An estimator for the GLM predictive of a Kronecker-factored Laplace Bayesian neural network

Romie Banerjee

August 3, 2025

1 Introduction

Given pre-trained model, one can *bayesianify* it by Laplace Approximation. This additional structure makes the model probabilistic and lets produce predictive *distributions* (instead of point-set predictions). There are two steps of getting probabilistic predictions (aka. the predictive) from an input, by the LA route.

The **posterior** is approximated by a Gaussian using Laplace Approximation, which reduces to calculating the Hessian of the loss function of the network, i.e. the second order information of the loss landscape. The Hessian is further simplified by the Generalized-Gauss-Newton (GGN) approximation, this reduces the second order calculation to a product of first-order calculations. In practice this is realized as the empirical fisher of the model. There are various research directions focussing on effectively computing the empirical Fisher for many-parameter models. One such method, kronecker-factored-approximate-curvature (KFAC) uses layer-wise kronecker factorization of the empirical fisher.

The **predictive** admits no simple closed form formula since the feed-forward function is not explicit. So we estimate it, either directly by monte-carlo sampling from the posterior and then applying feed-forward, or by linearizing the feed-forward the map (linearized Laplace). The common estimator for the predictive is Monte-carlo-Integration (MCI), which is an estimator for the actual Laplace-LLM predictive

- **Monte Carlo Integration**: an estimator for the true Laplace-BNN predictive
- **Generalized Linear Model (GLM)**: a closed form solution to the *linearized* Laplace-BNN

We propose a monte-carlo estimator for the GLM predictive and a corresponding low-rank approximation.

2 Bayesian Neural Networks

A BNN consists of a (differentiable) map $f : X \times \Theta \to Y$, where the spaces X, Θ and Y and Y are respectively the input, model weights and output spaces. The spaces are isomorphic as affine spaces to $X \simeq \mathbb{R}^n$, $\Theta \simeq \mathbb{R}^w$ and $Y \simeq \mathbb{R}^d$. The mapping $(x, \theta) \mapsto f(x, \theta)$ expresses the evaluation of a model θ on the input x. (Note: In this formulation, the aleatoric predictive is suppressed by considering on point-valued functions f.). In this setup, a regular feed-forward network is a BNN with a fixed choice of weights, for example $\theta = \theta_*$. We can denote the resulting network as a function $f_{\theta_*} : X \to Y$, $f_{\theta_*}(x) = f(x, \theta_*)$. The weights space Θ is equipped with a probability measure, the posterior distribution (usually intractable) $P(\theta|D)$. The mode of the posterior,

$$\theta_{\mathrm{MAP}} = \mathrm{argmin}_{\theta \in W} \log P(D|\theta) + \log P(\theta) \tag{1}$$

is usually estimated using empirical risk minimization. Given.e..g., an i.i.d. classification data-set $D := (x_n, y_n) \in X \times Y_{n=1}^N$, the weights $\theta \in \Theta$ are trained to minimize the regularized empirical risk $\mathcal{L}(D; \theta) = \log P(D|\theta) + \log P(\theta)$.

The predictive distribution $P(y|x)$ on the output space Y is the push-forward probability measure of $P(\theta|D)$ under the map $f(x, -) : \Theta \to Y$.

$$P(y|x) = f(x, -)_* P(\theta|D) \tag{2}$$

The (bayesian) prediction $\hat{f}(x)$ is the mean of $P(y|x)$, i.e.

$$\hat{f}(x) = \mathbb{E}_{y \sim P(y|x)}[y] = \mathbb{E}_{\theta \sim P(\theta|D)}\left[f(x, \theta)\right] \tag{3}$$

The predictive uncertainty is the (co)variance of the predictive distribution $P(y|x)$, i.e.,

$$\mathrm{unc}(\hat{f}(x)) = \mathrm{cov}_{y \sim P(y|x)}[y] = \mathrm{cov}_{\theta \sim P(\theta|D)}\left[f(x, \theta)\right] \tag{4}$$

3 Monte Carlo Integration for BNNs

The Monte-Carlo estimate of predictive distribution is obtained by first sampling weights $\theta_1, \cdots, \theta_k \sim P(\theta|D)$. This produces samples from the predictive distribution $\{f(x, \theta_i)\}_{i=1}^k \sim P(y|x)$. Compute the sample mean to estimate the Bayesian prediction

$$\overline{\hat{f}(x)} = \frac{1}{k} \sum_{i=1}^{k} f(x, \theta_i) \in \mathbb{R}^d \tag{5}$$

