Supplementary Material for Noisy Natural Gradient as Variational Inference

1. 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:

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

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

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

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

$$\begin{aligned}\tilde{\nabla}_{\boldsymbol{\mu}} h &= \boldsymbol{\Sigma}\nabla_{\boldsymbol{\mu}} h \\ \tilde{\nabla}_{\boldsymbol{\Sigma}} h &= 2\boldsymbol{\Sigma}\nabla_{\boldsymbol{\Sigma}} h \boldsymbol{\Sigma}\end{aligned} \tag{3}$$

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

$$\begin{aligned}\tilde{\nabla}_{\boldsymbol{\Sigma}} h &= -2\nabla_{\boldsymbol{\Lambda}} h \\ \tilde{\nabla}_{\boldsymbol{\Lambda}} h &= -2\nabla_{\boldsymbol{\Sigma}} h\end{aligned} \tag{4}$$

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

$$\begin{aligned}\nabla_{\boldsymbol{\mu}}\mathcal{L} &= \mathbb{E}\left[\nabla_{\mathbf{w}} \log p(\mathcal{D} \,|\, \mathbf{w}) + \lambda \nabla_{\mathbf{w}} \log p(\mathbf{w})\right] \\ \nabla_{\boldsymbol{\Sigma}}\mathcal{L} &= \frac{1}{2}\mathbb{E}\left[\nabla^2_{\mathbf{w}} \log p(\mathcal{D} \,|\, \mathbf{w}) + \lambda \nabla^2_{\mathbf{w}} \log p(\mathbf{w})\right] + \frac{\lambda}{2}\boldsymbol{\Sigma}^{-1}\end{aligned} \tag{5}$$

Based on eq. (5) and eq. (4), the natural gradient is given by:

$$\begin{aligned}\tilde{\nabla}_{\boldsymbol{\mu}}\mathcal{L} &= \boldsymbol{\Lambda}^{-1}\mathbb{E}\left[\nabla_{\mathbf{w}} \log p(\mathcal{D} \,|\, \mathbf{w}) + \lambda \nabla_{\mathbf{w}} \log p(\mathbf{w})\right] \\ \tilde{\nabla}_{\boldsymbol{\Lambda}}\mathcal{L} &= -\mathbb{E}\left[\nabla^2_{\mathbf{w}} \log p(\mathcal{D} \,|\, \mathbf{w}) + \lambda \nabla^2_{\mathbf{w}} \log p(\mathbf{w})\right] - \lambda\boldsymbol{\Lambda}\end{aligned} \tag{6}$$

2. 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 $\mathbf{W} \in \mathbb{R}^{n\times p}$,

$$\begin{aligned}&p(\mathbf{W}|\mathbf{M}, \mathbf{U}, \mathbf{V}) \\ &= \frac{\exp(\frac{1}{2}\mathrm{tr}[\mathbf{V}^{-1}(\mathbf{W}-\mathbf{M})^\top \mathbf{U}^{-1}(\mathbf{W}-\mathbf{M})])}{(2\pi)^{np/2}|\mathbf{V}|^{n/2}|\mathbf{U}|^{p/2}}\end{aligned} \tag{7}$$

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 $\mathbf{U}$.

$$\mathrm{vec}(\mathbf{W}) \sim \mathcal{N}(\mathrm{vec}(\mathbf{M}), \mathbf{V} \otimes \mathbf{U}) \tag{8}$$

3. Implementation Details

3.1. 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 $q(\mathbf{w}, \lambda) = q(\mathbf{w})q(\tau), q(\tau) = \text{Gamma}(\alpha^\tau, \beta^\tau)$, then the expected likelihood $\mathcal{L}_r$ can be computed as

$$\begin{aligned}\mathcal{L}_r &= \mathbb{E}_{q(\mathbf{w})}\mathbb{E}_{q(\tau)} \log p(y|\mathbf{x}, \mathbf{w}, \tau) \\ &= \mathbb{E}_{q(\mathbf{w})}\mathbb{E}_{q(\tau)} \log \mathcal{N}(y|\hat{y}(\mathbf{x}, \mathbf{w}), \frac{1}{\tau}) \\ &= \frac{1}{2}\mathbb{E}_{q(\mathbf{w})}\mathbb{E}_{q(\tau)}[\log \tau - \tau(y - \hat{y}(\mathbf{x}, \mathbf{w}))^2 - \log 2\pi] \\ &= \mathbb{E}_{q(\mathbf{w})}[\psi(\alpha^\tau) - \log \beta^\tau - \frac{\alpha^\tau}{\beta^\tau}(y - \hat{y}(\mathbf{x}, \mathbf{w}))^2 - \log 2\pi]\end{aligned} \tag{9}$$

