcal{D}\,|\,\mathbf{w}) + \nabla^2_{\mathbf{w}}\log p(\mathbf{w})\right].$$

Hence, if $\lambda = 1$ (as in proper variational inference), $\boldsymbol{\Lambda}$ will tend towards the expected Hessian of $-\log p(\mathbf{w}, \mathcal{D})$, so the update rule for $\boldsymbol{\mu}$ will somewhat resemble a Newton-Raphson update. Smaller values of λ result in increased weighting of the log-likelihood Hessian, and hence a more concentrated posterior. (We note that, in independent work, Khan et al. [14] recently derived a similar stochastic Newton update; see Section 4.)

Based on these formulas, we derive the following stochastic natural gradient ascent updates. In each iteration, we sample $(\mathbf{x}, y) \sim p_{\mathrm{data}}$ and $\mathbf{w} \sim q$.

$$\begin{aligned} \boldsymbol{\mu} &\leftarrow \boldsymbol{\mu} + \alpha\boldsymbol{\Lambda}^{-1}\left[\nabla_{\mathbf{w}}\log p(y\,|\,\mathbf{w},\mathbf{x}) + \frac{\lambda}{N}\nabla_{\mathbf{w}}\log p(\mathbf{w})\right] \\ \boldsymbol{\Lambda} &\leftarrow \left(1 - \frac{\lambda\beta}{N}\right)\boldsymbol{\Lambda} - \beta\left[\nabla^2_{\mathbf{w}}\log p(y\,|\,\mathbf{w},\mathbf{x}) + \frac{\lambda}{N}\nabla^2_{\mathbf{w}}\log p(\mathbf{w}),\right] \end{aligned} \tag{6}$$

where α and β are separate learning rates for $\boldsymbol{\mu}$ and $\boldsymbol{\Lambda}$. Roughly speaking, the update rule for $\boldsymbol{\Lambda}$ corresponds to an exponential moving average of the Hessian, and the update rule for $\boldsymbol{\mu}$ is a stochastic Newton step using $\boldsymbol{\Lambda}$.

This update rule has two problems. First, the log-likelihood Hessian may be hard to compute, and is undefined for neural net architectures which use non-differentiable activation functions such as ReLU. Second, if the negative log-likelihood is non-convex (as is the case for multilayer neural nets), the Hessian could have negative eigenvalues, so without further constraints, the update may result in $\mathbf{\Lambda}$ which is not positive semidefinite. We circumvent both of these problems by approximating the negative log-likelihood Hessian with the NGPE Fisher matrix $\mathbf{F} = \mathrm{Cov}_{\mathbf{x} \sim p_{\mathrm{data}}, y \sim r(y|\mathbf{x};\mathbf{w})}(\nabla_{\mathbf{w}} \log r(y|\mathbf{x};\mathbf{w}))$. This approximation guarantees that $\mathbf{\Lambda}$ 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 [8], so we refer to it henceforth as the *Graves approximation*.[2]

$$\mathbf{\Lambda} \leftarrow \left(1 - \frac{\lambda\beta}{N}\right)\mathbf{\Lambda} + \beta\left[\nabla_{\mathbf{w}} \log p(y \mid \mathbf{w}, \mathbf{x})\left(\nabla_{\mathbf{w}} \log p(y \mid \mathbf{w}, \mathbf{x})\right)^{\top} - \frac{\lambda}{N}\nabla^2_{\mathbf{w}} \log p(\mathbf{w})\right] \tag{7}$$

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 [22] for details.

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, by separating out the two terms in the moving average, we can rewrite the update Eqn. 7 in terms of exponential moving averages of the Fisher matrix and the prior Hessian:

$$\begin{aligned}
\mathbf{\Lambda} &= \frac{N}{\lambda}\bar{\mathbf{F}} + \bar{\mathbf{H}}_p \\
\bar{\mathbf{F}} &\leftarrow \left(1 - \frac{\lambda\beta}{N}\right)\bar{\mathbf{F}} + \frac{\lambda\beta}{N}\nabla_{\mathbf{w}} \log p(y \mid \mathbf{w}, \mathbf{x})\left(\nabla_{\mathbf{w}} \log p(y \mid \mathbf{w}, \mathbf{x})\right)^{\top} \\
\bar{\mathbf{H}}_p &\leftarrow \left(1 - \frac{\lambda\beta}{N}\right)\bar{\mathbf{H}}_p - \frac{\lambda\beta}{N}\nabla^2_{\mathbf{w}} \log p(\mathbf{w})
\end{aligned} \tag{8}$$

This update rule highlights an awkward interaction between the KL weight λ and the learning rates α and β. We can fix this by writing the update rules in terms of alternative learning rates $\tilde{\alpha} = \alpha\lambda/N$ and $\tilde{\beta} = \beta\lambda/N$.