Compute the sample covariance to estimate the predictive predictive

$$\overline{\mathrm{unc}(\hat{f}(x))} = \frac{1}{k-1} \sum_{i=1}^{k} f(x, \theta_i) f(x, \theta_i)^T \in \mathbb{R}^{d \times d} \tag{6}$$

One successful method following this principle is MCDropout [2, 1], where the posterior is estimated by variational inference on Bernoulli distributions. One disadvantage of the MC sampling method is that it is not post-hoc as the estimated $\overline{\mathrm{unc}(\hat{f}(x))}$ is an predictive for the Bayesian prediction $\overline{\hat{f}(x)}$ which in general does not equal to the original MAP prediction $f(x, \theta_{\mathrm{MAP}})$.

4 Laplace Approximation

([5]) Laplace approximation replaces the bayesian posterior with a normal distribution with θ_{MAP} as mean, $P(\theta|D) \sim_{LA} \mathcal{N}(\theta_{\mathrm{MAP}}, \Sigma)$, where the covariance $\Sigma \in \mathbb{R}^{w\times w}$ is the inverse of the Hessian

$$\Sigma := \left(\frac{\partial^2}{\partial \theta^2} \mathcal{L}(D;\theta)\bigg|_{\theta_{\mathrm{MAP}}} \right)^{-1} \tag{7}$$

This is second-order derivative calculation is often replaced in practice by the products of first-order derivatives , i.e. Generalized-Gauss-Newton approximation or equivalently the Fisher information matrix; the covariance of the gradients of the loss function at the empirical risk minimizer θ_{MAP}.

$$I_\theta(\theta_{\mathrm{MAP}}) = \mathrm{Cov}_{(x,y)\sim D} \left(\frac{\partial}{\partial \theta} \mathcal{L}((x,y);\theta)\bigg|_{\theta_{\mathrm{MAP}}} \right)$$

$$\boxed{\Sigma \sim I_\theta(\theta_{\mathrm{MAP}})^{-1}} \tag{8}$$

5 Generalized Linear Model for Laplace-BNNs

([3]) Define the generalized linear model by linearizing the BNN $f : X \times \Theta \to Y$ at $\theta = \theta_{\mathrm{MAP}}$ (via. Taylor series)

$$f_{\mathrm{lin}}(x,\theta) = f(x,\theta_{\mathrm{MAP}}) + \left[\frac{\partial}{\partial \theta} f(x,\theta)\bigg|_{\theta_{\mathrm{MAP}}} \right] (\theta - \theta_{\mathrm{MAP}}) \tag{9}$$

Denote the jacobian matrix $\frac{\partial}{\partial \theta} f(x,\theta)\big|_{\theta_{\mathrm{MAP}}}$ by $J(x) \in \mathbb{R}^{d\times w}$. Replace the posterior distribution $P(\theta|D)$ by the Laplace approximation $\mathcal{N}(\theta_{\mathrm{MAP}}, \Sigma)$. The GLM predictive distribution $P_{\mathrm{lin}}(y|x)$ on the output space Y is defined to be the push-forward measure of the Laplace posterior $N(\theta_{\mathrm{MAP}}, \Sigma)$ on the weight space Θ under the map $f_{\mathrm{lin}}(x,-) : \Theta \to Y$. Since $f_{\mathrm{lin}}(x,\theta)$ is affine on θ, the predictive distribution $P_{\mathrm{lin}}(y|x)$ is normal.

The mean and covariance of the normal distribution $P_{\mathrm{lin}}(y|x)$ are as follows

$$\begin{aligned}\mathbb{E}_{y\sim P_{\text{lin}}(y|x)}[y] &= \mathbb{E}_{\theta\sim\mathcal{N}(\theta_{\text{MAP}},\Sigma)}\left[f_{\text{lin}}(x,\theta)\right]\\ &= f(x,\theta_{\text{MAP}}) + J(x) * \mathbb{E}_{\theta\sim\mathcal{N}(\theta_{\text{MAP}},\Sigma)}\left[(\theta-\theta_{\text{MAP}})\right]\\ &= f(x,\theta_{\text{MAP}})\end{aligned}$$

$$\begin{aligned}\text{cov}_{y\sim P_{\text{lin}}(y|x)}[y] &= \text{cov}_{\theta\sim\mathcal{N}(\theta_{\text{MAP}},\Sigma)}\left[f_{\text{lin}}(x,\theta)\right]\\ &= \text{cov}_{\theta\sim\mathcal{N}(\theta_{\text{MAP}},\Sigma)}\left[J(x) * (\theta-\theta_{\text{MAP}})\right]\\ &= \text{cov}_{\eta\sim\mathcal{N}(0,\Sigma)}\left[J(x) * \eta\right]\\ &= J(x) * \Sigma * J(x)^T\end{aligned}$$

