Uncertainty Estimation with Noisy Optimizers

Orhun Güley ORHUN.GUELEY@TUM.DE
03721364

1. Introduction

Deep learning methods have been extremely successful in the last decade. Specifically in tasks such as only the prediction accuracy are taken into account, they became preeminent. But in tasks that requires high risk or interpretability, high prediction accuracy is not the only metric that needed to be satisfied. In those kinds of tasks such as autonomous driving, medical diagnosis and finance, in order to make reliable decisions, we need uncertainty measures. As an example, autonomous vehicles are constantly in need to make reliable decisions since one minor prediction might cost human life. Bayesian Inference is a very effective technique that allows models to quantify uncertainty, meaning that we will have a metric that how certain our model is on a specific input.

For simpler models, it is possible to quantify uncertainty by directly deriving the equation for posterior distribution by using Bayes rule:

$$p(w|\mathcal{D}) = \frac{p(\mathcal{D}|w)p(w)}{p(\mathcal{D})} = \frac{p(\mathcal{D}|w)p(w)}{\int p(\mathcal{D}|w)p(w)} = \frac{p(\mathcal{D}|w)p(w)}{\sum_{i=1}^{n} p(\mathcal{D}|w)p(w)} \tag{1}$$

However, for more complex models such as deep neural networks, it is not possible to find closed for solution for the posterior. As a consequence, a common framework to integrate uncertainty measure to deep neural networks is to find approximations for the posterior. Some of these approximations techniques such as Markov Chain Monte Carlo(MCMC) and variational inference(VI) with or without mean-field approximation have been very popular in Bayesian deep learning. Learning in deep learning is a nonlinear optimization problem. In the past years, stochastic gradient descent(SGD) and its variations have been very popular. These variations include momentum based methods, which the algorithm tries to estimate the momentum, that accelarates the convergence of the loss. In this report, I will be summarizing the approximation of Variational Inference methods by integrating natural gradient method into the popular optimizers used in deep learning.

Natural gradient method is applicable to probabilistic optimization models, which makes it a good alternative to traditional stochastic gradient descent methods. Briefly, natural gradient methods scale the gradient by implementing the steepest direction in the distribution space, instead of the space of parameters of the models, which is Euclidean. Intuitively, this approach make sense because it is not likely that the true distance between distributions are Euclidean. It has been shown that by adding noise/perturbation to popular nonlinear optimizers such as Adam and RMSProp, one can perform variational inference.

2. Background

2.1 Bayesian Inference

As briefly explained in the previous eqn. (1), given a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)_{i=1}^n\}$, Bayesian neural networks tries to find the posterior distribution of the weights, given the training data, $p(w|\mathcal{D})$. The calculation of posterior is obtained by putting a prior distribution $p(w)$ on weights and then using Bayes

rule(1), we have the need calculate $p(\mathcal{D}|w)p(w)/p(\mathcal{D})$. The untractable part of this equation is the calculation of the normalizing constant, in which we need to calculate the integral $p(\mathcal{D}) = \int p(\mathcal{D}|w)p(w)$. Since the calculation of posterior distribution is not tractable in neural networks, Bayesian methods tries to find an approximation for true posterior $p(w|\mathcal{D})$. Several popular methods such as variational inference, Markov Chain Monte Carlo have been used for approximating the posterior distribution but in practice, they are rarely used due to their computational expense. As an alternative for those methods, less theoretical methods such as Monte Carlo Dropout(MC-Dropout) are used in the last years.

In the papers I am going to summarize(Zhang et al., 2018; Khan et al., 2018; Osawa et al., 2019), it has been shown that by modifying and perturbing some of the commonly used nonlinear optimizers, they perform variational inference.

2.1.1 KULLBACK-LEIBLER DIVERGENCE

Kullback-Leibler divergence has its roots from relative entropy in information theory. It can be defined as a non-symmetric and non-negative measure between two probability distributions. It can be defines as follows:

$$\mathbb{KL}\left(q(x)\|p(x)\right) = \int q(x)\log\frac{q(x)}{p(x)}dx = \sum_{x\in X} p(x)\log\frac{p(x)}{q(x)}$$

where the integral version represents the continuous case and the summation version represents the discrete case. KL divergence is a frequently used measure while doing Bayesian inference using variational inference, where we try to approximate the true posterior with a candidate distribution q.

2.1.2 VARIATIONAL INFERENCE

Since the calculation of posterior distribution is not tractable in complex models such as deep neural networks, we use variational inference, where we use a distribution $q_\theta(w)$, parametrized by θ. As an example, if we model our posterior to be a Gaussian distribution $p(w|\mathcal{D}) := \mathcal{N}(w|\mu,\, diag(\sigma^2))$ or $p(w|\mathcal{D}) := \mathcal{N}(w|\mu,\, \Sigma)$, the parameters would be $\theta = (\mu, \sigma)$ or $\theta = (\mu, \Sigma)$, where $\mu, \sigma \in \mathbb{R}^D$ and $\Sigma \in \mathbb{R}^{DxD}$. In order to track how close is our approximate posterior distribution q_θ to the true posterior distribution, we need a distance metric. We know from the previous section that KL divergence can be interpreted as a distance metric(theoretically not a distance metric since $\mathbb{KL}[x\|y] \neq \mathbb{KL}[y\|x]$) between two probability distributions, one solution proposed by Bishop (2006) is finding the parameters of distribution q_θ by minimizing the $\mathbb{KL}[q(\mathbf{w})\|p(w|\mathcal{D})]$. We see that minimizing the KL divergence is equal to maximizing a variational objective.

$$\begin{aligned}
\theta^\star &= \arg\min_\theta \mathbb{KL}[q_\theta(\mathbf{w})\|p(w|\mathcal{D})] \\
&= \arg\min_\theta \int q_\theta(\mathbf{w})\log\frac{q_\theta(\mathbf{w})}{p(w|\mathcal{D})}\mathrm{d}\mathbf{w} \\
&= \arg\min_\theta \int q_\theta(\mathbf{w})\log\frac{q_\theta(\mathbf{w})p(\mathcal{D})}{p(w)p(\mathcal{D}|w)}\mathrm{d}\mathbf{w} \\
&= \arg\min_\theta \mathbb{E}_{q_\theta(w)}[\log q_\theta(w) - \log p(\mathcal{D}|w) - \log p(\mathbf{w}) + \underbrace{\log p(\mathcal{D})}_{\substack{\text{Not}\\ \text{dependent}\\ \text{of }\theta}}] \qquad (2) \\
&= \arg\min_\theta \mathbb{E}_{q_\theta(w)}[\log q_\theta(w) - \log p(\mathcal{D}|w) - \log p(\mathbf{w})] \\
&= \arg\min_\theta \underbrace{\mathbb{KL}[q_\theta(\mathbf{w} \mid \theta)\|p(\mathbf{w})] - \mathbb{E}_{q(\mathbf{w}|\theta)}[\log p(\mathcal{D} \mid \mathbf{w})]}_{\mathcal{F}(\mathcal{D},\theta)=-\mathcal{L}(\boldsymbol{\theta},q)} \qquad (3)
\end{aligned}$$

The objective function stated in eqn.(3) is called variational free energy. If we denote the variational free energy function as $\mathcal{F}(\mathcal{D}, \theta)$, we can rewrite the eqn.(3) as

$$\mathbb{KL}[q_\theta(\mathbf{w})\|p(w|\mathcal{D})] = \mathcal{F}(\mathcal{D},\theta) + \log p(\mathcal{D}) \qquad (4)$$

Additionally, we can interpret the negative variational free energy function as a lower bound for the log marginal likelihood term $\log p(\mathcal{D})$. Let's define the term $\mathcal{L}(\boldsymbol{\theta}, q)$ as $\mathcal{L}(\boldsymbol{\theta}, q) = -\mathcal{F}(\mathcal{D}, \theta)$. $\log p(\mathcal{D})$ is not dependent on parameters θ. Then, we can rewrite the eqn(4) as

$$\mathbb{KL}[q_\theta(\mathbf{w}) \| p(w|\mathcal{D})] = -\mathcal{L}(\boldsymbol{\theta}, q) + \log p(\mathcal{D}) \tag{5}$$

As stated in the Section(2.1.1), the KL divergence is always non-negative. That's why the $\mathcal{L}(\boldsymbol{\theta}, q)$ describes a lower bound for the marginal likelihood term $\log p(\mathcal{D})$, and $\mathcal{L}(\boldsymbol{\theta}, q)$ is called evidence lower bound(ELBO).(6)

$$\mathcal{L}(\boldsymbol{\theta}, q) = \log p(\mathcal{D}) - \underbrace{\mathbb{KL}[q_\theta(\mathbf{w}) \| p(w|\mathcal{D})]}_{\geq 0} \tag{6}$$

$$\mathcal{L}(\boldsymbol{\theta}, q) \leq \log p(\mathcal{D}) \tag{7}$$

As stated in the Section(2.1.1), the KL divergence is always non-negative. That's why the $\mathcal{L}(\boldsymbol{\theta})$ describes a lower bound for the marginal likelihood term $\log p(\mathcal{D})$, and $\mathcal{L}(\boldsymbol{\theta})$ is called evidence lower bound(ELBO). As a result, minimizing the KL divergence is equal to maximizing the ELBO term.

