Laplace Redux - Effortless Bayesian Deep Learning

Erik Daxberger, Agustinus Kristiadi,
Alexander Immer, Runa Eschenhagen,
Matthias Bauer, Philipp Hennig

Presenter: Lanya Yang

Jan 18, 2026

The paper

Laplace Redux – Effortless Bayesian Deep Learning

Erik Daxberger[*,c,m] **Agustinus Kristiadi**[*,t] **Alexander Immer**[*,e,p] **Runa Eschenhagen**[*,t]
Matthias Bauer[d] **Philipp Hennig**[t,m]

[c]University of Cambridge
[m]MPI for Intelligent Systems, Tübingen
[t]University of Tübingen
[e]Department of Computer Science, ETH Zurich
[p]Max Planck ETH Center for Learning Systems
[d]DeepMind, London

Abstract

Bayesian formulations of deep learning have been shown to have compelling theoretical properties and offer practical functional benefits, such as improved predictive uncertainty quantification and model selection. The Laplace approximation (LA) is a classic, and arguably the simplest family of approximations for the intractable posteriors of deep neural networks. Yet, despite its simplicity, the LA is not as popular as alternatives like variational Bayes or deep ensembles. This may be due to assumptions that the LA is expensive due to the involved Hessian computation, that it is difficult to implement, or that it yields inferior results. In this work we show that these are misconceptions: we (i) review the range of variants of the LA including versions with minimal cost overhead; (ii) introduce `laplace`, an easy-to-use software library for PyTorch offering user-friendly access to all major flavors of the LA; and (iii) demonstrate through extensive experiments that the LA is competitive with more popular alternatives in terms of performance, while excelling in terms of computational cost. We hope that this work will serve as a catalyst to a wider

https://arxiv.org/abs/2106.14806 Daxberger et al. (2021)

Bayesian Inference

Given data $\mathcal{D}$ and parameters θ, the posterior is

$$p(\theta \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \theta)\, p(\theta)}{Z}, \qquad Z = \int p(\mathcal{D} \mid \theta)\, p(\theta)\, d\theta.$$

Bayesian Inference

Given data $\mathcal{D}$ and parameters θ, the posterior is

$$p(\theta \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \theta)\, p(\theta)}{Z}, \qquad Z = \int p(\mathcal{D} \mid \theta)\, p(\theta)\, d\theta.$$

Challenge. Exact Bayesian inference is hard to compute:

- The normalising constant Z is rarely available in closed form.
- Predictive quantities require difficult integrals, e.g.

$$p(y \mid x_*, \mathcal{D}) = \int p(y \mid x_*, \theta)\, p(\theta \mid \mathcal{D})\, d\theta.$$

Approximate Bayesian inference methods

- **Variational Inference (VI)**:
  Choose a tractable family $q(\theta)$ and fit it by minimising $\mathrm{KL}\big(q(\theta) \,\|\, p(\theta \mid \mathcal{D})\big)$ (equivalently, maximise the ELBO)
- **Laplace Approximation (LA)**:
  Approximate $p(\theta \mid \mathcal{D})$ by a Gaussian distribution

Laplace Approximation

The maximum a posteriori estimation of θ is defined as

$$\theta_{\text{MAP}} = \arg\max_{\theta} p(\theta \mid \mathcal{D})$$

Let $h(\theta) = p(\mathcal{D}|\theta)p(\theta)$.

Laplace Approximation

The maximum a posteriori estimation of θ is defined as

$$\theta_{\text{MAP}} = \arg\max_{\theta} p(\theta \mid \mathcal{D})$$

Let $h(\theta) = p(\mathcal{D}|\theta)p(\theta)$. Taylor-expanding $\log h$ around θ_{MAP} up to the second order yield

$$\log h(\theta) \approx \log h(\theta_{\text{MAP}}) - \frac{1}{2}(\theta - \theta_{\text{MAP}})^{\top} \Lambda (\theta - \theta_{\text{MAP}}),$$

$$\text{where} \quad \Lambda := -\nabla^2_{\theta} \log h(\theta)\big|_{\theta=\theta_{\text{MAP}}},$$