$$\begin{aligned}
\boldsymbol{\mu} &\leftarrow \boldsymbol{\mu} + \tilde{\alpha}\left(\bar{\mathbf{F}} + \frac{\lambda}{N}\bar{\mathbf{H}}_p\right)^{-1}\left[\nabla_{\mathbf{w}} \log p(y \mid \mathbf{w}, \mathbf{x}) + \frac{\lambda}{N}\nabla_{\mathbf{w}} \log p(\mathbf{w})\right] \\
\bar{\mathbf{F}} &\leftarrow (1 - \tilde{\beta})\bar{\mathbf{F}} + \tilde{\beta}\nabla_{\mathbf{w}} \log p(y \mid \mathbf{w}, \mathbf{x})\left(\nabla_{\mathbf{w}} \log p(y \mid \mathbf{w}, \mathbf{x})\right)^{\top} \\
\bar{\mathbf{H}}_p &\leftarrow (1 - \tilde{\beta})\bar{\mathbf{H}}_p - \tilde{\beta}\nabla^2_{\mathbf{w}} \log p(\mathbf{w})
\end{aligned} \tag{9}$$

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 $\bar{\mathbf{H}}_p$ (see below), and that the weights are sampled from q, which is a Gaussian with covariance $\mathbf{\Sigma} = \mathbf{\Lambda}^{-1} = (\frac{N}{\lambda}\bar{\mathbf{F}} + \bar{\mathbf{H}}_p)^{-1}$. 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

In the special case of a spherical Gaussian prior, we have that $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}; \mathbf{0}, \eta\mathbf{I})$, and therefore $\nabla^2_{\mathbf{w}} \log p(\mathbf{w}) = -\eta^{-1}\mathbf{I}$. Therefore, $\bar{\mathbf{H}}_p = \bar{\eta}^{-1}\mathbf{I}$, where $\bar{\eta}^{-1}$ is an exponential moving average of

[2]Eqn. 7 leaves ambiguous what distribution the gradients are sampled from. One option is to use the same gradients used for optimization, as is done by Graves's method [8] and by Adam [15]. The Fisher matrix estimated using the empirical gradients is known as the *empirical Fisher*. Alternatively, one can sample the targets from the model's predictions, as done in K-FAC [23]. The resulting $\mathbf{F}$ is known as the *true Fisher*. The true Fisher is a better approximation to the Hessian [22], and this is what we use throughout our experiments.

the prior precision. (If η is fixed, then we have simply $\bar{\mathbf{H}}_p = \eta^{-1}\mathbf{I}$). 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 $\lambda/N\eta$. In practice, it may be advantageous to add additional damping for purposes of optimization. However, we found we did not need to do this in any of our experiments.[3]

Formula above demonstrates interesting connection between covariance and Fisher information, thus imposing different structures on Fisher corresponds to varying posterior distributions. As a full covariance multivariate Gaussian posterior is computationally impractical for all but the smallest networks, next we will show several simplied Fisher structures and connect them to different posterior families.

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. Perhaps the simplest approach is to approximate $\mathbf{F}$ with a diagonal matrix $\text{diag}(\mathbf{f})$, as done by Adagrad [7] and Adam [15]. For our NNG approach, this yields the following updates, where division and squaring are applied elementwise:

$$\begin{aligned} \boldsymbol{\mu} &\leftarrow \boldsymbol{\mu} + \tilde{\alpha}\left[\nabla_{\mathbf{w}} \log p(y \,|\, \mathbf{w}, \mathbf{x}) - \frac{\lambda}{N\eta}\mathbf{w}\right] \Big/ \left(\bar{\mathbf{f}} + \frac{\lambda}{N\eta}\right) \\ \bar{\mathbf{f}} &\leftarrow (1-\tilde{\beta})\bar{\mathbf{f}} + \tilde{\beta}[\nabla_{\mathbf{w}} \log p(y \,|\, \mathbf{w}, \mathbf{x})]^2 \end{aligned} \tag{10}$$

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 regard these differences as inessential. We define *noisy Adam* by modifying the above update rules to be consistent with Adam in these two respects; the full procedure is given in Alg. 1. We note that these modifications 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.

Algorithm 1 Noisy Adam. Superscript (k) denotes the kth iteration. Differences from standard Adam are shown in blue.

```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2$: Exponential decay rates for updating $\boldsymbol{\mu}$ and the Fisher $\mathbf{F}$
Require: $\lambda, \eta$ : KL weighting, prior variance
  $k \leftarrow 0$
  $\mathbf{m} \leftarrow \mathbf{0}$
  Calculate the damping term $\gamma = \frac{\lambda}{N\eta}$ (Note: in standard Adam, $\gamma$ is typically set to $10^{-8}$)
  while stopping criterion not met do
    $k \leftarrow k + 1$
    $\mathbf{w}^{(k)} \sim \mathcal{N}(\boldsymbol{\mu}^{(k-1)}, \frac{\lambda}{N} \text{diag}(\mathbf{f}^{(k-1)} + \gamma)^{-1})$
    $\mathbf{v}^{(k)} = \nabla_{\mathbf{w}} \log p(y \,|\, \mathbf{w}^{(k)}, \mathbf{x}) - \gamma \cdot \mathbf{w}^{(k)}$
    $\mathbf{m}^{(k)} \leftarrow \beta_1 \cdot \mathbf{m}^{(k-1)} + (1-\beta_1) \cdot \mathbf{v}^{(k)}$    (Update momentum)
    $\mathbf{f}^{(k)} \leftarrow \beta_2 \cdot \mathbf{f}^{(k-1)} + (1-\beta_2) \cdot (\nabla_{\mathbf{w}^{(k)}} \log p(y \,|\, \mathbf{w}^{(k)}, \mathbf{x}))^2$
    $\tilde{\mathbf{m}}^{(k)} \leftarrow \mathbf{m}^{(k)}/(1-\beta_1^k)$
    $\hat{\mathbf{m}}^{(k)} \leftarrow \tilde{\mathbf{m}}^{(k)}/(\sqrt{\mathbf{f}^{(k)}} + \gamma)$
    $\boldsymbol{\mu}^{(k)} \leftarrow \boldsymbol{\mu}^{(k-1)} + \alpha \cdot \hat{\mathbf{m}}^{(k)}$    (Update parameters)
  end while
```