2.1.3 REPARAMETRIZATION TRICK

Reparametrization trick is a simple deterministic transformation that helps while sampling from a distribution. In case that we set out variational posterior to a Gaussian distribution $p(w|\mathcal{D}) := \mathcal{N}(w|\mu, \Sigma)$, our estimated parameters will be $\theta = (\mu, \Sigma)$. Normally, while doing backpropagation we need to take the gradients for the Gaussian distribution. But using reparametrization trick stated below, we can free our gradients of μ and Σ from the Gaussian distribution by using a parameter-free noise ϵ.

1. Draw samples from noise $\epsilon \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
2. Set $\mathbf{w} = t(\epsilon, \theta = \{\mu, \Sigma\}) = \mu + \Sigma \circ \epsilon$

2.2 Natural Gradient

2.2.1 INTRODUCTION

In optimization of the deep learning cost functions, learning takes place in the space of parameters, which is not Euclidean but actually Riemannian. And popular nonlinear optimizers such as Adam, stochastic gradient descent(SGD) are operating on the Euclidean space. Natural gradient method proposed by (Amari, 1997, 2016) takes its roots from information geometry, which aims to change the gradients direction from the steepest direction in the Euclidean space to the steepest direction the Riemannian space. Let's me briefly introduce what is Riemannian metric and Fisher Information Matrix.

2.2.2 RIEMANNIAN METRICS AND FISHER INFORMATION MATRIX

Let's consider a data point $\theta = (\theta^{(1)}, ..., \theta^{(n)})$ in a n-dimensional manifold $\mathbf{M}$. The tangent space $\mathbf{T}_\theta$ at point θ is defined as a vector space spanned by n tangent vectors along the coordinate curves of $\theta_\mathbf{i}$, denoted as $\{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$ by Amari (2016). And these tangent vector $\mathbf{e}_i$ is identified with the partial derivative operator as below.

$$\boldsymbol{e}_i \approx \partial_i = \frac{\partial}{\partial \theta^{(i)}} \qquad \boldsymbol{e}_i f = \partial_i f(\boldsymbol{\theta})$$

where $f(\boldsymbol{\theta})$ is a differentiable function. For the manifold of probability distributions, the expression for a tangent vector is

$$\boldsymbol{e}_i \approx \partial_i \log p(x|\theta)$$

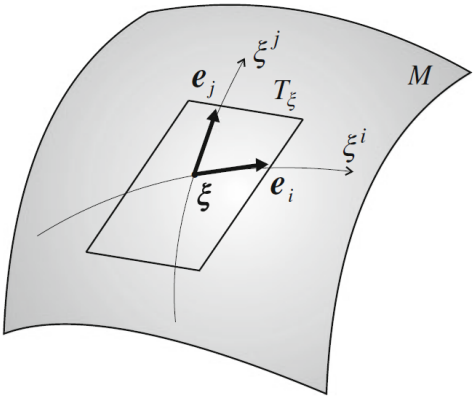


Figure 1: Tangent space T_θ and basis/tangent vectors $\boldsymbol{e}_i$.(Figure taken from Amari (2016))

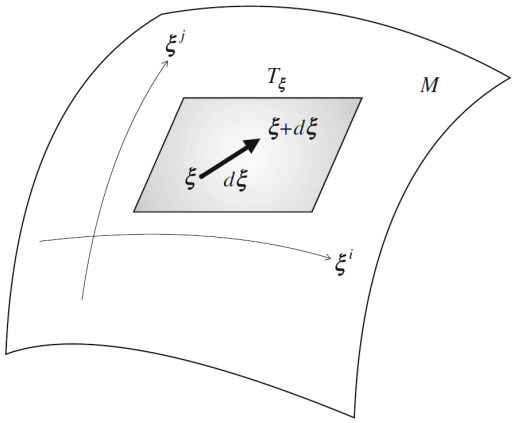


Figure 2: Infinitesimal vector ∂_i and T_θ.(Figure taken from Amari (2016))

Please see 1 and 2 for a visual geometric interpretation.

Let $\mathbf{G} = (g_{ij})$ a matrix, which is consisting of the inner products of the basis vectors of our tangent space $\mathbf{T}_\theta$ below.

$$g_{ij}(\theta) = \langle \mathbf{e}_i, \mathbf{e}_j \rangle$$

$\mathbf{G}$ is a positive-definite matrix which is dependent on θ and it is a metric tensor where its components change to

$$g_{\theta_o \theta_n} = J^i_{\theta_o} J^j_{\theta_n} g_{ij} \quad \text{where} \quad J_i = \frac{\partial \theta^i_n}{\partial \theta^j_o} \quad \text{and} \quad J_i = \frac{\partial \theta^j_o}{\partial \theta^i_n}$$

by coordinate transformation. A manifold is a Riemannian manifold when a metric tensor defined.

Back to our case, we have a manifold of probability distributions and we can define an inner product by using the stochastic expression

$$\langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle = \mathrm{E}\left[\partial_i \log p(x|\boldsymbol{\theta}) \partial_j \log p(x|\boldsymbol{\theta}^{\mathrm{T}})\right] \tag{8}$$

In matrix notation, we have the resulting matrix

$$\mathrm{F} = \mathop{\mathbb{E}}_{p(x|\theta)}\left[\nabla_\theta \log p(x|\theta) \nabla_\theta \log p(x|\theta)^{\mathrm{T}}\right] \tag{9}$$

which is called the Fisher Information Matrix(FIM). It is trivial to see that the FIM is the covariance matrix of our likelihood function and it intuitively makes sense that we are scaling our gradient with a covariance matrix of our likelihood function in order to change gradient directions to probability distribution space. In our case, since we are using using a variational objective function instead of the maximum likelihood, our FIM becomes

$$\mathrm{F} = \mathop{\mathbb{E}}_{q_\theta(\mathbf{w})}\left[\nabla_\theta q_\theta(\mathbf{w}) \nabla_\theta q_\theta(\mathbf{w})^{\mathrm{T}}\right] \tag{10}$$

2.2.3 NATURAL GRADIENT: STEEPEST DIRECTION IN THE RIEMANNIAN MANIFOLD

As it was briefly mentioned in Section 2.2, natural gradient method scales the gradient with a Riemannian metric, which corresponds to the inverse of its FIM of the score function. In probabilistic setting, this score function is a probability distribution. The likelihood function can be a proper choice in this task, but in a Bayesian setting that we are trying to approximate the true posterior, our variational posterior $q_\theta(\mathbf{w})$ would be the right choice for score function. This scaling help the optimizer to operate in a

Riemannian manifold.

Deep neural networks are placed into the category of singular models since they have unidentifiable regions in their objective functions. A model is classified as a singular model(Lin) if it has unidentifiable or $det(F) = 0$ for some $\mathbf{w}$. In such regions, learning becomes really slow and this issue is known as the plateau phenomena(Amari (2016)). Natural gradient method overcomes this issue.

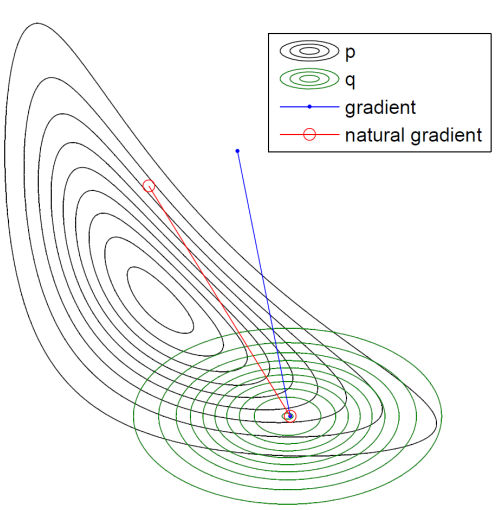


Figure 3: Gradient and vs natural gradient directions for mean of variational distribution q. VI with a diagonal covariance is applied to the posterior $p(x, y) \propto \exp\left[-9(xy-1)^2 - x^2 - y^2\right]$. Figure taken from Kuusela et al. (2009).

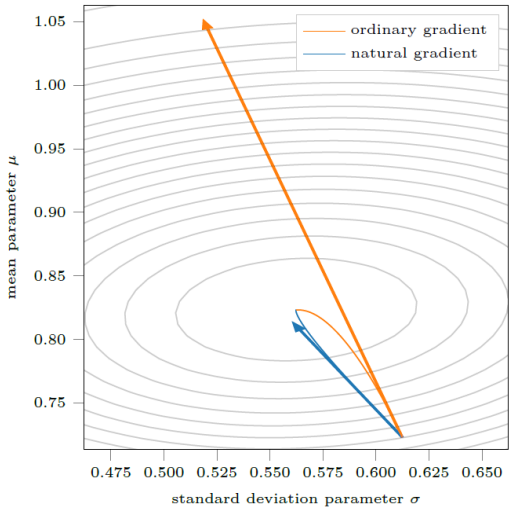


Figure 4: Arrows correspond the gradients and the curves correspond the path of gradients. Figure taken from Salimbeni et al. (2018)

A nice visual example comparing natural gradient with gradient is given by the Figure 3. Kuusela et al. (2009) used natural gradient method for their work on mixture of Gaussians. Figure 3 clearly that the natural gradient enhances the updates in the directions with more uncertainty. Another visual example can be find in Figure 4, which is taken from the work by Salimbeni et al. (2018). The contours in the 4 is from a Gaussian Process(GP) with a Bernoulli likelihood and variational posterior. It is trivial the see that the direction in natural gradient is much more scaled than the ordinary gradient. Additionally the path for natural gradient is taking small steps and does not follow the contours different from ordinary gradient's path.