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

$$\mathcal{L} = \mathcal{L}_r - \mathrm{D}_{\mathrm{KL}}(q(\mathbf{w})\|p(\mathbf{w})) - \mathrm{D}_{\mathrm{KL}}(q(\tau)\|p(\tau)) \tag{10}$$

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.

3.2. 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:

$$\mathrm{D}_{\mathrm{KL}}(q \,\|\, p) \approx \frac{1}{2}\mathbf{v}^\top \mathbf{F}\mathbf{v} \tag{11}$$

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:

$$\tilde{\alpha} = \min\left(\tilde{\alpha}_{\text{max}}, \sqrt{\frac{c}{\mathbf{v}^\top\mathbf{F}\mathbf{v}}}\right) \tag{12}$$

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 $\tilde{\alpha}_{\text{max}}$ 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.

3.3. 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,

$$\rho(X, Y) = \frac{\mathrm{E}[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X\sigma_Y} \tag{13}$$

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.

3.4. 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 iterations 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 $q(\boldsymbol{\phi})$ as prior.)

$$\begin{aligned}&\mathrm{D}_{\mathrm{KL}}\left(q(\boldsymbol{\phi}')\,\|\,q(\boldsymbol{\phi})\right) = \\ &\frac{1}{2}\tilde{\alpha}^2\begin{bmatrix}\mathrm{vec}\left\{\nabla_{\boldsymbol{\mu}}\mathcal{L}\right\}\\ \mathrm{vec}\left\{\nabla_{\boldsymbol{\Sigma}}\mathcal{L}\right\}\end{bmatrix}^T\begin{bmatrix}\mathbf{F}_{\boldsymbol{\mu}}^{-1} & \\ & \mathbf{F}_{\boldsymbol{\Sigma}}^{-1}\end{bmatrix}\begin{bmatrix}\mathrm{vec}\left\{\nabla_{\boldsymbol{\mu}}\mathcal{L}\right\}\\ \mathrm{vec}\left\{\nabla_{\boldsymbol{\Sigma}}\mathcal{L}\right\}\end{bmatrix}\end{aligned} \tag{14}$$

Where $\tilde{\alpha}$ is the step-size. As shown in eq. (2), the Fisher matrix for $\boldsymbol{\mu}$ is given by $\mathbf{F}_{\boldsymbol{\mu}} = \boldsymbol{\Sigma}^{-1}$, thus the first term in eq. (14) 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

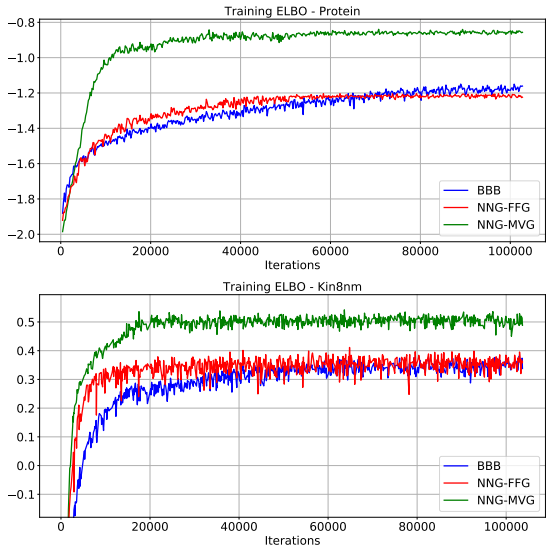

Figure 1: Training curves for all three methods. For each method, we tuned the learning rate for updating the posterior mean. Note that BBB and NNG-FFG use the same form of q, while NNG-MVG uses a more flexible q distribution.