[3]We speculate that because the precision $\boldsymbol{\Lambda}$ is fit using variational inference rather than a Taylor approximation, it is encouraged to reflect the global shape of the local mode of the distribution, helping to stabilize the update.

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 in order to compactly capture posterior correlations between different weights [18, 31]. 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. $\mathcal{MN}(\mathbf{W}_l; \mathbf{M}, \mathbf{\Sigma}_1, \mathbf{\Sigma}_2) = \mathcal{N}(\text{vec}(\mathbf{W}); \text{vec}(\mathbf{M}), \mathbf{\Sigma}_2 \otimes \mathbf{\Sigma}_1)$. (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.) One can sample from an MVG distribution by sampling a matrix $\mathbf{E}$ of i.i.d. standard Gaussians and then taking $\mathbf{W} = \mathbf{M} + \mathbf{\Sigma}_1^{1/2}\mathbf{E}\mathbf{\Sigma}_2^{1/2}$, where $\boldsymbol{\mu} = \text{vec}(\mathbf{M})$. 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 $m^2 n^2/2$. Therefore, MVGs are potentially powerful due to their compact representation of posterior covariances between weights. However, training MVG posteriors is very difficult, since computing the gradients and enforcing the positive semidefinite constraint for $\mathbf{\Sigma}_1$ and $\mathbf{\Sigma}_2$ typically requires expensive matrix operations such as inversion. Therefore, existing methods for fitting MVG posteriors typically impose additional structure such as diagonal covariance [18] or products of Householder transformations [31] to ensure efficient updates.

We observe that K-FAC [23] uses a Kronecker-factored approximation to the Fisher matrix for each layer's weights, as in Eqn. 2. By plugging this approximation in to Eqn. 8, 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 estimated $\mathbf{A}_l$ 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 Eqn. 9:

$$\begin{aligned} \bar{\mathbf{A}}_l &\leftarrow (1-\tilde{\beta})\bar{\mathbf{A}}_l + \tilde{\beta}\mathbf{a}_l\mathbf{a}_l^T \\ \bar{\mathbf{S}}_l &\leftarrow (1-\tilde{\beta})\bar{\mathbf{S}}_l + \tilde{\beta}\{\nabla_{\mathbf{s}_l}\log p(y\,|\,\mathbf{w},\mathbf{x})\}\{\nabla_{\mathbf{s}_l}\log p(y\,|\,\mathbf{w},\mathbf{x})\}^T \end{aligned} \tag{11}$$

We note that our scheme is not equivalent to performing natural gradient ascent directly on $\mathbf{A}_l$ and $\mathbf{S}_l$, so it is best regarded as a tractable approximation to Eqn. 9. Conveniently, because these factors are estimated from the empirical covariances, they (and hence also $\mathbf{\Lambda}$) are automatically positive semidefinite.

Plugging the above formulas into Eqn. 8 does not quite yield an MVG posterior due to the addition of $\bar{\mathbf{H}}_p$. For general $\bar{\mathbf{H}}_p$, there may be no compact representation of $\mathbf{\Lambda}$. However, for spherical Gaussian priors[4], we can approximate Σ using a trick proposed by [23] in the context of damping. In particular, we add $\pi_l\sqrt{\frac{\lambda}{N\eta}}\mathbf{I}$ and $\frac{1}{\pi_l}\sqrt{\frac{\lambda}{N\eta}}\mathbf{I}$ for a scalar constant π_l to the individual Kronecker factors $\mathbf{A}_l$ and $\mathbf{S}_l$. In this way, the covariance $\mathbf{\Sigma}_l = \text{Cov}(\text{vec}(\mathbf{W}_l))$ decomposes as the Kronecker product of two terms:

$$\begin{aligned} \mathbf{\Sigma}_l &= \frac{\lambda}{N}[\mathbf{S}_l^{\gamma}]^{-1} \otimes [\mathbf{A}_l^{\gamma}]^{-1} \\ &\triangleq \frac{\lambda}{N}\left(\bar{\mathbf{S}}_l + \frac{1}{\pi_l}\sqrt{\frac{\lambda}{N\eta}}\mathbf{I}\right)^{-1} \otimes \left(\bar{\mathbf{A}}_l + \pi_l\sqrt{\frac{\lambda}{N\eta}}\mathbf{I}\right)^{-1} \end{aligned} \tag{12}$$

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.