Let $\tilde{\nabla}_\theta$ be the term for natural gradient. Then, the mathematical equation for the natural gradient of our variational objective is defined as

$$\begin{aligned}\tilde{\nabla}_\theta \mathcal{L}(\boldsymbol{\theta}, q) &\overset{\text{def}}{=} \nabla_\theta \mathcal{L}(\theta) \mathbb{E}_{\mathbf{z}} \left[\left(\nabla \log p_\theta(\mathbf{z})\right)^T \left(\nabla \log p_\theta(\mathbf{z})\right) \right]^{-1} \\ &\overset{\text{def}}{=} \nabla_\theta \mathcal{L}(\boldsymbol{\theta}, q) \mathbf{F}^{-1}\end{aligned} \tag{11}$$

Until now, we talked about natural gradient method enhance the gradient descent by scaling the gradient with FIM, which allows us to find the steepest direction in in the KL-divergence in each step. As formalized by Pascanu and Bengio (2014), let us define our statement as a constrained optimization problem. We are trying to find the $\Delta\theta$ that minimizes the second order Taylor expansion of $\mathcal{L}$ with the constraint that the KL-divergence between p_θ and $p_{\theta+\Delta\theta}$ is a constant:

$$\begin{aligned}\arg\min_{\Delta\theta} \quad & \mathcal{L}(\theta + \Delta\theta) \\ s.t. \quad & KL\left(p_\theta \| p_{\theta+\Delta\theta}\right) = \text{ const.}\end{aligned} \tag{12}$$

By setting the KL divergence as a constant, we guarantee that we will have constant speed while moving along the manifold of our objective function, which prevents the update slowed down in a curvature or plateau.
Assuming that $\Delta\theta \to 0$, we can approximate KL divergence by its second order Taylor series:

$$\begin{aligned} KL\left(p_\theta \| p_{\theta+\Delta\theta}\right) &\approx \underbrace{KL\left(p_\theta \| p_\theta\right)}_{=0} - \underbrace{\mathbb{E}_{\mathbf{z}}\left[\nabla \log p_\theta(\mathbf{z})\right]}_{=0} \Delta\theta - \tfrac{1}{2}\Delta\theta^T \mathbb{E}_{\mathbf{z}}\left[\nabla^2 \log p_\theta\right] \Delta\theta \\ &= \tfrac{1}{2}\Delta\theta^T \mathbb{E}_{\mathbf{z}}\left[-\nabla^2 \log p_\theta(\mathbf{z})\right] \Delta\theta \\ &= \tfrac{1}{2}\Delta\theta^T \mathbf{F} \Delta\theta \end{aligned} \tag{13}$$

As described in Section 2.1.1, KL-divergence between same distributions is equal to zero. And the second term is also zero, due to $\mathbb{E}[\nabla \log p(x \mid \theta)]_{p(x|\theta)} = \int \nabla \log p(x|\theta) p(x|\theta) \mathrm{d}x = \int \frac{\nabla p(x|\theta)}{p(x|\theta)} p(x \mid \theta) \mathrm{d}x = \int \nabla p(x|\theta) \mathrm{d}x = \nabla \int p(x|\theta) \mathrm{d}x = \nabla 1 = 0$

Since we have derived the second order Taylor approximation of KL-divergence, now we can find the distance that minimizes our loss function $\mathcal{L}(\theta)$. Let me denote the minimum $\Delta\theta$ that gives us the constant change in KL-divergence as $(\Delta\theta)^*$. Put simply: the distance $(\Delta\theta)^*$ gives us the most change in KL-divergence per unit distance. We can find the minimum distance from this minimization:

$$\begin{aligned} (\Delta\theta)^* = \arg\min_{\Delta\theta} \quad & \mathcal{L}(\theta + \Delta\theta) \\ \text{s.t.} \quad & KL\left(p_\theta \| p_{\theta+\Delta\theta}\right) = \mathrm{c} \end{aligned} \tag{14}$$

We can write the equivalent optimization problem in Lagrangian formulation in order to transform the constrained optimization problem to a unconstrained optimization problem. We use the second order Taylor series approximation of KL-divergence we had in eqn.(13) and the first order Taylor approximation of $\mathcal{L}(\theta + d)$ to approximate the objective function.

$$\begin{aligned} (\Delta\theta)^* &= \arg\min_{\Delta\theta} \mathcal{L}(\theta + \Delta\theta) + \lambda\left(\mathrm{KL}\left[p_\theta \| p_{\theta+\Delta\theta}\right] - c\right) \\ &\approx \arg\min_{\Delta\theta} \mathcal{L}(\theta) + \nabla_\theta \mathcal{L}(\theta)^{\mathrm{T}}(\Delta\theta) + \frac{1}{2}\lambda(\Delta\theta)^{\mathrm{T}}\mathrm{F}(\Delta\theta) - \lambda c \end{aligned} \tag{15}$$

Since we get rid of the constraints, we just need to set the derivative with respect to d to zero:

$$\begin{aligned} 0 &= \frac{\partial}{\partial(\Delta\theta)} \mathcal{L}(\theta) + \nabla_\theta \mathcal{L}(\theta)^{\mathrm{T}}(\Delta\theta) + \frac{1}{2}\lambda(\Delta\theta)^{\mathrm{T}}\mathrm{F}(\Delta\theta) - \lambda c \\ &= \nabla_\theta \mathcal{L}(\theta) + \lambda \mathrm{F}(\Delta\theta) \\ \lambda \mathrm{F}(\Delta\theta) &= -\nabla_\theta \mathcal{L}(\theta) \\ (\Delta\theta) &= -\frac{1}{\lambda}\mathrm{F}^{-1} \nabla_\theta \mathcal{L}(\theta) \end{aligned} \tag{16}$$

Now, we have that up to a constant factor $1/\lambda$, we get the optimal direction of the gradients by taking account the local curvature space defined by the inverse of FIM. The term $1/\lambda$ can be interpreted as a learning rate and at the end we have the natural gradient definition we defined at eqn.(11).

3. Bayesian Inference with Noisy/Perturbed Natural Gradients

We were introduced in Section 2.1 that how to Bayesian inference work in deep neural networks. We can make use of reparametrization trick by using a noise from a Gaussian distribution in order to prevent avoid using approximations for the gradient of Gaussian distribution. For stochastic gradient ascent, this is a straightforward task. We can update our variational parameters as below:

$$\mu_{t+1} = \mu_t + \rho_t \hat{\nabla}_\mu \mathcal{L}_t, \quad \sigma_{t+1} = \sigma_t + \delta_t \hat{\nabla}_\sigma \mathcal{L}_t \tag{17}$$

where t is the iteration number, $\hat{\nabla}_\mu \mathcal{L}_t$ is the gradient estimates for the batch and ρ_t, δ_t are the learning rates.

In machine learning research, there are 2 major variants of natural gradient that are frequently in use. The first one in named natural gradient for point estimation(NGPE) and the second one is natural gradient for variational inference(NGVI).

As the name suggests, NGPE tries to estimate just the predictive distribution $p(y|\mathbf{x}, \mathbf{w})$ by optimizing over a loss function. For this task, the FIM is constructed by calculating the
$\mathrm{F} = \mathbb{E}_{p(y|\mathbf{x},\mathbf{w})}[\nabla_\theta \log p(y|\mathbf{x}, \mathbf{w}) \nabla_\theta \log p(y|\mathbf{x}, \mathbf{w})^{\mathrm{T}}] \approx \mathrm{Cov}_{\mathbf{x}\sim p(\mathcal{D}), y\sim p(y|\mathbf{x},\mathbf{w})}[\nabla_{\mathbf{w}} \log p(y|\mathbf{x}, \mathbf{w})]$.

For NGVI, we now need to combine our background from Section 2.1 and 2.2. In variational Bayesian deep learning, we try to fit the parameters of a variational posterior $q_\theta(\mathbf{w})$ to maximize our variational objective ELBO((3)). Integrating natural gradient into this setting is simple. We need to compute the FIM of our variational posterior q instead of the predictive distribution, which is
$\mathrm{F} = \mathbb{E}_{q_\theta(\mathbf{w})}[\nabla_\theta \log q_\theta(\mathbf{w}) \nabla_\theta \log q_\theta(\mathbf{w})^{\mathrm{T}}] \approx \mathrm{Cov}_{\mathbf{w}\sim q_\theta(\mathbf{w})}[\nabla_{\mathbf{w}} \log q_\theta(\mathbf{w})]$.

3.1 Variational Inference using Noisy Natural Gradient

Zhang et al. (2018) show that we can approximate NGVI updates with a variaant of NGPE with adaptive noise which they called Noisy Natural Gradient(NNG). This insight is critical since it allow us to train variational posteriors with the noisy versions of the well-known optimizers.