Laplace Approximation

The maximum a posteriori estimation of θ is defined as

$$\theta_{\text{MAP}} = \arg\max_{\theta} p(\theta \mid \mathcal{D})$$

Let $h(\theta) = p(\mathcal{D}|\theta)p(\theta)$. Taylor-expanding $\log h$ around θ_{MAP} up to the second order yield

$$\log h(\theta) \approx \log h(\theta_{\text{MAP}}) - \frac{1}{2}(\theta - \theta_{\text{MAP}})^{\top} \Lambda (\theta - \theta_{\text{MAP}}),$$

$$\text{where} \quad \Lambda := -\nabla^2_{\theta} \log h(\theta)\big|_{\theta=\theta_{\text{MAP}}},$$

$$Z = \int h(\theta) d\theta \approx h(\theta_{\text{MAP}})(2\pi)^{d/2} (\det \Lambda)^{-1/2}.$$

Laplace Approximation

The maximum a posteriori estimation of θ is defined as

$$\theta_{\text{MAP}} = \arg\max_{\theta} p(\theta \mid \mathcal{D})$$

Let $h(\theta) = p(\mathcal{D}|\theta)p(\theta)$. Taylor-expanding $\log h$ around θ_{MAP} up to the second order yield

$$\log h(\theta) \approx \log h(\theta_{\text{MAP}}) - \frac{1}{2}(\theta - \theta_{\text{MAP}})^\top \Lambda (\theta - \theta_{\text{MAP}}),$$

$$\text{where} \quad \Lambda := -\nabla^2_\theta \log h(\theta)\big|_{\theta=\theta_{\text{MAP}}},$$

$$Z = \int h(\theta) d\theta \approx h(\theta_{\text{MAP}})(2\pi)^{d/2} (\det \Lambda)^{-1/2}.$$

Therefore,

$$p(\theta|\mathcal{D}) = \frac{h(\theta)}{Z} \approx (2\pi)^{-d/2}(\det \Lambda)^{1/2} \exp\left(-\frac{1}{2}(\theta - \theta_{\text{MAP}})^\top \Lambda (\theta - \theta_{\text{MAP}})\right).$$

which can be identified as the density of $\mathcal{N}(\theta; \theta_{\text{MAP}}, \Sigma)$, where $\Sigma = \Lambda^{-1}$. This is called Laplace Approximation (LA).

Deep learning

Deep learning

Given a training dataset $\mathcal{D} := \{(x_n \in \mathbb{R}^M,\ y_n \in \mathbb{R}^C)\}_{n=1}^N$, the weights $\theta \in \mathbb{R}^D$ of an **L-layer** NN $f_\theta : \mathbb{R}^M \to \mathbb{R}^C$ are trained to minimize the empirical risk

$$\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^D} \mathcal{L}(\mathcal{D}; \theta) = \arg\min_{\theta \in \mathbb{R}^D} \left(r(\theta) + \sum_{n=1}^{N} \ell(y_n, f_\theta(x_n)) \right).$$

where $\ell(\cdot, \cdot)$ is a loss function and $r(\theta)$ is a regularizer.

Limitations.

- Standard deep learning typically yields **point estimates** $\hat{\theta}$, with limited uncertainty quantification.
- Models can be **overconfident on out-of-distribution (OOD)** inputs.

Beyesian Deep Learning

From the **Bayesian viewpoint**,
$r(\theta) = -\log p(\theta)$ and $\ell(y_n, f_\theta(x_n)) = -\log p(y_n \mid f_\theta(x_n))$, so

$$\mathcal{L}(\mathcal{D}; \theta) = -\log h(\theta)$$

and

$$\hat{\theta} = \theta_{\text{MAP}}$$

Beyesian Deep Learning

From the **Bayesian viewpoint**,
$r(\theta) = -\log p(\theta)$ and $\ell(y_n, f_\theta(x_n)) = -\log p(y_n \mid f_\theta(x_n))$, so