K-FAC is a very efficient optimizer, and has been observed to yield significant speedups over standard methods for training convolutional networks [9] and improved sample efficiency in reinforcement learning [32]. The algorithm is GPU-friendly, and efficient implementations typically only introduce small (e.g. 1.5–2x) overhead compared with ordinary SGD. However, our focus here is not optimization speed, but rather the ability of noisy K-FAC to fit flexible MVG posteriors, in order to obtain improved uncertainty estimates compared with fully factorized Gaussians.

[4]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 2 Noisy K-FAC. Subscript l denotes layers, $\mathbf{w}_l = \mathrm{vec}(\mathbf{W}_l)$, and $\boldsymbol{\mu}_l = \mathrm{vec}(\mathbf{M}_l)$. We assume zero momentum for simplicity. Differences from standard K-FAC are shown in blue.

```
Require: $\alpha$: stepsize
Require: $\beta$: exponential moving average parameter for covariance factors
Require: $\lambda, \eta$ : KL weighting, prior variance
Require: stats and inverse update intervals $T_{\mathrm{stats}}$ and $T_{\mathrm{inv}}$
  $k \leftarrow 0$
  Initialize $\{\boldsymbol{\mu}_l\}_{l=1}^{L}, \{\mathbf{S}_l\}_{l=1}^{L}, \{\mathbf{A}_l\}_{l=1}^{L}$
  Calculate the damping term $\gamma = \frac{\lambda}{N\eta}$ (Note: in standard K-FAC, $\gamma$ is chosen manually)
  while stopping criterion not met do
    $k \leftarrow k + 1$
    $\mathbf{W}_l \sim \mathcal{MN}(\mathbf{M}_l, \ \frac{\lambda}{N}[\mathbf{A}_l^{\gamma}]^{-1}, \ [\mathbf{S}_l^{\gamma}]^{-1})$
    if $k \equiv 0 \pmod{T_{\mathrm{stats}}}$ then
      Sample the targets from the model's predictive distribution.
      Update the factors $\{\mathbf{S}_l\}_{l=1}^{L}, \{\mathbf{A}_l\}_{l=0}^{L-1}$ using Eqn. 11 with decay rate $\beta$
    end if
    if $k \equiv 0 \pmod{T_{\mathrm{inv}}}$ then
      Calculate the inverses $\{[\mathbf{S}_l^{\gamma}]^{-1}\}_{l=1}^{L}, \{[\mathbf{A}_l^{\gamma}]^{-1}\}_{l=0}^{L-1}$ using Eqn. 12.
    end if
    $\mathbf{V}_l = \nabla_{\mathbf{W}_l} \log p(y \,|\, \mathbf{w}, \mathbf{x}) - \gamma \cdot \mathbf{W}_l$
    $\mathbf{M}_l \leftarrow \mathbf{M}_l + \alpha[\mathbf{A}_l^{\gamma}]^{-1}\mathbf{V}_l[\mathbf{S}_l^{\gamma}]^{-1}$
  end while
```

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 [23]. 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 [23] for the details.

4 Related Work

Variational inference was first applied to neural networks by [27] and [11]. More recently, [8] proposed a practical method for variational inference with fully factorized Gaussian posteriors which used a simple (but biased) gradient estimator. Improving on that work, [5] proposed a unbiased gradient estimator using the reparameterization trick of [17]. In place of the ELBO, [10] proposed to use expectation propagation with fully factorized posterior distributions and found a closed-form approximation which avoided the sampling for weights. [16] 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. [18] introduced the matrix variate Gaussian posterior as well as a Gaussian process approximation. [31] 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 [28], [19] applied normalizing flows to the auxiliary latent variables to greatly enhance the posterior approximation. However, the introduction of auxiliary variables results in a looser variational lower bound.

Since natural gradient was proposed by Amari [2], there has been much work on tractable approximations. [12] observed that for exponential family posteriors, the exact natural gradient could be tractably computed using stochastic versions of variational Bayes E-M updates. [23] proposed K-FAC for performing efficient natural gradient optimization in deep neural networks. Following on that work, K-FAC has been adopted in many tasks to gain optimization benefits, including convolutional networks [9] and Reinforcement Learning [32], and was shown to be amenable to distributed computation [4].