Although we talked about the reparametrization trick for derivative of the Gaussian distribution, Zhang et al. (2018) built their work on the Gaussian gradient estimator of Opper and Archambeau (2009) as given below:

$$\begin{aligned} \nabla_{\boldsymbol{\mu}} \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})}[f(\mathbf{w})] &= \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})}\left[\nabla_{\mathbf{w}} f(\mathbf{w})\right] \\ \nabla_{\boldsymbol{\Sigma}} \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})}[f(\mathbf{w})] &= \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})}\left[\nabla^2_{\mathbf{w}} f(\mathbf{w})\right] \end{aligned} \tag{18}$$

Zhang et al. (2018) assume that the variational posterior q is a multivariate Gaussian parametrized by $\theta = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$. They build their proof on eqn. (18), and determine that the natural gradient of ELBO with respect to μ and the precision matrix $\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1}$:

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

Firstly, it is observed that the the term inside the expectation in gradient of ELBO with respect to $\boldsymbol{\mu}$ is MAP estimation of w. Secondly, the update for $\boldsymbol{\mu}$ is preconditioned by $\boldsymbol{\Sigma}^{-1}$, which implies a faster movement in the higher posterior uncertainty - known as the plateau regions -. The equation for the gradient with respect to $\boldsymbol{\Lambda}$ implies that we have a fixed value for $\boldsymbol{\Lambda}$ when we set the gradient to zero:

$$\Lambda = -\mathbb{E}_q\left[\nabla^2_{\mathbf{w}} \log p(\mathcal{D} \mid \mathbf{w}) + \nabla^2_{\mathbf{w}} \log p(\mathbf{w})\right] \tag{20}$$

It is trivial the see that for $\lambda = 1$, $\boldsymbol{\Lambda}$ corresponds to the expected Hessian of $-\log p(\mathbf{w}, \mathcal{D})$, which look appears to be like the Newton-Raphson update($x_{n+1} = x_n - \frac{f\prime(x_n)}{f''(x_n)}$ instead of $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$). For easiness, they assume a spherical Gaussian prior $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}/\eta)$, and then we have $\nabla^2_{\mathbf{w}} \log p(\mathbf{w}) = \eta \mathbf{I}$. In each iteration, we draw samples from $(\mathbf{x}, y) \sim p(\mathcal{D})$ and $\mathbf{w} \sim q_\theta(\mathbf{w})$, and apply the following natural gradient updates based on eqn. (19):

$$\begin{aligned} \mu &\leftarrow \mu + \alpha \Lambda^{-1}\left[\nabla_{\mathbf{w}} \log p(y|\mathbf{x}, \mathbf{w}) - \tfrac{\eta}{N}\mathbf{w}\right] \\ \boldsymbol{\Lambda} &\leftarrow \left(1 - \tfrac{\beta}{N}\right)\boldsymbol{\Lambda} - \beta\left[\nabla^2_{\mathbf{w}} \log p(y \mid \mathbf{x}, \mathbf{w}) - \tfrac{\eta}{N}\mathbf{I}\right] \end{aligned} \tag{21}$$

where α and β are separate learning rates for $\boldsymbol{\mu}$ and $\boldsymbol{\Lambda}$. Imprecisely, the update rule for $\boldsymbol{\Lambda}$ implies the exponential moving average of the Hessian and the update rule for $\boldsymbol{\mu}$ implies a stochastic Newton-Raphson step.

The update rule has two problems. The first one is that the Hessian might be hard to compute since deep neural networks use activation functions such as ReLU, which is not differentiable in every point. And the second problem is that the Hessian might have negative eigen-values if the negative log-likelihood is not convex, which means that our update for $\boldsymbol{\Lambda}$ might not be positive semi-definite. Zhang et al. (2018) proposed using the NGPE FIM $\mathrm{F} = \mathrm{Cov}_{\mathbf{x}\sim p(\mathcal{D}), y\sim p(y|\mathbf{x},\mathbf{w})}[\nabla_{\mathbf{w}} \log p(y|\mathbf{x},\mathbf{w})]$, which prevents both of the problems stated:

$$\Lambda \leftarrow \left(1 - \frac{\beta}{N}\right)\Lambda + \beta \left[\underbrace{\nabla_{\mathbf{w}} \log p(y|\mathbf{x},\mathbf{w}) \nabla_{\mathbf{w}} \log p(y|\mathbf{x},\mathbf{w})^{\mathrm{T}}}_{\mathbf{F}} + \frac{\eta}{N}\mathbf{I}\right] \tag{22}$$

This approximation of Hessian assures that $\boldsymbol{\Lambda}$ will be positive semi-definite and it will allow tractable approximations, which is used in K-FAC. This approximation of FIM is proposed by Graves (2011), which is known as Graves approximation.

With a fixed prior variance η, $\boldsymbol{\Lambda}$ will become a damped version of the moving average of the FIM and this approximation can be rewritten as

$$\begin{aligned} &\boldsymbol{\Lambda} = N\overline{\mathbf{F}} + \eta\mathbf{I} \\ &\overline{\mathbf{F}} \leftarrow (1-\tilde{\beta})\overline{\mathbf{F}} + \tilde{\beta}[\nabla_{\mathbf{w}} \log p(y|\mathbf{x},\mathbf{w}) \nabla_{\mathbf{w}} \log p(y|\mathbf{x},\mathbf{w})^{\mathrm{T}}] \end{aligned} \tag{23}$$

After defining alternative learning rates as $\tilde{\alpha} = \alpha/N$ and $\tilde{\beta} = \beta/N$, we also edit the update rule of μ as

$$\mu \leftarrow \mu + \tilde{\alpha}\left(\overline{\mathbf{F}} + \frac{\eta}{N}\mathbf{I}\right)^{-1}\left[\mathcal{D}\mathbf{w} - \frac{\eta}{N}\mathbf{w}\right] \tag{24}$$

Pay attention that the new update rule for μ resembles NGPE since it can be viewed as a natural gradient update with the damped version of the FIM. Even though we are doing NGPE, we can sample the weight from the variational posterior q, which is a normal distribution with $\mathbf{w} \sim \mathcal{N}\left(\boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1}\right)$, where $\Lambda^{-1} = (N\overline{\mathbf{F}} + \eta\mathbf{I})^{-1}$. With this update rules, one can make Bayesian inference with a noisy version of natural gradient method.

3.1.1 NOISY ADAM

As you might notice, until now we treat our covariance matrix Λ^{-1} as a full covariance Gaussian. This is unrealistic since the number of parameters needed for a full covariance matrix is $(dim(w))^2$, which is unrealistic for deep neural networks. Zhang et al. (2018) proposed a solution by approximating the FIM with a diagonal matrix $\mathbf{f}$. For their noisy natural gradient version of Adam, the updates for μ and $\mathbf{f}$ are

$$\begin{aligned} &\mu \leftarrow \mu + \tilde{\alpha}\left[\mathcal{D}\mathbf{w} - \tfrac{\eta}{N}\mathbf{w}\right] / \left(\overline{\mathbf{f}} + \tfrac{\eta}{N}\right) \\ &\overline{\mathbf{f}} \leftarrow (1-\tilde{\beta})\overline{\mathbf{f}} + \tilde{\beta}\mathcal{D}\mathbf{w}^2 \end{aligned} \tag{25}$$

3.1.2 KRONECKER-FACTORED APPROXIMATE CURVATURE

Calculating the inverse of the FIM is the major challenge while using the natural gradient method integrated to optimizers. Large deep neural networks have millions of parameters and calculating this inverse in naive way is computationaly expensive. K-FAC algorithm proposed by Martens and Grosse (2020) uses a Kronecker-factored approximation to the FIM to perform efficient approximate natural gradient updates. Let l be the lth layer of the neural network. Denote the activation function at layer l as $\mathbf{a}_l \in \mathbb{R}^{n_1}$,, the weight matrix at layer l as $\mathbf{W}_l \in \mathbb{R}^{n_1 \times n_2}$, and the output at layer l as $\mathbf{s}_l \in \mathbb{R}^{n_2}$,. Then, we have $\mathbf{s}_l = \mathbf{W}_l^T \mathbf{a}_l$. For easiness, the following notations are defined:

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

Accordingly, the gradient of weights at layer l is $\mathcal{D}\mathbf{W}_l = \mathbf{a}_l \mathbf{g}_l^T$. This gradient formulation help K-FAC to decouple the FIM $\mathbf{F}_l$ by approximating $\mathbf{a}_l$ and $\mathbf{g}_l$ as independent:

$$\begin{aligned} \mathbf{F}_l &= \mathbb{E}\left[\operatorname{vec}\{\mathcal{D}\mathbf{W}_l\} \operatorname{vec}\{\mathcal{D}\mathbf{W}_l\}^\top\right] = \mathbb{E}\left[\mathbf{g}_l\mathbf{g}_l^\top \otimes \mathbf{a}_l\mathbf{a}_l^\top\right] \\ &\approx \mathbb{E}\left[\mathbf{g}_l\mathbf{g}_l^\top\right] \otimes \mathbb{E}\left[\mathbf{a}_l\mathbf{a}_l^\top\right] = \mathbf{S}_l \otimes \mathbf{A}_l = \tilde{\mathbf{F}}_l \end{aligned} \tag{26}$$

Consequently, assuming that the layers are independent, the whole FIM $\tilde{\mathbf{F}}$ can be approximated as block diagonal consisting of layerwise FIMs $\tilde{\mathbf{F}}_l$. By this approximation, we circumvent the quadratic storage cost of the exact FIM by just storing the values for $\tilde{\mathbf{A}}_l$ and $\tilde{\mathbf{S}}_l$. Another critical advantage of this approximation is that we now have a tractable computation of the approximate natural gradient as below:

$$\begin{aligned} \tilde{\mathbf{F}}_l^{-1} \operatorname{vec}\{\nabla_{\mathbf{W}_l} h\} &= \mathbf{S}_l^{-1} \otimes \mathbf{A}_l^{-1} \operatorname{vec}\{\nabla_{\mathbf{W}_l} h\} \\ &= \operatorname{vec}\left[\mathbf{A}_l^{-1} \nabla_{\mathbf{W}_l} h \mathbf{S}_l^{-1}\right] \end{aligned} \tag{27}$$

With the approximation of inverse of FIM at eqn. (27), the calculation for the natural gradient becomes much more storage and computation efficient.