This property makes GLM suitable for post-hoc predictive estimation as the Bayesian prediction $\hat{f}_{\text{lin}}(x)$ which is simply the mean of the predictive distribution, agrees with the original prediction from the model θ_{MAP}.

$$\hat{f}_{\text{lin}}(x) = f(x,\theta_{\text{MAP}}) \tag{10}$$

The predictive at this point further has a simple analytical formula,

$$\boxed{\text{unc}(\hat{f}_{\text{lin}}(x)) = J(x) * \Sigma * J(x)^T} \tag{11}$$

6 Laplace BNNs with KFAC posterior

([6, 7]) The FI matrix $I = I_\theta(\theta_{\text{MAP}})$ has a very large size $w \times w$. For practical applications various approximations of I are available. The KFAC method employs two levels of approximation

1. Treating each layer on the neural network separately, ignoring cross-layer terms, expressing $I \in \mathbb{R}^{w\times w}$ in block diagonal form with diagonal blocks $I_{(l)} \in \mathbb{R}^{w_l \times w_l}$, where w_l is the size of the l-th layer and, $\sum_{l=1}^{L} w_l = w$. This reduces the size of I from w^2 to $\sum w_l^2$.

2. For each layer, the gradients ∇_l in this layer, is expressible as a Kronecker product $\nabla_l = a_{l-1} \otimes g_l$, where a_{l-1} is the incoming activation from layer $l-1$ and g_l is the otgoing gradient of layer l. The Fisher information block for layer l is then $I_{(l)} = \mathbb{E}\left[\nabla_l * \nabla_l^T\right] = \mathbb{E}\left[(a_{l-1}\otimes g_l) * (a_{l-1}\otimes g_l)^T\right]$. The KFAC method makes the approximation $I_{(l)} \approx \mathbb{E}\left[a_{l-1}\otimes a_{l-1}^T\right] \otimes \mathbb{E}\left[g_l \otimes g_l^T\right] =: Q_{(l)} \otimes H_{(l)}$. Here $Q_{(l)} \in \mathbb{R}^{l_{\text{in}}\times l_{\text{in}}}$ and $H_{(l)} \in \mathbb{R}^{l_{\text{out}}\times l_{\text{out}}}$ and the l-th layer is a map $\mathbb{R}^{l_{\text{in}}} \to \mathbb{R}^{l_{\text{out}}}$. This reduces the size from $\sum w_l^2 = \sum (l_{\text{in}} l_{\text{out}})^2$ to $\sum (l_{\text{in}}^2 + l_{\text{out}}^2)$.

3. The kronecker factored block-diagonal form $I_{\text{KFAC}} = \text{diag}\left[Q_{(1)}\otimes H_{(1)}, \cdots, Q_{(L)}\otimes H_{(L)}\right]$.

4. The KFAC posterior covariance, Σ_{KFAC} has the form

$$\begin{pmatrix} Q_{(1)}^{-1} \otimes H_{(1)}^{-1} & & \\ & \ddots & \\ & & Q_{(L)}^{-1} \otimes H_{(L)}^{-1} \end{pmatrix} \tag{12}$$

7 The GLM predictive for Laplace-KFAC BNNs

In order to use this during inference one needs access two matrices,

- the covariance matrix Σ or the inverse of the Fisher information matrix $I_\theta(\theta_{\text{MAP}})$ which is computed only once offline and used repeatedly during inference (independent of the input x)
- the jacobian $J(x)$, which must be computed for every input x.

Using the given co-ordinatization $\theta_1, \ldots, \theta_w$ of Θ, the entries of the jacobian matrix $J(x)$ are (13), where $\frac{\partial f_i}{\partial \theta_j}$ is short for $\frac{\partial}{\partial \theta_j} f_i(x, \theta)\Big|_{\theta=\theta_{\text{MAP}}}$, and $(f_1, \ldots, f_d)$ are the scalar components of f. The rows are the gradients of the scalar valued functions $f_i, 1 \leq i \leq d$.

$$J(x) = \begin{pmatrix} - & [\nabla f_1]^T & - \\ & \vdots & \\ - & [\nabla f_d]^T & - \end{pmatrix} \tag{13}$$

During inference time these rows can be computed using back-propagation. This would require d back-propagation steps, one for each scalar value of f.