		Test RMSE			Test log-likelihood			
Dataset	BBB	PBP	PBP_MV	NNG-MVG	BBB	PBP	PBP_MV	NNG-MVG
Boston	2.517±0.022	3.014±0.180	3.137±0.155	**2.296**±**0.029**	-2.500±0.004	-2.574±0.089	-2.666±0.081	**-2.336**±**0.005**
Concrete	5.770±0.066	5.667±0.093	5.397±0.130	**5.173**±**0.070**	-3.169±0.011	-3.161±0.019	**-3.059**±**0.029**	-3.073±0.014
Energy	0.499±0.019	1.804±0.048	0.556±0.016	**0.438**±**0.003**	-1.552±0.006	-2.042±0.019	**-1.151**±**0.016**	-1.411±0.002
Kin8nm	0.079±0.001	0.098±0.001	0.088±0.001	**0.076**±**0.000**	1.118±0.004	0.896±0.006	1.053±0.012	**1.151**±**0.006**
Naval	**0.000**±**0.000**	0.006±0.000	0.002±0.000	**0.000**±**0.000**	6.431±0.082	3.731±0.006	4.935±0.051	**7.182**±**0.057**
Pow. Plant	4.224±0.007	4.124±0.035	**4.030**±**0.036**	4.085±0.006	-2.851±0.001	-2.837±0.009	-2.830±0.008	**-2.818**±**0.002**
Protein	4.390±0.009	4.732±0.013	4.490±0.012	**4.058**±**0.006**	-2.900±0.002	-2.973±0.003	-2.917±0.003	**-2.820**±**0.002**
Wine	0.639±0.002	0.635±0.008	0.641±0.006	**0.634**±**0.001**	-0.971±0.003	-0.968±0.014	-0.969±0.013	**-0.961**±**0.001**
Yacht	0.983±0.055	1.015±0.054	**0.676**±**0.054**	0.827±0.017	-2.380±0.004	-1.634±0.016	**-1.024**±**0.025**	-2.274±0.003
Year	9.076±NA	**8.879**±**NA**	9.450±NA	8.885±NA	-3.614±NA	-3.603±NA	**-3.392**±**NA**	-3.595±NA

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

In independent work, Khan et al. [14] derived a stochastic Newton update similar to Eqs. 4 and 5. As their focus was on optimization rather than variational inference, they did not include the KL term, and as a result, their formula for $\mathbf{\Lambda}$ involved a running sum of the individual Hessians, rather than an exponential moving average.

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 and convergence speed? (2) Can NNG achieve better uncertainty estimates? (3) Does it enable more efficient exploration in active learning and reinforcement learning?

To evaluate the effectiveness of our method, we compared it with Bayes By Backprop (BBB) [5], probabilistic backpropagation (PBP) with factorial gaussian posterior [10] and PBP with matrix variate Gaussian posterior (PBP_MV) [31]. Our method with full covariance multivariate Gaussian, fully factorial Gaussian, matrix variate Gaussian and block tridiagonal posterior are denoted as NNG-full, NNG-FFG, NNG-MVG and NNG-BlkTri, respectively.

5.1 Regression Benchmarks

We first experimented with regression datasets from the UCI collection [3] which were introduced as a standard BNN benchmark by [10]. All experiments used networks with one hidden layer unless stated otherwise (experimental details see Appendix C). Following previous works [10] [18], 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-MVG method achieved substantially better RMSE and log-likelihoods than BBB due to the more flexible posterior. NNG-MVG also outperformed PBP_MV, which parameterizes matrix variate Gaussians with orthogonal matrices. A further comparison of the prediction performance on two-layer neural networks can be found in Appendix D.

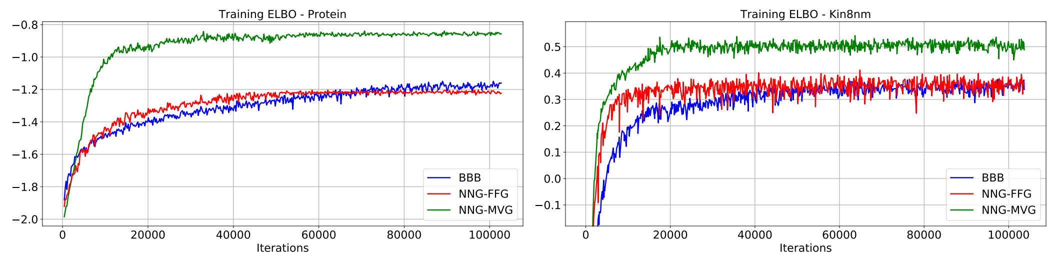


Figure 1: Training curves for all three methods. For each method, we tuned the learning rate for updating the posterior mean. Note that BBB and NNG-FFG use the same form of q, while NNG-MVG uses a more flexible q distribution.

While optimization was not the primary focus of this work, we compared NNG with the baseline BBB in terms of convergence. Training curves for two regression datasets are shown in Fig. 1. We

Dataset	PBP_R	PBP_A	NNG-MVG_R	NNG-MVG_A	HMC_R	HMC_A
Boston	6.716±0.500	5.480±0.175	6.364±0.287	**5.139±0.033**	5.750±0.222	5.156±0.150
Concrete	12.417±0.392	11.894±0.254	11.869±0.187	11.592±0.117	**10.564±0.198**	11.484±0.191
Energy	3.743±0.121	3.399±0.064	3.419±0.054	**3.035±0.038**	3.264±0.067	3.118±0.062
Kin8nm	0.259±0.006	0.254±0.005	0.247±0.002	0.236±0.002	0.226±0.004	**0.223±0.003**
Naval	0.015±0.000	0.016±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	5.821±0.056	5.421±0.028	5.229±0.097	**4.800±0.074**
Wine	0.945±0.044	0.809±0.011	0.773±0.009	0.841±0.007	**0.740±0.011**	0.749±0.010
Yacht	5.388±0.339	4.508±0.158	7.142±0.171	6.245±0.068	4.644±0.237	**3.211±0.120**

Table 2: Average test RMSE in active learning. The suffix _R denotes the random selection control, and the suffix _A denotes active learning.