3.1.3 NOISY K-FAC WITH MATRIX VARIATE GAUSSIAN POSTERIOR

Matrix variate Gaussian distribution is a generalization of multivariate Gaussian distribution for matrix random variables, which take into account for both row-wise and column-wise correlations. It is more definitive than having a diagonal covariance matrix for a multivariate Gaussian distribution.

From section(3.1.2), we know that for each weight matrix in the network, we can calculate the inverse FIM with Kronecker-factored approximation as in eqn. (27). By plugging in this inverse FIM approximation to eqn. (23), we will have the MVG posterior. The update rule for $\overline{\mathbf{A}}_l$ and $\overline{\mathbf{S}}_l$ is as follows:

$$\begin{aligned} \overline{\mathbf{A}}_l &\leftarrow (1-\tilde{\beta})\overline{\mathbf{A}}_l + \tilde{\beta}\mathbf{a}_l\mathbf{a}_l^\top \\ \overline{\mathbf{S}}_l &\leftarrow (1-\tilde{\beta})\overline{\mathbf{S}}_l + \tilde{\beta}\mathcal{D}\mathbf{s}_l\mathcal{D}\mathbf{s}_l^\top \end{aligned} \tag{28}$$

Additionally, since $\overline{\mathbf{A}}_l$ and $\overline{\mathbf{S}}_l$ matrices estimated from empirical covariances, they are positive semi-definite. Since we still need to add the Hessian of the prior distribution, we still do not have a MVG posterior. However, since we set our prior as a spherical Gaussian, we can approximate the Σ using a damping trick proposed by Martens and Grosse (2020). With this trick, Σ_l decomposes as the Kronecker product of two terms:

$$\begin{aligned} \mathbf{\Sigma}_l &= \frac{1}{N}\left[\mathbf{S}_l^\gamma\right]^{-1} \otimes \left[\mathbf{A}_l^\gamma\right]^{-1} \\ &\triangleq \frac{1}{N}\left(\overline{\mathbf{S}}_l + \frac{1}{\pi_l}\sqrt{\frac{\eta}{N}}\mathbf{I}\right)^{-1} \otimes \left(\overline{\mathbf{A}}_l + \pi_l\sqrt{\frac{\eta}{N}}\mathbf{I}\right)^{-1} \end{aligned} \tag{29}$$

Using this factorization, we have MVG posterior$\mathcal{MN}\left(\mathbf{W}_l; \mathbf{M}_l, \frac{1}{N}\left[\mathbf{A}_l^{\gamma}\right]^{-1}, \left[\mathbf{S}_l^{\gamma}\right]^{-1}\right)$, where the factor $1/N$ is arbitrarily assigned to the first factor. Please see Algorithm 1 to see noisy K-FAC as pseudo-code.

Algorithm 1: Noisy K-FAC. Subscript l denotes layers, . We assume zero momentum for simplicity. Differences from standard K-FAC are shown in red

input : α: Stepsize
input : β: Exponential moving average parameter
input : η, γ_{ex},: prior variance, extrinsic damping term
input : 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 $\gamma_{\text{in}} = \frac{\eta}{N}$, total damping term $\gamma = \gamma_{\text{in}} + \gamma_{\text{ex}}$.
while *not converged* **do**
  $k \leftarrow k + 1$
  $\mathbf{W}_l \sim \mathcal{MN}\left(\mathbf{M}_l, \frac{1}{N}\left[\mathbf{A}_l^{\gamma_{\text{in}}}\right]^{-1}, \left[\mathbf{S}_l^{\gamma_{\text{in}}}\right]^{-1}\right)$
  if $k \equiv 0 \,(\text{mod} T_{stats})$ **then**
    Update the factors $\{\mathbf{S}_l\}_{l=1}^{L}, \{\mathbf{A}_l\}_{l=0}^{L-1}$ using eq. (12)
  if $k \equiv 0 \,(\text{mod} T_{inv})$ **then**
    Calculate the inverses $\left\{\left[\mathbf{S}_l^{\gamma}\right]^{-1}\right\}_{l=1}^{L}, \left\{\left[\mathbf{A}_l^{\gamma}\right]^{-1}\right\}_{l=0}^{L-1}$ using eq. (13).
  $\mathbf{V}_l = \nabla_{\mathbf{W}_l} \log p(y \mid \mathbf{x}, \mathbf{w}) - \gamma_{\text{in}} \cdot \mathbf{W}_l$
  $\mathbf{M}_l \leftarrow \mathbf{M}_l + \alpha \left[\mathbf{A}_l^{\gamma}\right]^{-1} \mathbf{V}_l \left[\mathbf{S}_l^{\gamma}\right]^{-1}$

3.2 Fast and Scalable Bayesian Deep Learning by Weight-Perturbation in Adam

The work proposed by Khan et al. (2018) is directly approximate the posterior distribution q instead of approximating it with a variant of NGPE methods. As stated in Section 3, NGVI methods calculate the natural gradient by scaling the gradient with FIM of the approximate posterior q. They select the prior distribution and the variational posterior from a Gaussian distribution with diagonal variances, where $p(\mathbf{w}) := \mathcal{N}(\mathbf{w}, \boldsymbol{\theta} \mid 0, \mathbf{I}/\eta)$, and $q_\theta(\mathbf{w}) := \mathcal{N}\left(\boldsymbol{w}, \boldsymbol{\theta} \mid \boldsymbol{\mu}, \text{diag}\left(\boldsymbol{\sigma}^2\right)\right)$. $\eta \in \mathbb{R}$ corresponds to the precision parameter with $\eta > 0$, and $\mu, \sigma \in \mathbb{R}^D$ correspond to mean and standard deviation of q. They build their work upon the natural gradient method of Salimbeni et al. (2018), where they propose the following update:

$$\begin{aligned} \mu_{t+1} &= \mu_t + \beta_t \sigma_{t+1}^2 \circ \left[\hat{\nabla}_\mu \mathcal{L}_t\right] \\ \sigma_{t+1}^{-2} &= \sigma_t^{-2} - 2\beta_t \left[\widehat{\nabla}_{\sigma^2} \mathcal{L}_t\right] \end{aligned} \tag{30}$$

where $\beta_t > 0$ and $a \circ b$ refers to the element-wise multiplication of a and b. Observe that that this update differs from eqn. (17) in one important aspect: the learning rate β_t in NGVI update is adapted by variance. This update requires a constraint on variance $\sigma^2 > 0$. But constraint is eliminated by using a FIM approximation as we will see in next sub-section.

3.2.1 VARIATIONAL ONLINE GAUSS-NEWTON (VOGN)

In their work, Khan et al. (2018) show that we can express the NGVI update in terms of the MLE objective. Let's denote the MLE objective and and minibatch stochastic-gradient estimates as

$$f(\boldsymbol{\theta}) := \frac{1}{N} \sum_{i=1}^{N} f_i(\boldsymbol{\theta}), \quad \hat{\mathrm{g}}(\boldsymbol{\theta}) := \frac{1}{M} \sum_{i \in \mathcal{M}} \nabla_\theta f_i(\boldsymbol{\theta}), \tag{31}$$

where $f_i(\boldsymbol{\theta}) := -\log p\left(\mathcal{D}_i \mid \boldsymbol{\theta}\right)$ corresponds to negative log-likelihood of i 'th data example, and the minibatch $\mathcal{M}$ contains M examples chosen uniformly at random. With the same way, we can approxi-

mate the Hessian which is denoted by $\widehat{\nabla}^2_{\theta\theta} f(\boldsymbol{\theta})$. They show the the NGVI update can be written in terms of stochastically approximated Hessian of f as

$$\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t - \beta_t \left(\hat{\mathrm{g}}\left(\boldsymbol{\theta}_t\right) + \tilde{\eta}\boldsymbol{\mu}_t\right) / \left(\mathbf{s}_{t+1} + \tilde{\eta}\right), \quad \mathbf{s}_{t+1} = \left(1-\beta_t\right)\mathbf{s}_t + \beta_t \operatorname{diag}\left[\widehat{\nabla}^2_{\theta\theta} f\left(\boldsymbol{\theta}_t\right)\right] \tag{32}$$

where $\mathbf{a}/\mathbf{b}$ denote the element-wise division, and the variational posterior q is approximated using one Monte-Carlo (MC) sample $\boldsymbol{\theta}_t \sim \mathcal{N}\left(\boldsymbol{\theta} \mid \boldsymbol{\mu}_t, \boldsymbol{\sigma}_t^2\right)$ with $\boldsymbol{\sigma}_t^2 := 1/\left[N\left(\mathbf{s}_t + \tilde{\eta}\right)\right]$ and $\tilde{\eta} := \eta/N$. This update resembles the online-Newton meghod, since s_t contains an online estimate of the diagonal of the Hessian. That's why this method is called "variational online-Newton" (VON).