$$\mathcal{L}(\mathcal{D}; \theta) = -\log h(\theta)$$

and

$$\hat{\theta} = \theta_{\text{MAP}}$$

For example, the widely used weight regularizer

$$r(\theta) = \tfrac{1}{2}\gamma^{-2}\|\theta\|_2^2$$

corresponds to a centered Gaussian prior

$$p(\theta) = \mathcal{N}(\theta; 0, \gamma^2 I).$$

Beyesian Deep Learning

For multi-class classification, let $f_\theta(x) \in \mathbb{R}^C$ be the logits and define the categorical likelihood as

$$p_\theta(y = c \mid x) = \text{softmax}(f_\theta(x))_c = \frac{\exp(f_{\theta,c}(x))}{\sum_{j=1}^{C} \exp(f_{\theta,j}(x))}.$$

Beyesian Deep Learning

For multi-class classification, let $f_\theta(x) \in \mathbb{R}^C$ be the logits and define the categorical likelihood as

$$p_\theta(y = c \mid x) = \mathrm{softmax}(f_\theta(x))_c = \frac{\exp(f_{\theta,c}(x))}{\sum_{j=1}^{C} \exp(f_{\theta,j}(x))}.$$

Then the corresponding loss (negative log-likelihood)

$$\ell\big(y, f_\theta(x)\big) = -\log p_\theta(y \mid x)$$

is exactly the **cross-entropy** loss :

$$\ell\big(y, f_\theta(x)\big) = -\log \mathrm{softmax}(f_\theta(x))_y.$$

LA in Bayesian Deep Learning

Assume a Gaussian prior on the weights:

$$p(\theta) = \mathcal{N}(\theta; 0, \gamma^2 I).$$

LA in Bayesian Deep Learning

Assume a Gaussian prior on the weights:

$$p(\theta) = \mathcal{N}(\theta; 0, \gamma^2 I).$$

Laplace approximation. Approximate the posterior by a Gaussian around the MAP:

$$p(\theta \mid \mathcal{D}) \approx \mathcal{N}\big(\theta; \theta_{\text{MAP}}, \Sigma\big), \qquad \Sigma = \Lambda^{-1},$$

where

$$\Lambda := -\nabla^2_\theta \log h(\theta)\big|_{\theta=\theta_{\text{MAP}}}, \qquad h(\theta) = \exp(-\mathcal{L}(\mathcal{D}; \theta)).$$

LA in Bayesian Deep Learning

Assume a Gaussian prior on the weights:

$$p(\theta) = \mathcal{N}(\theta; 0, \gamma^2 I).$$

Laplace approximation. Approximate the posterior by a Gaussian around the MAP:

$$p(\theta \mid \mathcal{D}) \approx \mathcal{N}\big(\theta; \theta_{\text{MAP}}, \Sigma\big), \qquad \Sigma = \Lambda^{-1},$$

where

$$\Lambda := -\nabla^2_\theta \log h(\theta)\big|_{\theta=\theta_{\text{MAP}}}, \qquad h(\theta) = \exp(-\mathcal{L}(\mathcal{D}; \theta)).$$

Decomposition of Λ.

$$\Lambda = -\nabla^2_\theta \log p(\theta) - \nabla^2_\theta \log p(\mathcal{D} \mid \theta)\Big|_{\theta=\theta_{\text{MAP}}}$$

$$= \gamma^{-2} I - \sum_{n=1}^{N} \nabla^2_\theta \log p\big(y_n \mid f_\theta(x_n)\big)\Big|_{\theta=\theta_{\text{MAP}}}.$$

Challenge. The second term scales quadratically with the number of network parameters.

Four components of LA

- Inference over all weights or subsets of weights
- Hessian approximation and their factorizations
- Hyperparameter Tuning
- Approximate predictive distribution

Hessian approximation

The **Fisher information matrix** is a **positive semi-definite** approximation to the Hessian of the log-likelihood of NNs.