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.

5.2 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 [30], 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 [20], which in each step attempts to achieve the maximum reduction in posterior entropy. Under the assumption of i.i.d. Gaussian noise (as assumed by all models under consideration), this is equivalent to choosing the unlabeled example with the largest predictive variance. Active learning using the information gain criterion was introduced as a BNN benchmark by [10]; our experiments are based on their protocol.

All methods under consideration use the same neural network architecture and prior, and differ only in the inference procedure. We first investigated how accurately 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 outputs of HMC as the "ground truth" predictive variance. Table 3 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 [10], which select next data with highest predictive variance. As a control, we also 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 all methods is reported in Table 2. 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. 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.

5.3 Reinforcement Learning

We next experimented with using uncertainty to provide intrinsic motivation in reinforcement learning. Reinforcement learning problems with sparse rewards can be particularly challenging, since the

Dataset	BBB	PBP	NNG-MVG	NNG-BlkTri
Boston	0.733±0.021	0.761±0.032	**0.891±0.021**	0.889±0.024
Concrete	0.809±0.027	0.817±0.028	0.913±0.010	**0.922±0.006**
Energy	0.414±0.074	0.471±0.076	0.617±0.087	**0.646±0.088**
Kin8nm	0.681±0.018	0.587±0.021	0.731±0.021	**0.759±0.023**
Naval	0.316±0.094	0.270±0.098	0.596±0.073	**0.598±0.070**
Pow. Plant	0.633±0.046	0.509±0.068	0.829±0.020	**0.853±0.020**
Wine	0.915±0.011	0.883±0.042	0.957±0.009	**0.964±0.006**
Yacht	0.617±0.053	0.620±0.053	0.717±0.072	**0.727±0.070**

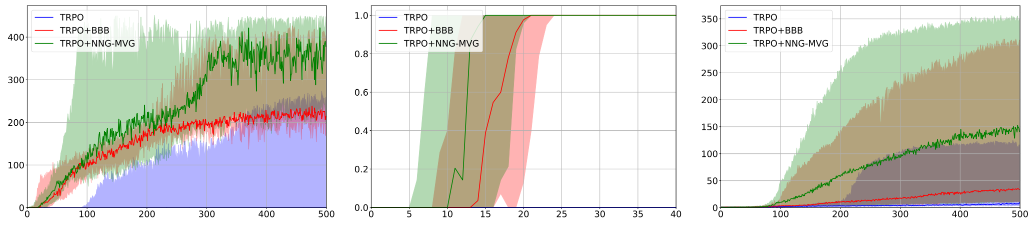
Table 3: Pearson correlation of each algorithm's predictive variances with those of HMC, which we consider as the "ground truth."

agent may need to execute a complex behavior even to obtain a single nonzero reward. Houthooft et al. [13] presented an intrinsic motivation mechanism called 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:

$$r^*(s_t, a_t, s_{t+1}) = r(s_t, a_t) + \eta \mathrm{D}_{\mathrm{KL}}(p(\theta|\xi_t, a_t, s_{t+1}) \,\|\, p(\theta|\xi_t)) \tag{13}$$

where $r(s_t, a_t)$ is the original reward function or external reward, and $\eta \in \mathbb{R}_+$ is a hyperparameter controlling the urge to explore. In above formulation, the true posterior is generally intractable. [13] approximated it using Bayes by Backprop (BBB), a fully factorized Gaussian variational posterior [5]. We experimented with replacing the fully factorized posterior with our NNG-MVG model.

Following the experimental setup of [13], we tested our method in three continuous control tasks and sparsified the rewards in the following way. A reward of $+1$ is given in CartPoleSwingup when $cos(\beta) > 0.8$, with β 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. We compared our NNG-MVG dynamics model with a Gaussian noise baseline, as well as the original VIME formulation using BBB. All experiments are based on the rllab [6] benchmark code base and used TRPO to optimize the policy itself [29]. Because all the methods used the same policy gradient method and only differd only in the dynamics model, the performance differences should reflect the effectiveness of the BNN's uncertainty measure for detecting novelty.



(a) CartPoleSwingup (b) MountainCar (c) DoublePendulum

Figure 2: 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.

Performance is measured by the average return (under the original MDP's rewards, not including the intrinsic term) at each iteration. Fig. 2 shows the performance results in three tasks. Consistently with Houthooft et al. [13], 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 two different types of 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.