Unfortunately, since the f is non-convex, we can still have a negative variance σ^2. Khan et al. (2018) overcome this issue by using the following approximation of the Hessian:

$$\nabla^2_{\theta_j\theta_j} f(\boldsymbol{\theta}) \approx \frac{1}{M} \sum_{i \in \mathcal{M}} \left[\nabla_{\theta_j} f_i(\theta)\right]^2 := \hat{h}_j(\boldsymbol{\theta}) \tag{33}$$

where θ_j is the j 'th element of θ. This approximation will always be non-negative, therefore if the initial σ^2 at $t = 1$ is positive, it will keep being positive in the next iterations as well. As a result, the new update rule for s_t becomes

$$\mathbf{s}_{t+1} = \left(1-\beta_t\right)\mathbf{s}_t + \beta_t \hat{h}_j(\boldsymbol{\theta}) \tag{34}$$

This the approximated Hessian for the update of s_t we have the Variational Online Gauss-Newton (VOGN). Unfortunately, the implementation is not trivial with the current existing deep learning frameworks since they do not support computation of individual gradients which is required for the Hessian approximation at eqn. (33).

Demonstration of VOGN method for deep learning is done by Osawa et al. (2019) recently, where they propose a distributed training scheme for VOGN method. Since VOGN updates are similar to Adam, they integrate the momentum term into the algorithm which increases the speed of convergence in deep learning.

They also parallelise the the computation of Monte-Carlo samples over multiple GPU, which increase the speed. After the calculation of different MC samples on different GPU, they averaged over the losses and gradients in single GPU and continues the rest of the algorithm on single GPU. The combination of two parallelism techniques with different MC samples for different inputs theoretically reduces the variance during the training. As it is already discussed in the previous paragraph, there is a need of extra computation due to the fact the current deep learning frameworks are not supporting the access of the gradient for individual samples. But since this is not a theoretical limitation, that can be implemented in the future. Please see the pseudocode in (5) for details:

3.2.2 VARIATIONAL RMSPROP(VPROP)

As a consequence of the previously mentioned problems of implementing VOGN, an approximate Hessian by gradient magnitude(GM) is proposed:

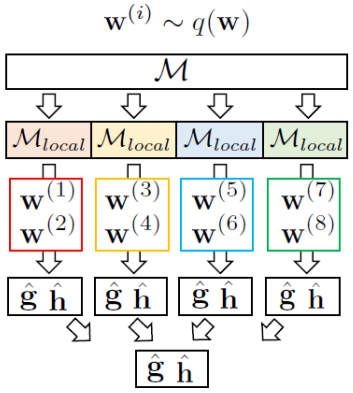
$$\nabla^2_{\theta_j\theta_j} f(\boldsymbol{\theta}) \approx \left[\frac{1}{M} \sum_{i \in \mathcal{M}} \nabla_{\theta_j} f_i(\theta)\right]^2 = \left[\hat{g}_j(\boldsymbol{\theta})\right]^2 \tag{35}$$

Instead of computing the sum of squared gradients as it is in GGN, this approximation computes the square of the sum of gradients, which is also used by the well-known optimizer RMSprop:

$$\theta_{t+1} = \theta_t - \alpha_t \hat{\mathrm{g}}\left(\theta_t\right) / \left(\sqrt{\bar{\mathrm{s}}_{t+1}} + \delta\right), \quad \bar{\mathrm{s}}_{t+1} = \left(1-\beta_t\right)\bar{\mathrm{s}}_t + \beta_t\left[\hat{\mathrm{g}}\left(\theta_t\right) \circ \hat{\mathrm{g}}\left(\theta_t\right)\right] \tag{36}$$

Algorithm 1: Variational Online Gauss Newton (VOGN)

Initialise $\boldsymbol{\mu}_0$, $\mathbf{s}_0$, $\mathbf{m}_0$.
$N \leftarrow \rho N$, $\tilde{\delta} \leftarrow \tau\delta/N$.
repeat
  Sample a minibatch $\mathcal{M}$ of size M.
  Split $\mathcal{M}$ into each GPU (local minibatch $\mathcal{M}_{local}$).
  for each GPU in parallel **do**
    for $k = 1, 2, \ldots, K$ **do**
      Sample $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
      $\mathbf{w}^{(k)} \leftarrow \boldsymbol{\mu} + \epsilon\boldsymbol{\sigma}$ with $\boldsymbol{\sigma} \leftarrow (1/(N(\mathbf{s} + \tilde{\delta} + \gamma)))^{1/2}$.
      Compute $\mathbf{g}_i^{(k)} \leftarrow \nabla_w \ell(\mathbf{y}_i, \mathbf{f}_{w^{(k)}}(\mathbf{x}_i)), \forall i \in \mathcal{M}_{local}$ using the method described in Appendix B.
      $\hat{\mathbf{g}}_k \leftarrow \frac{1}{M} \sum_{i \in \mathcal{M}_{local}} \mathbf{g}_i^{(k)}$.
      $\hat{\mathbf{h}}_k \leftarrow \frac{1}{M} \sum_{i \in \mathcal{M}_{local}} (\mathbf{g}_i^{(k)})^2$.
    end for
    $\hat{\mathbf{g}} \leftarrow \frac{1}{K} \sum_{k=1}^{K} \hat{\mathbf{g}}_k$ and $\hat{\mathbf{h}} \leftarrow \frac{1}{K} \sum_{k=1}^{K} \hat{\mathbf{h}}_k$.
  end for
  AllReduce $\hat{\mathbf{g}}$, $\hat{\mathbf{h}}$.
  $\mathbf{m} \leftarrow \beta_1 \mathbf{m} + (\hat{\mathbf{g}} + \tilde{\delta}\boldsymbol{\mu})$.
  $\mathbf{s} \leftarrow (1 - \tau\beta_2)\mathbf{s} + \beta_2 \hat{\mathbf{h}}$.
  $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - \alpha \mathbf{m}/(\mathbf{s} + \tilde{\delta} + \gamma)$.
until stopping criterion is met



Learning rate	α
Momentum rate	β_1
Exp. moving average rate	β_2
Prior precision	δ
External damping factor	γ
Tempering parameter	τ
# MC samples for training	K
Data augmentation factor	ρ

Figure 5: Table is prepared from the experimental results of (Khan et al., 2018; Zhang et al., 2018)

where $\bar{\mathrm{s}}_t$ is the vector that adapts the learning rate and, δ is a small scalar added to avoid dividing by zero. With this GM approximation of Hessian and minor modifications in the update equations, VON update look like very similar to RMSprop. Other that using the GM approximation, the only difference is the s_{t+1} term in eqn.(32) should be square rooted as below:

$$\mu_{t+1} = \mu_t - \alpha_t(\hat{\mathrm{g}}(\theta_t) + \tilde{\eta}\mu_t)/(\sqrt{\mathrm{s}_{t+1}} + \tilde{\eta}), \quad \mathrm{s}_{t+1} = (1 - \beta_t)\,\mathrm{s}_t + \beta_t\,[\hat{\mathrm{g}}\,(\theta_t) \circ \hat{\mathrm{g}}\,(\theta_t)] \tag{37}$$

where $\theta_t \sim \mathcal{N}\left(\theta \mid \mu_t, \sigma_t^2\right)$ with $\sigma_t^2 := 1/[N(\,\mathrm{s}_t + \tilde{\eta})]$. Red parts correspond to the differences from standart RMSprop. This sampling is done by reparametrization trick, mentioned in the Section 2.1.3. This reparametrization implies weight perturbation, where you add random noise with variance σ_t^2 to the mean values of the weights. With this method, we can perform variational inference.

The approximation performance of the GM method is also an important point. Obviously, GM approximation is not the best approximation for Hessian computation but Khan et al. (2018) provide a proof that the given a minibatch of size M, the expectation of the GM approximation is somewhere between the GGN and square of the full-batch gradient.

3.2.3 VARIATIONAL ADAM(VADAM)

Momentum methods generally have the form of Polyak's heavy ball method

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \bar{\alpha}_t \nabla_\theta f_1\,(\boldsymbol{\theta}_t) + \bar{\gamma}_t\,(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}) \tag{38}$$

where f_1 is the objective function we are maximizing and the last term is momentum. They propose natural-momentum version of eqn.(38) by replacing the Euclidean distance with KL divergence.
The variational posterior q is assumed to be an exponential-family distribution with natural parameter η and proposed the following natural-momentum method:

$$\eta_{t+1} = \eta_t + \bar{\alpha}_t \tilde{\nabla}_\eta \mathcal{L}_t + \bar{\gamma}_t\,(\eta_t - \eta_{t-1}) \tag{39}$$

where $\tilde{\nabla}$ it the natural-gradients with respect to natural parameter space and the gradient scaled by the Fisher information matrix of $q(\theta)$. They show that the eqn.(39) can be expressed as VON update with momentum:

$$\begin{aligned}\mu_{t+1} &= \mu_t - \bar{\alpha}_t \left[\frac{1}{\mathrm{s}_{t+1} + \tilde{\eta}}\right] \circ \left(\nabla_\theta f\left(\boldsymbol{\theta}_t\right) + \tilde{\eta}\boldsymbol{\mu}_t\right) + \bar{\gamma}_t \left[\frac{\mathbf{s}_t + \tilde{\eta}}{\mathbf{s}_{t+1} + \tilde{\eta}}\right] \circ \left(\boldsymbol{\mu}_t - \boldsymbol{\mu}_{t-1}\right) \\ \mathbf{s}_{t+1} &= \left(1 - \bar{\alpha}_t\right) \mathbf{s}_t + \bar{\alpha}_t \nabla^2_{\theta\theta} f\left(\boldsymbol{\theta}_t\right)\end{aligned} \tag{40}$$

where $\theta_t \sim \mathcal{N}\left(\theta \mid \mu_t, \sigma_t^2\right)$ with $\sigma_t^2 := 1/[N(\mathrm{s}_t + \tilde{\eta})]$. By rewriting these updates, VON update with momentum become pretty similar to standart adam updates:

$$\begin{aligned}\theta_{t+1} &= \theta_t - \tilde{\alpha}_t \left[\frac{1}{\sqrt{\hat{s}_{t+1}} + \delta}\right] \circ \nabla_\theta f\left(\theta_t\right) + \tilde{\gamma}_t \left[\frac{\sqrt{\hat{s}_t} + \delta}{\sqrt{\hat{s}_{t+1}} + \delta}\right] \circ \left(\theta_t - \theta_{t-1}\right) \\ \hat{s}_{t+1} &= \gamma_2 \hat{s}_t + \left(1 - \gamma_2\right) \left[\hat{g}\left(\theta_t\right)\right]^2\end{aligned} \tag{41}$$

where $\tilde{\alpha}_t, \tilde{\gamma}_t$ are appropriately defined in terms of the Adam's learning rate α and γ_1 : $\tilde{\alpha}_t := \alpha\left(1 - \gamma_1\right) / \left(1 - \gamma_1^t\right)$ and $\tilde{\gamma}_t := \gamma_1 \left(1 - \gamma_1^{t-1}\right) \left(1 - \gamma_1^t\right)$. Red parts correspond to the differences from standart Adam(see Algorithm 3 for the details).

Algorithm 2: Noisy Adam. Differences from standard Adam are shown in red.

input : α: Stepsize
input : β_1, β_2: Exponential decay rates for updating μ and $\mathbf{f}$
input : , η, γ_{ex},: prior variance, extrinsic damping term

$\mathbf{m} \leftarrow 0$;
Calculate the intrinsic damping term $\gamma_{\text{in}} = \frac{\eta}{N}$, total damping term $\gamma = \gamma_{\text{in}} + \gamma_{\text{ex}}$.
while *not converged* **do**
  $\mathbf{w} \sim \mathcal{N}\left(\boldsymbol{\mu}, \frac{1}{N} \operatorname{diag}\left(\mathbf{f} + \gamma_{\text{in}}\right)^{-1}\right)$
  $\mathrm{g} \leftarrow \nabla_{\mathbf{w}} \log p(\mathcal{D}|w)$
  $\mathbf{m} \leftarrow \beta_1 \cdot \mathbf{m} + (1 - \beta_1) \cdot (\mathbf{g} + \gamma_{\text{in}} \cdot \mathbf{w})$
  $\mathbf{f} \leftarrow \beta_2 \cdot \mathbf{f} + (1 - \beta_2) \cdot (\mathbf{g} \circ \mathbf{g})$.
  (Update momentum)
  $\tilde{\mathbf{m}} \leftarrow \mathbf{m} / \left(1 - \beta_1^k\right)$
  $\hat{\mathbf{m}} \leftarrow \tilde{\mathbf{m}} / (\mathbf{f} + \gamma)$
  $\mu \leftarrow \mu + \alpha \cdot \hat{\mathrm{m}}$ (Update parameters)

Algorithm 3: Vadam. Differences from standard Adam are shown in red

input : α: Stepsize
input : β_1, β_2: Exponential decay rates for updating μ and $\mathbf{f}$
input : η:prior variance

$\mathbf{m} \leftarrow 0$;
while *not converged* **do**
  $\boldsymbol{w} \leftarrow \boldsymbol{\mu} + \sigma \circ \epsilon$, where
  $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}), \boldsymbol{\sigma} \leftarrow 1/\sqrt{N\mathbf{s} + \eta}$
  $\mathrm{g} \leftarrow -\nabla \log p\left(\mathcal{D}_i \mid \boldsymbol{\theta}\right)$
  $\mathbf{m} \leftarrow \beta_1 \mathbf{m} + (1 - \beta_1)\left(\mathbf{g} + \mu \frac{\eta}{N}\right)$
  $\mathbf{s} \leftarrow \beta_2 \mathbf{s} + (1 - \beta_2)\left(\mathbf{g} \circ \mathbf{g}\right)$
  $\hat{\mathbf{m}} \leftarrow \mathbf{m} / \left(1 - \beta_1^t\right), \quad \hat{\mathbf{s}} \leftarrow \mathbf{s} / \left(1 - \beta_2^t\right)$
  $\mu \leftarrow \mu - \alpha \hat{\mathbf{m}} / (\sqrt{\hat{\mathbf{s}}} + \frac{\eta}{N})$

4. Comparison of Noisy/Perturbed Optimizers

Noisy Adam(2) and Vadam(3) methods are algorithmically very similar to each other and they are both easy to implement. The probabilistic modelling is also quite similar since they both use spherical Gaussian distribution for prior and Gaussian distribution with diagonal covariance for approximate posterior. Two optimizer differentiate from each other with their gradient estimations of Gaussian. Noisy Adam uses Oper Estimator while Vadam is using reparametrization trick. Another difference of these optimizers is that the derivation of Vadam takes the natural-momentum estimation into the account from Polyak's heavy ball method, which noisy Adam doesn't provide. In practice, those two methods are shown to perform very similar, with slightly better results on noisy Adam.

The distributional version of VOGN proposed by Osawa et al. (2019) also employs momentum term, which has also a Adam like update. The only difference is that the algorithm is implemented in distributed manner and more critically, the Hessian approximation used for this method is Generalized Gauss Newton(GGN) approximation, which is a better approximation than the Gradient Magnitude(GM)

	Noisy Adam	**Vadam**	**Noisy K-FAC**	**VOGN**
Prior	Spherical Gaussian	Spherical Gaussian	Spherical Gaussian	Spherical Gaussian
Posterior	Gaussian with diagonal covariance	Gaussian with diagonal covariance	Gaussian with diagonal covariance	Gaussian with diagonal covariance
Hessian Approximation	Gradient Magnitude(GM)	Gradient Magnitude(GM)	Gradient Magnitude(GM)	Generalized Gauss Newton(GGN)
Gradient Estimators of Gaussian	Opper Estimator	Reparametrization Trick	Opper Estimator	Reparametrization Trick

Table 1: Difference and similarities between the noisy optimizer summarized.

approximation used in Vadam, Noisy Adam and Noisy K-FAC. As it is stated in Section (3.2.1), the downside of this approximation(eqn. 33 and tenth line in Figure 5) is that there is a need of accessing the gradients of individual samples which is not supported with the current deep learning frameworks and this slows down the update.

Different from the other methods, the approximate posterior choice for Noisy K-FAC is a matrix-variate Gaussian(MVG) distribution. That give Noisy K-FAC the ability to capture the correlations between weights since MVG has two covariance matrices. As an example, if we have a weight matrix $\mathbf{W} \in \mathbb{R}^{nxd}$, the number of parameters to represent MVG covariance(symmetric positive semi-definite $\mathbf{A} \in \mathbb{R}^{nxn}$ and symmetric positive semi-definite $\mathbf{S} \in \mathbb{R}^{dxd}$) would be $\frac{n^2+d^2}{2}$. For a multivariate Gaussian with diagonal covariance matrix, it will be nd and for a multivariate Gaussian with full covariance matrix, it would be $\frac{n^2d^2}{2}$. As a result, MVG has a more compact representation than a diagonal covariance matrix. Please see Table 1 to see the differences and similarities between the noisy optimizers in tabularized fashion.

5. Evaluation

5.1 Noisty Optimizers in Regression Tasks

For regression tasks, the optimizers are experimented with the datasets from UCI collection. The experimental setup done by Zhang et al. (2018) and Khan et al. (2018) is not exactly the same but similar. They both used a neural network with one hidden layer and 50 hidden units(Zhang et al. (2018) used 100 units for 2 largest datasets) with ReLU activation function.

Noisy Adam(3.1.1) and Vadam(3.2.3) are shown to perform very similar, with slightly better results on noisy Adam. Both have also compared to other popular Bayesian deep learning methods such as black box variational(BBVI) inference and MC-Dropout. MC-Dropout outperforms both versions of Adam. The convergence rate is another topic which is discussed by (Zhang et al., 2018; Khan et al., 2018), the plots in Figure 6 clearly shows that VOGN and Vadam converges mush faster that BBVI.