When the covariance is expressed in Kronecker factored form the GLM uncertianty expression must be

$$\text{unc}(\hat{f}_{\text{lin}}(x)) = J(x)_{\text{KF}} * \Sigma_{\text{KFAC}} * J(x)_{\text{KF}}^T \tag{14}$$

where $J(x)_{\text{KF}}$ is the jacobian in a kronecker-factored form.

When $d = 1$, i.e. $f(x, \theta) \to \mathbb{R}$ is scalar valued, consider the jacobian $\frac{\partial}{\partial \theta} f(x, \theta)\big|_{\theta_{\text{MAP}}} \in \mathbb{R}^{1 \times w}$. For every layer $l \in [1, \cdots, L]$, the layerwise derivative has a kronecker decomposition $\frac{\partial}{\partial \theta_l} f(x, \theta)\Big|_{\theta_{\text{MAP}}} = a_{l-1} \otimes g_l$, where $a_{l-1} \in \mathbb{R}^{1 \times l_{\text{in}}}$ is the incoming activation and $g_l \in \mathbb{R}^{1 \times l_{\text{out}}}$ the outgoing gradient for layer l. This gives a decomposition of the jacobian into blocks.

$$
\begin{aligned}
J(x) &= \left(\frac{\partial}{\partial \theta_1} f(x,\theta)\bigg|_{\theta_{\mathrm{MAP}}}, \cdots, \frac{\partial}{\partial \theta_L} f(x,\theta)\bigg|_{\theta_{\mathrm{MAP}}} \right) \\
&\in \mathbb{R}^{d \times (w_1 + \cdots + w_L)} \\
&= \begin{pmatrix} | & & | \\ a_0^j \otimes g_1^j & \cdots & a_{L-1}^j \otimes g_L^j \\ | & & | \end{pmatrix}
\end{aligned}
$$

where the superscript $j \in (1, \cdots, d)$. Finally the GLM-predictive calculation,

$$
\begin{aligned}
&J(x) * \begin{pmatrix} Q_{(1)}^{-1} \otimes H_{(1)}^{-1} & & \\ & \ddots & \\ & & Q_{(L)}^{-1} \otimes H_{(L)}^{-1} \end{pmatrix} * J(x)^T \\
&= \mathrm{Diag}_{j=1}^{d} \left[\sum_{l=1}^{L} \left(a_{l-1}^j * Q_{(l)}^{-1} * {a_{l-1}^j}^T \right) \otimes \left(g_l^j * H_{(l)}^{-1} * {g_{l-1}^j}^T \right) \right] \qquad (15)
\end{aligned}
$$

8 MC-GLM: monte-carlo estimation of the GLM predictive

Given an input x, a pre-trained network θ_* and the network function $f_{\theta_*}(-)$, the Laplace approximation method and GLM inference estimates the predictive distribution by

$$P(y|x) = \mathcal{N}(f_{\theta_*}(x), J(x) * \Sigma * J(x)^T)$$

where Σ is the covariance of the Laplace approximation normal distribution of the bayesian posterior and $J(x)$ is the jacobian of the function $f_\theta(x)$ w.r.t. θ at θ_*. While the covariance Σ is computed offline, the jacobian $J(x)$ must be computed online. This is expensive when f_θ takes values in many dimensions. Monte-carlo sampling from the Bayesian posterior replaces the jacobian $J(x)$ by a low-rank and base-changed version $A(x)$ and the predictive covariance is estimated as

$$A(x) * A(x)^T \approx J(x) * \Sigma * J(x)^T$$

Unlike $J(x)$, which is calculated row-wise by calling back-prop for every scalar value of $f_\theta(x)$, the calculation of $A(x)$ is done columnwise by computing approximated directional derivatives along the directions comimg from the monte-carlo weight samples. This makes the MC-GLM method have limited time requirements, based on the numer of monte-carlo sampling steps.