References

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

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

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

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

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

[6] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

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

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

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

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

[11] Geoffrey E Hinton and Drew 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[32] Yuhuai Wu, Elman Mansimov, Shun Liao, Roger Grosse, and Jimmy Ba. 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,

$$\begin{aligned}&\arg\min_{\boldsymbol{\theta}\in\mathcal{S}} \alpha(\nabla_{\boldsymbol{\theta}} h)^\top \boldsymbol{\theta} + \mathrm{D}_{\mathrm{KL}}(p_{\boldsymbol{\theta}} \,\|\, p_{\boldsymbol{\theta}_0})\\ \approx &\arg\min_{\boldsymbol{\theta}\in\mathcal{S}} \alpha(\nabla_{\boldsymbol{\theta}} h)^\top \boldsymbol{\theta} + \frac{1}{2}(\boldsymbol{\theta}-\boldsymbol{\theta}_0)^\top \mathbf{F}(\boldsymbol{\theta}-\boldsymbol{\theta}_0)\end{aligned}$$

Then the optimal solution to this optimization problem is given by $\boldsymbol{\theta} - \alpha\mathbf{F}^{-1}\nabla_{\boldsymbol{\theta}} h$. Here $\mathbf{F} = \nabla^2_{\boldsymbol{\theta}}\, \mathrm{D}_{\mathrm{KL}}(p_{\boldsymbol{\theta}} \,\|\, p_{\boldsymbol{\theta}_0}))$ is the Fisher matrix and α is the learning rate. Note that $h(\boldsymbol{\theta})$ and $\mathrm{D}_{\mathrm{KL}}(p_{\boldsymbol{\theta}} \,\|\, p_{\boldsymbol{\theta}_0})$ 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:

$$\mathrm{D}_{\mathrm{KL}}(\mathcal{N} \,\|\, \mathcal{N}_0) = \frac{1}{2}\left[\log\frac{|\boldsymbol{\Sigma}_0|}{|\boldsymbol{\Sigma}|} - d + \mathrm{tr}(\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\Sigma})\right] + (\boldsymbol{\mu}-\boldsymbol{\mu}_0)^\top\boldsymbol{\Sigma}_0^{-1}(\boldsymbol{\mu}-\boldsymbol{\mu}_0) \tag{14}$$

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

$$\begin{aligned}\mathbf{F}_{\boldsymbol{\mu}} &= \nabla^2_{\boldsymbol{\mu}}\mathrm{D}_{\mathrm{KL}} = \boldsymbol{\Sigma}_0^{-1} \approx \boldsymbol{\Sigma}^{-1}\\ \mathbf{F}_{\boldsymbol{\Sigma}} &= \nabla^2_{\boldsymbol{\Sigma}}\mathrm{D}_{\mathrm{KL}} = \frac{1}{2}\boldsymbol{\Sigma}^{-1}\otimes\boldsymbol{\Sigma}^{-1}\end{aligned} \tag{15}$$

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

$$\begin{aligned}\tilde{\nabla}_{\boldsymbol{\mu}} h &= \boldsymbol{\Sigma}\nabla h\\ \tilde{\nabla}_{\boldsymbol{\Sigma}} h &= 2\boldsymbol{\Sigma}\nabla h\boldsymbol{\Sigma}\end{aligned} \tag{16}$$

An analogous derivation gives us $\tilde{\nabla}_{\boldsymbol{\Lambda}} h = 2\boldsymbol{\Lambda}\nabla h\boldsymbol{\Lambda}$. 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

$$\begin{aligned}\tilde{\nabla}_{\boldsymbol{\Sigma}} h &= -2\nabla_{\boldsymbol{\Lambda}} h\\ \tilde{\nabla}_{\boldsymbol{\Lambda}} h &= -2\nabla_{\boldsymbol{\Sigma}} h\end{aligned} \tag{17}$$

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

$$\begin{aligned}\nabla_{\boldsymbol{\mu}}\mathcal{L} &= \mathbb{E}\left[\nabla_{\mathbf{w}}\log p(\mathcal{D}\,|\,\mathbf{w}) + \lambda\nabla_{\mathbf{w}}\log p(\mathbf{w})\right]\\ \nabla_{\boldsymbol{\Sigma}}\mathcal{L} &= \frac{1}{2}\mathbb{E}\left[\nabla^2_{\mathbf{w}}\log p(\mathcal{D}\,|\,\mathbf{w}) + \lambda\nabla^2_{\mathbf{w}}\log p(\mathbf{w})\right] + \frac{\lambda}{2}\boldsymbol{\Sigma}^{-1}\end{aligned} \tag{18}$$

Based on Eqn. 18 and Eqn. 17, the natural gradient is given by:

$$\begin{aligned}\tilde{\nabla}_{\boldsymbol{\mu}}\mathcal{L} &= \boldsymbol{\Lambda}^{-1}\mathbb{E}\left[\nabla_{\mathbf{w}}\log p(\mathcal{D}\,|\,\mathbf{w}) + \lambda\nabla_{\mathbf{w}}\log p(\mathbf{w})\right]\\ \tilde{\nabla}_{\boldsymbol{\Lambda}}\mathcal{L} &= -\mathbb{E}\left[\nabla^2_{\mathbf{w}}\log p(\mathcal{D}\,|\,\mathbf{w}) + \lambda\nabla^2_{\mathbf{w}}\log p(\mathbf{w})\right] - \lambda\boldsymbol{\Lambda}\end{aligned} \tag{19}$$