$\mathbf{F}_{\boldsymbol{\Sigma}}$ with $\frac{1}{2}\boldsymbol{\Sigma}^{-1} \otimes \boldsymbol{\Sigma}^{-1}$, we get the following identity

$$\begin{aligned}&\mathrm{vec}\left\{\nabla_{\boldsymbol{\Sigma}}\mathcal{L}\right\}^T\mathbf{F}_{\boldsymbol{\Sigma}}\,\mathrm{vec}\left\{\nabla_{\boldsymbol{\Sigma}}\mathcal{L}\right\} = \\ &\begin{bmatrix}\mathrm{vec}\left\{\nabla_{\boldsymbol{\Sigma}_1}\mathcal{L}\right\}\\ \mathrm{vec}\left\{\nabla_{\boldsymbol{\Sigma}_2}\mathcal{L}\right\}\end{bmatrix}^T\begin{bmatrix}\mathbf{F}_{\boldsymbol{\Sigma}_1} & \mathbf{F}\\ \mathbf{F}^T & \mathbf{F}_{\boldsymbol{\Sigma}_2}\end{bmatrix}\begin{bmatrix}\mathrm{vec}\left\{\nabla_{\boldsymbol{\Sigma}_1}\mathcal{L}\right\}\\ \mathrm{vec}\left\{\nabla_{\boldsymbol{\Sigma}_2}\mathcal{L}\right\}\end{bmatrix}\end{aligned} \tag{15}$$

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

$$\begin{aligned}\mathbf{F}_{\boldsymbol{\Sigma}_1} &= \frac{n}{2}\left(\boldsymbol{\Sigma}_1^{-1} \otimes \boldsymbol{\Sigma}_1^{-1}\right)\\ \mathbf{F}_{\boldsymbol{\Sigma}_2} &= \frac{m}{2}\left(\boldsymbol{\Sigma}_2^{-1} \otimes \boldsymbol{\Sigma}_2^{-1}\right)\end{aligned} \tag{16}$$

By further ignoring off-diagonal block $\mathbf{F}$ in eq. (15), we can decompose $\mathrm{vec}\left\{\nabla_{\boldsymbol{\Sigma}}\mathcal{L}\right\}^T\mathbf{F}_{\boldsymbol{\Sigma}}^{-1}\,\mathrm{vec}\left\{\nabla_{\boldsymbol{\Sigma}}\mathcal{L}\right\}$ into two terms,

$$\begin{aligned}&\mathrm{vec}\left\{\nabla_{\boldsymbol{\Sigma}_1}\mathcal{L}\right\}^T\mathbf{F}_{\boldsymbol{\Sigma}_1}^{-1}\,\mathrm{vec}\left\{\nabla_{\boldsymbol{\Sigma}_1}\mathcal{L}\right\}\\ &= \frac{2}{n}\,\mathrm{vec}\left\{\nabla_{\boldsymbol{\Sigma}_1}\mathcal{L}\right\}^T\mathrm{vec}\left(\boldsymbol{\Sigma}_1^{-1}\nabla_{\boldsymbol{\Sigma}_1}\mathcal{L}\boldsymbol{\Sigma}_1^{-1}\right)\end{aligned} \tag{17}$$

and

$$\begin{aligned}&\mathrm{vec}\left\{\nabla_{\boldsymbol{\Sigma}_2}\mathcal{L}\right\}^T\mathbf{F}_{\boldsymbol{\Sigma}_2}^{-1}\,\mathrm{vec}\left\{\nabla_{\boldsymbol{\Sigma}_2}\mathcal{L}\right\}\\ &= \frac{2}{m}\,\mathrm{vec}\left\{\nabla_{\boldsymbol{\Sigma}_2}\mathcal{L}\right\}^T\mathrm{vec}\left(\boldsymbol{\Sigma}_2^{-1}\nabla_{\boldsymbol{\Sigma}_2}\mathcal{L}\boldsymbol{\Sigma}_2^{-1}\right)\end{aligned} \tag{18}$$

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

4. Additional Results

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 1. Note that VMG introduced pseudo input-output pairs to enhance the flexibility of posterior distribution.

Table 1: 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 1 . 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.

References

Asuncion, A. and Newman, D. Uci machine learning repository, 2007.

Ba, J., Grosse, R., and Martens, J. Distributed second-order optimization using kronecker-factored approximations. 2016.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

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

Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pp. 1109–1117, 2016.

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

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

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

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