The GLM uncterianty term 11 can be estimated by computing sample covariance,

$$\begin{aligned} J(x) * \Sigma * J(x)^T &= \mathrm{cov}_{\eta \in \mathcal{N}(0,\Sigma)} \left[J(x) * \eta \right] \\ &\approx \frac{1}{N-1} \sum_{i=1}^{N} \left[J(x) * \eta_i \right] \left[J(x) * \eta_i \right]^T \end{aligned}$$

where $(\eta_1, \cdots, \eta_n) \sim \mathcal{N}(0, \Sigma)$ are i.i.d samples. The noise vectors η can be interpreted as members of the tangent space $T_{\theta_{\mathrm{MAP}}}(\Theta)$, and the terms in the summand, $J(x) * \eta$ are simply the directional derivatives $D_{\vec{\eta}} f = \lim_{h \to 0} \left(f(\theta_{\mathrm{MAP}} + h\vec{\eta}) - f(\theta_{\mathrm{MAP}}) \right) / h$.

The MC-GLM method to estimate predictive is the following:

1. Sample $(\vec{\eta}_1, \ldots, \vec{\eta}_k) \sim \mathcal{N}(0, \Sigma)^{\times k}$, $k \ll w$. Define,
2. $$\overline{A}(x) := \begin{pmatrix} | & & | \\ \bar{D}_{\vec{\eta}_1} f & \cdots & \bar{D}_{\vec{\eta}_k} f \\ | & & | \end{pmatrix} \in \mathbb{R}^{d \times k}$$
3. The approximate directional derivative $\bar{D}$ is calculated using finite differences.
4. mcglm-unc$(f(x, \theta_{\mathrm{MAP}})) := \overline{A}(x) * \overline{A}(x)^T$

The Cholesky decomposition $\Sigma = B * B^T$ lets us write the predictive $J(x) * \Sigma * J(x)^T$ as $A(x) * A(x)^T$, where $A(x) = J(x) * B$. The matrix $A(x)$ is simply the jacobian $J(x)$ base-changed to the a new basis formed out of the columns of B. From this point of view the idea of MC-GLM is to base-change $J(x)$ using a different and low-rank basis for $T_{\theta_{\mathrm{MAP}}}(W)$, obtained by sampling.

9 Sampling from the KFAC posterior

In order to generate a sample from a n-variable normal $\vec{z} \sim \mathcal{N}(\vec{\mu}, \Sigma)$ one has to write it as $\vec{z} = \vec{\mu} + B\vec{x}$, where B is a $n \times n$ matrix such that $BB^T = \Sigma$ and $\vec{x} \sim \mathcal{N}(\vec{0}, I_n)$. The matrix B can be found using Cholesky decomposition and is guaranteed when Σ is a covariance. (Notation: $B = \mathrm{ch}(\Sigma)$.)

When the covariance is in KFAC form as in 12, we observe that the cholesky decomposition commutes with block-diagonal, inverse and kronecker product. Consequenty the Cholesky is computed as a block diagonal matrix with the blocks $\mathrm{ch}(Q_{(l)})^{-1} \otimes \mathrm{ch}(H_{(l)})^{-1}$ along the diagonal. For every layer l, let $\mathcal{N}(\vec{\mu}_l, Q_{(l)}^{-1} \otimes H_{(l)}^{-1})$ be the restriction of $\mathcal{N}(\vec{\mu}, \Sigma_{\mathrm{KFAC}})$ restricted to weights of f_{θ_l}.The layerwise sample obtained as

$$\begin{aligned} \vec{z}_l &= \left(\mathrm{ch}(Q_{(l)})^{-1} \otimes \mathrm{ch}(H_{(l)})^{-1} \right) * \vec{x}_l \\ &= \mathrm{ch}(Q_{(l)})^{-1} * \vec{x}_l.\mathrm{reshape}(l_{\mathrm{in}}, l_{\mathrm{out}}) * \mathrm{ch}(H_{(l)})^{-1} \end{aligned}$$

where $\vec{x}_l \sim \mathcal{N}(\vec{\mu}_l, I_{w_l})$.