B Matrix Variate Gaussian

Recently Matrix Variate Gaussian (MVG) distribution are also used in Bayesian neural networks [18] [31]. A matrix variate Gaussian distributions models a Gaussian distribution for a matrix $\mathbf{W} \in \mathbb{R}^{n\times p}$,

$$p(\mathbf{W}|\mathbf{M},\mathbf{U},\mathbf{V}) = \frac{\exp(\frac{1}{2}\,\mathrm{tr}[\mathbf{V}^{-1}(\mathbf{W}-\mathbf{M})^\top\mathbf{U}^{-1}(\mathbf{W}-\mathbf{M})])}{(2\pi)^{np/2}|\mathbf{V}|^{n/2}|\mathbf{U}|^{p/2}} \tag{20}$$

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 $\mathbf{U}$.

$$\mathrm{vec}(\mathbf{W}) \sim \mathcal{N}(\mathrm{vec}(\mathbf{M}), \mathbf{V}\otimes\mathbf{U}) \tag{21}$$

C Experimental Settings

C.1 Experimental settings for regression

The datasets are randomly splitted into training and test sets, with 90% of the data for training and the remaining for testing. To reduce the randomness, we repeat the splitting process for 20 times (except from two largest datasets, i.e., "Year" and "Protein", where we repeat 5 times and 1 times, respectively.) For all datasets except two largest ones, we use neural networks with 50 hidden units. For two largest datasets, we use 100 hidden units. We also introduce a Gamma prior, $p(\tau) = Gam(a_0 = 6, b_0 = 6)$ for the precision of the Gaussian likelihood and include the posterior $q(\tau) = Gam(a_1, b_1)$ into variational objective. In training, the input features and training targets are normalized to be zero mean and unit variance. We remove 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. All models run for 1000 epochs except for "Protein" and "Year" where we use 500. 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 Experimental settings for active learning

Following the experimental protocol in PBP [10], we split each dataset into training and test sets with 20 and 100 data points. All remaining data are included in pool sets. In all experiments, we use a neural network with one hidden layer and 10 hidden units.

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

C.3 Experimental settings for reinforcement learning

In all three tasks, CartPoleSwingup, MountainCar and DoublePendulum, we use one-layer Bayesian Neural Network with 32 hidden units for both BBB and NNG-MVG. And we use rectified linear unit (RELU) as our activation function. The number of samples drawn from variational posterior is fixed to 10 in the training process. For TRPO, the batch size is 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 is updated in each epoch with 500 iterations and 10 batch size. For the policy network, one-layer network with 32 tanh units is used.

D Results for two-layer nets on UCI datasets

	Test RMSE				Test log-likelihood			
Dataset	BBB	PBP	PBP_MV	NNG-MVG	BBB	PBP	PBP_MV	NNG-MVG
Boston	2.982±0.077	3.134±0.139	3.112±0.151	**2.451±0.022**	-2.803±0.010	-2.527±0.046	-2.539±0.076	**-2.383±0.012**
Concrete	6.010±0.067	5.170±0.125	5.075±0.138	**4.976±0.057**	-3.216±0.008	-3.061±0.026	-3.043±0.031	**-2.993±0.014**
Energy	0.596±0.014	0.791±0.046	**0.448±0.013**	0.462±0.011	-1.589±0.006	-1.724±0.012	**-1.014±0.011**	-1.452±0.003
Kin8nm	0.069±0.000	0.072±0.000	0.069±0.003	**0.068±0.001**	1.265±0.004	1.219±0.003	1.259±0.008	**1.273±0.001**
Naval	**0.000±0.000**	0.005±0.000	0.002±0.000	**0.000±0.000**	6.651±0.040	3.172±0.047	4.853±0.059	**7.071±0.013**
Pow. Plant	4.077±0.011	4.011±0.034	3.908±0.039	**3.881±0.009**	-2.815±0.002	-2.804±0.007	-2.781±0.008	**-2.762±0.003**
Protein	3.866±0.010	4.553 ±0.020	3.939±0.019	**3.713±0.014**	-2.767±0.002	-2.933±0.005	-2.772±0.008	**-2.713±0.004**
Wine	0.646±0.004	0.636±0.008	0.642±0.007	**0.631±0.004**	-0.988±0.005	**-0.952±0.013**	-0.972±0.009	-0.954±0.008
Yacht	1.590±0.078	0.864±0.042	0.806±0.061	**0.783±0.013**	-2.681±0.019	-1.778±0.015	**-1.642±0.021**	-2.249±0.014
Year	8.791±NA	8.787±NA	8.721±NA	**8.617±NA**	-3.570±NA	-3.365±NA	**-3.332±NA**	-3.569±NA

Table 4: Averaged predictions with standard errors in terms of RMSE, log-likelihood for the regression dataset using two-layer neural networks. PBP_MV denotes the method proposed by [10].