On the other hand, Noisy K-FAC have comparable performance with MC-Dropout, with a slightly

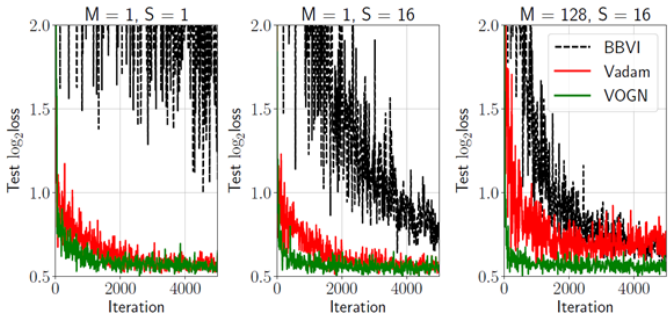


Figure 6: Table is prepared from the experimental results of (Khan et al., 2018; Zhang et al., 2018)

	Test log-likelihood				
Dataset	**MC-Dropout**	**BBVI**	**Vadam**	**Noisy Adam**	**Noisy K-FAC**
Boston	-2.46 +/- 0.06	-2.73 +/- 0.05	-2.85 +/- 0.07	-2.558 +/- 0.032	**-2.417 +/- 0.029**
Concrete	-3.04 +/- 0.02	-3.24 +/- 0.02	-3.39 +/- 0.02	-3.145 +/- 0.023	**-3.039 +/- 0.025**
Energy	-1.99 +/- 0.02	-2.47 +/- 0.02	-2.15 +/- 0.07	-1.629 +/- 0.020	**-1.421 +/- 0.005**
Kin8nmm	0.95 +/- 0.01	0.95 +/- 0.01	0.76 +/- 0.00	1.112 +/- 0.008	**1.148 +/- 0.007**
Naval	3.80 +/- 0.01	4.46 +/- 0.03	4.72 +/- 0.22	6.231 +/- 0.041	**7.079 +/- 0.034**
Power	-2.80 +/- 0.01	-2.88 +/- 0.01	-2.88 +/- 0.01	-2.803 +/- 0.010	**-2.776 +/- 0.011**
Wine	**-0.93 +/- 0.01**	1.00 +/- 0.001	-1.00 +/- 0.01	-0.976 +/- 0.016	-0.969 +/- 0.014
Yacht	**-1.55 +/- 0.03**	-2.41 +/- 0.02	-1.70 +/- 0.03	-2.412 +/- 0.006	-2.316 +/- 0.006

Table 2: Comparison of Noisy Adam, Noisy K-FAC and Vadam with other popular methods. Table is prepared from the experimental results of (Khan et al., 2018; Zhang et al., 2018)

advantage on Noisy K-FAC. It makes sense that noisy K-FAC performs better than Noisy Adam and Vadam due to its more expressive covariance for the approximate posterior. Please see the Table 2 for details on the experiment results.

5.2 Noisy Optimizers in Classification Tasks

The distributed version of momentum employed VOGN and Noisy K-FAC is experimented in classification tasks by Osawa et al. (2019). For the experiments they use the CIFAR-10 and ImageNet datasets, and famous neural network architectures such as LeNet, AlexNet and ResNet.

It is observed that based on NLL, ECE and AUROC metrics on 5 different dataset/architecture combinations(15 in total), VOGN performs the best or tied best on 10 metrics, and is second-best on the other 5(see Table 3). Plots comparing the validation accuracy of the VOGN with MC-dropout and standard Adam can be seen in Figure 7. VOGN also have a similar performance with these methods in terms of validation accuracy. The only experimental setup that involves Noisy K-FAC is the ImageNet/ResNet-18 setup in which Noisy K-FAC has comparable performance with Adam, and MC-dropout.

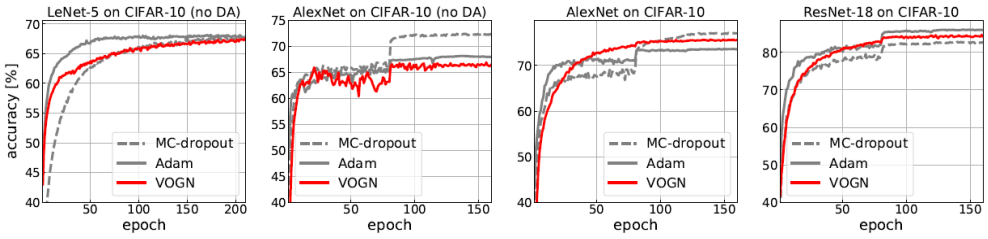


Figure 7: Figure taken from Osawa et al. (2019)

6. Conclusion

The work done by (Zhang et al., 2018; Khan et al., 2018; Osawa et al., 2019) has shown that, noisy optimizers which employs natural gradient method are good and efficient ways of performing variational inference in Bayesian deep learning.

In the experiments, it is observed that noisy optimizers employed by natural gradient method perform similar and sometimes better than the other Bayesian methods such as MC-Dropout and black box variational inference. Except VOGN, they are easy to implement with current deep learning frameworks. These noisy optimizers are also experimented and performed well in the active learning and reinforcement learning tasks but they are not mentioned in this summary paper due to page limit.

Dataset/ Architecture	Optimiser	Train/Validation Accuracy (%)	Validation NLL	Epochs	Time/epoch (s)	ECE	AUROC
CIFAR-10/ LeNet-5(no DA)	Adam	71.98 / **67.67**	**0.937**	210	6.96	**0.021**	0.794
	BBB	66.84 / 64.61	1.018	800	11.43	0.045	0.784
	MC-dropout	68.41 / **67.65**	0.99	210	6.95	0.087	**0.797**
	VOGN	70.79 / **67.32**	**0.938**	210	18.33	0.046	**0.8**
CIFAR-10/AlexNet(no DA)	Adam	100.0 / 67.94	2.83	161	3.12	0.262	0.793
	MC-dropout	97.56 / **72.20**	1.077	160	3.25	0.14	**0.818**
	VOGN	79.07 / 69.03	**0.93**	160	9.98	**0.024**	0.796
CIFAR-10/ AlexNet	Adam	97.92 / 73.59	1.48	161	3.08	0.262	0.793
	MC-dropout	80.65 / **77.04**	**0.667**	160	3.2	0.114	0.828
	VOGN	81.15 / 75.48	0.703	160	10.02	**0.016**	**0.832**
CIFAR-10/ ResNet-18	Adam	97.74 / **86.00**	0.55	160	11.97	0.082	**0.877**
	MC-dropout	88.23 / 82.85	0.51	161	12.51	0.166	0.768
	VOGN	91.62 / 84.27	**0.477**	161	53.14	**0.04**	0.876
ImageNet/ ResNet-18	SGD	82.63 / **67.79**	**1.38**	90	44.13	0.067	0.856
	Adam	80.96 / 66.39	1.44	90	44.4	0.064	0.855
	MC-dropout	72.96 / 65.64	1.43	90	45.86	**0.012**	0.856
	OGN	85.33 / 65.76	1.6	90	63.13	0.128	0.854
	VOGN	73.87 / **67.38**	**1.37**	90	76.04	0.029	0.854
	K-FAC	83.73 / 66.58	1.493	60	133.69	0.158	0.842
	Noisy K-FAC	72.28 / 66.44	1.44	60	179.27	0.08	0.852

Table 3: Performance comparisons of VOGN and Noisy K-FAC on different dataset/architecture combinations. DA means 'Data Augmentation', NLL refers to 'Negative Log Likelihood' (lower is better) and ECE refers to 'Expected Calibration Error'. (lower is better), AUROC refers to 'Area Under ROC curve' (higher is better).

References

Shun-ichi Amari. Neural learning in structured parameter spaces - natural riemannian gradient. In M. C. Mozer, M. Jordan, and T. Petsche, editors, Advances in Neural Information Processing Systems*, volume 9, pages 127–133. MIT Press, 1997. URL* `https://proceedings.neurips.cc/paper/1996/file/39e4973ba3321b80f37d9b55f63ed8b8-Paper.pdf`.

Shun-ichi Amari. Information Geometry and Its Applications*. Springer Publishing Company, Incorporated, 1st edition, 2016. ISBN 4431559779.*

Christopher M. Bishop. Pattern Recognition and Machine Learning*. Springer, 2006.*

Alex Graves. Practical variational inference for neural networks. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems*, volume 24, pages 2348–2356. Curran Associates, Inc., 2011. URL* `https://proceedings.neurips.cc/paper/2011/file/7eb3c8be3d411e8ebfab08eba5f49632-Paper.pdf`.

Mohammad Emtiyaz Khan, Didrik Nielsen, Voot Tangkaratt, Wu Lin, Yarin Gal, and Akash Srivastava. Fast and scalable bayesian deep learning by weight-perturbation in adam, 2018.

Mikael Kuusela, T. Raiko, Antti Honkela, and J. Karhunen. A gradient-based algorithm competitive with variational bayesian em for mixture of gaussians. 2009 International Joint Conference on Neural Networks*, pages 1688–1695, 2009.*

Shaowei Lin. What is singular learning theory? URL `https://shaoweilin.github.io/montreal.pdf`.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature, 2020.

Manfred Opper and Cédric Archambeau. The variational gaussian approximation revisited. Neural Comput.*, 21(3):786–792, March 2009. ISSN 0899-7667.*

Kazuki Osawa, Siddharth Swaroop, Anirudh Jain, Runa Eschenhagen, Richard E. Turner, Rio Yokota, and Mohammad Emtiyaz Khan. Practical deep learning with bayesian principles, 2019.

Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks, 2014.

Hugh Salimbeni, Stefanos Eleftheriadis, and James Hensman. Natural gradients in practice: Non-conjugate variational inference in gaussian process models, 2018.

Guodong Zhang, Shengyang Sun, David Duvenaud, and Roger Grosse. Noisy natural gradient as variational inference, 2018.