10 Algorithms

Algorithm 1 KFAC estimation: Offline calculation of Σ_{KFAC} through gradients (and their kroneker factorizations) of the loss function at the MAP weights, iterated on the training data

```
$f_\theta : X \to Y \leftarrow$ nueral network model
$f_{\theta_1}, \cdots, f_{\theta_L} \leftarrow$ layers of $f_\theta$
$D \leftarrow$ training dataset$\{(x_i, y_i) | 1 \leq i \leq N\}$
$\mathcal{L}(x_i, y_i) \leftarrow$ loss function
$\theta_{\text{MAP}} \leftarrow$ ERM w.r.t $\mathcal{L}$ on $D$.
procedure KFAC($f_\theta, \theta_{\text{MAP}}, D$)        ▷ The KFAC Fisher estimation for a pre-trained model
    $I_{\text{KFAC}} \leftarrow$ dict()        ▷ dictionary to store KFAC-fisher blocks
    for $l \leftarrow [1, \cdots, L]$ do
        $I_{\text{KFAC}}[l] = [0_{l_{\text{in}} \times l_{\text{in}}}, 0_{l_{\text{out}} \times l_{\text{out}}}]$
    end for
    for $(x_i, y_i) \in D$ do        ▷ run through training data
        for $l \in$ range($L$) do        ▷ $L =$ no. of layer
            $v_l \leftarrow \frac{\partial}{\partial \theta_l} \mathcal{L}(x_i, y_i)\Big|_{\theta_{\text{MAP}}}$        ▷ backprop loss to get gradients of layer $l$
            $a_{l-1} \leftarrow$ input activation to $l$-th layer
            $g_l \leftarrow$ output gradient of $l$-th layer
            vec($v_l$) $= a_{l-1} \otimes g_l$        ▷ kronecker decomposition of gradient
            $Q_{(l)} \leftarrow a_{l-1} * a_{l-1}^T$        ▷ outer product
            $H_{(l)} \leftarrow g_l * g_l^T$        ▷ outer product
            $\Sigma_{\text{KFAC}}[l][0] + = Q_{(l)}$
            $\Sigma_{\text{KFAC}}[l][1] + = H_{(l)}$
            return $I_{\text{KFAC}}$
        end for
    end for
end procedure
procedure Invert($I_{\text{KFAC}}$)        ▷ Invert the KFAC Fisher
    $\Sigma_{\text{KFAC}} \leftarrow$ dict()
    for $l \in$ range($L$) do
        $\Sigma_{\text{KFAC}}[l] \leftarrow [Q_{(l)}^{-1}, H_{(l)}^{-1}]$
    end for
    return $\Sigma_{\text{KFAC}}$
end procedurereturn $\Sigma_{\text{KFAC}}$
```

Algorithm 2 GLM predictive: (1) Online computation of the Jacobian $\frac{\partial}{\partial\theta}f(x,\theta)\big|_{\theta_{\text{MAP}}}$ in KF form $J_{\text{KF}}(x)$, and (2) GLM predictive computation $J_{\text{KF}}(x) * \Sigma_{KFAC} * J_{\text{KF}}(x)$

```
$x \leftarrow$ input
$f_{\theta_{\text{MAP}}} \leftarrow$ MAP model
$f = (f_1, \cdots, f_d)$                                        ▷ Scalar values of $f$
**procedure** $J_{\text{KF}}(x, \theta_{\text{MAP}})$   ▷ Compute jacobian $\frac{\partial}{\partial\theta}f(x,\theta)\big|_{\theta_{\text{MAP}}}$ and store in KF format
    $J \leftarrow$ dict()
    **for** $j \in$ range($d$) **do**                                   ▷ $d = \dim(Y)$
        **for** $l \in$ range($L$) **do**                               ▷ $l$ = no.of layers
            $a^j_{l-1} \otimes g^j_l \leftarrow \frac{\partial}{\partial\theta_l} f_j(x, \theta_{\text{MAP}})$
            $J[l][j] = [a^j_{l-1}, g^j_l]$
        **end for**
    **end for**
    **return** $J$
**end procedure**
**procedure** GLM-UNC($J_{\text{KF}}(x), \Sigma_{\text{KFAC}}$)   ▷ Compute Linearized predictive covariance from jacobian and curvature
    $M \leftarrow$ array[$d$][$L$]
    $N \leftarrow$ array[$d$]
    **for** $l \in$ range($L$) **do**
        **for** $j \in$ range($d$) **do**
            $Q^{-1}_{(l)}, H^{-1}_{(l)} \leftarrow \Sigma_{\text{KFAC}}[l]$
            $a^j_{l-1}, g^j_l \leftarrow J_{\text{KF}}(x)[l][j]$
            $M[l,j] \leftarrow \left({a^j_{l-1}}^T * Q^{-1}_{(l)} * a^j_{l-1}\right) * \left({q^j_l}^T * H^{-1}_{(l)} * q^j_l\right)$
        **end for**
        $N[j] \leftarrow \sum_{l=1}^{L} M[l,j]$
    **end for**
    **return** diag($N[1], \ldots, N[d]$)
**end procedure**
```

Algorithm 3 Sample from KFAC posterior

```
$\Sigma_{\text{KFAC}}$                                          ▷ $\Sigma$ in KFAC dictionary format
for $l \in \Sigma_{\text{KFAC}}$.keys() do
    $Q_l, H_l \leftarrow \Sigma_{\text{KFAC}}[l]$
    $l_{\text{in}}, l_{\text{out}} \leftarrow Q_l$.size()[0], $H_l$.size()[0]
    $Q_l, H_l \leftarrow \text{ch}(Q_l), \text{ch}(H_l)$
    $x_l \sim \mathcal{N}(0, I_{l_{\text{in}} * l_{\text{out}}})$
    $x_l \leftarrow x_l$.reshape($l_{\text{in}}, l_{\text{out}}$)
    $z_l \leftarrow Q_l * x_l * H_l$
end for
return dict($l = z_l$)
```

Algorithm 4 MC-GLM predictive (online)

```
$x \leftarrow$ input
$\epsilon \leftarrow$                                              ▷ small positive float
$f_{\theta_{\text{MAP}}} \leftarrow$ MAP model
$\Sigma_{\text{KFAC}} \leftarrow$ KFAC covariance
$\eta_1, \cdots, \eta_k \sim \mathcal{N}(0, \Sigma_{\text{KFAC}})$                ▷ Sample noise vectors 3
$G \leftarrow [\,]$
for $i \in \text{range}(k)$ do                              ▷ run through MC weight samples
    $G$.append($f(x, \theta_{\text{MAP}} + \epsilon * \eta_i)$)
end for
$A \leftarrow \text{stack}(G)$                                       ▷ size $(d, k)$
$A \leftarrow A(x) - [f(x, \theta_{\text{MAP}})] * k$                        ▷ size $(d, k)$
$A \leftarrow A/\epsilon$                                            ▷ size $(d, k)$
$B \leftarrow \frac{1}{k-1} A * A^T$                                    ▷ size $(d, d)$
return $B$
```

Algorithm 5 MCI predictive (online)

```
$x \leftarrow$ input
$f_{\theta_{\text{MAP}}} \leftarrow$ MAP model
$\Sigma_{\text{KFAC}} \leftarrow$ KFAC covariance
$\eta_1, \cdots, \eta_k \sim \mathcal{N}(0, \Sigma_{\text{KFAC}})$                ▷ Sample noise vectors 3
MCP $\leftarrow [\,]$                                          ▷ Monte Carlo Predictions
for $i \in \text{range}(k)$ do                              ▷ run through MC weight samples
    MCP.append$f(x, \theta_{\text{MAP}} + \eta_i)$
end for
$\overline{\text{MCP}} = \frac{1}{k} \sum_{i=1}^{k} \text{MCP}[i]$                         ▷ Bayesian prediction
$U = \frac{1}{k-1} \sum_{i=1}^{k} \left(\text{MCP}[i] - \overline{\text{MCP}}\right) * \left(\text{MCP}[i] - \overline{\text{MCP}}\right)^T$    ▷ Sample covariance
return $\overline{\text{MCP}}, U$
```

References

[1] Y. Gal. Zoubin Ghahramani; *Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning.* Proceedings of The 33rd International Conference on Machine Learning, PMLR 48:1050-1059, 2016.

[2] Y. Gal *Uncertainty in Deep Learning. PhD thesis*, University of Cambridge, 2016.

[3] Immer, Korzepa, Bauer *Improving predictions of bayesian neural nets via local linearization*, Proceedings of the 24th international conference on Artificial Intelligence and Statistics (AISTATS) 2021, PLMR: Volume 130, 2021.

[4] Lee, Humt, *Estimating Model Uncertainty of Neural Networks in Sparse Information Form*, ICML, 2020.

[5] Mackay *Bayesian model comparison and backprop-nets*, NeurIPS, 1992.

[6] Martens, G. *Optimizing neural networks with kronecker-factored approximate curvature*, Proceedings of the 32nd ICML, 2015.

[7] Martens, G. *A Kronecker-factored approximate Fisher matrix for convolutional layers*, International Conference on Machine Learning, 2016