Hessian approximation

The **Fisher information matrix** is a **positive semi-definite** approximation to the Hessian of the log-likelihood of NNs. Let $p(x, y; \theta)$ be the joint density function for X and Y conditional on the value of θ. (In our case, $p(x, y \mid \theta) = p(y \mid f_\theta(x))$).
The Fisher information matrix F of $P_{x,y}(\theta)$ is given by

$$F = \mathbb{E}_{p(x,y)}\left[\nabla_\theta \log p(x, y \mid \theta) \nabla_\theta \log p(x, y \mid \theta)^\top\right],$$
$$= -\mathbb{E}_{p(x,y)}\left[\nabla^2_\theta \log p(x, y \mid \theta)\right].$$

Hessian approximation

The **Fisher information matrix** is a **positive semi-definite** approximation to the Hessian of the log-likelihood of NNs. Let $p(x, y; \theta)$ be the joint density function for X and Y conditional on the value of θ. (In our case, $p(x, y \mid \theta) = p(y \mid f_\theta(x))$).
The Fisher information matrix F of $P_{x,y}(\theta)$ is given by

$$F = \mathbb{E}_{p(x,y)}\left[\nabla_\theta \log p(x, y \mid \theta) \nabla_\theta \log p(x, y \mid \theta)^\top\right],$$

$$= -\mathbb{E}_{p(x,y)}\left[\nabla^2_\theta \log p(x, y \mid \theta)\right].$$

Because $p(x, y \mid \theta) = p(y \mid x, \theta)\, q(x)$, we have Martens (2020)

$$\nabla_\theta \log p(x, y \mid \theta) = \nabla_\theta \log p(y \mid x, \theta) + \nabla_\theta \log q(x) = \nabla_\theta \log p(y \mid x, \theta).$$

Hessian approximation

The **Fisher information matrix** is a **positive semi-definite** approximation to the Hessian of the log-likelihood of NNs. Let $p(x, y; \theta)$ be the joint density function for X and Y conditional on the value of θ. (In our case, $p(x, y \mid \theta) = p(y \mid f_\theta(x))$).
The Fisher information matrix F of $P_{x,y}(\theta)$ is given by

$$F = \mathbb{E}_{p(x,y)}\left[\nabla_\theta \log p(x, y \mid \theta) \nabla_\theta \log p(x, y \mid \theta)^\top\right],$$
$$= -\mathbb{E}_{p(x,y)}\left[\nabla^2_\theta \log p(x, y \mid \theta)\right].$$

Because $p(x, y \mid \theta) = p(y \mid x, \theta)\, q(x)$, we have Martens (2020)

$$\nabla_\theta \log p(x, y \mid \theta) = \nabla_\theta \log p(y \mid x, \theta) + \nabla_\theta \log q(x) = \nabla_\theta \log p(y \mid x, \theta).$$

so F can be written as

$$F = \mathbb{E}_{q(x)}\left[\mathbb{E}_{p(y|x)}\left[\nabla_\theta \log p(y \mid x, \theta) \nabla_\theta \log p(y \mid x, \theta)^\top\right]\right].$$

Hessian approximation

Therefore, the Hessian

$$\sum_{n=1}^{N} \nabla_\theta^2 \log p\big(y_n \mid f_\theta(x_n)\big)\big|_{\theta=\theta_{\text{MAP}}}$$

can be approximated by the Fisher information matrix

$$F := \sum_{n=1}^{N} \mathbb{E}_{p(y|f_\theta(x_n))}\left[s(x_n, y)\, s(x_n, y)^\top\right], \tag{1}$$

where

$$s(x, y) := \nabla_\theta \log p\big(y \mid f_\theta(x)\big)\big|_{\theta=\theta_{\text{MAP}}}.$$

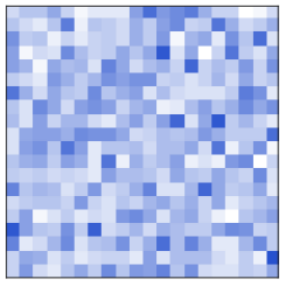
Hessian factorizations

Diagonal Factorization

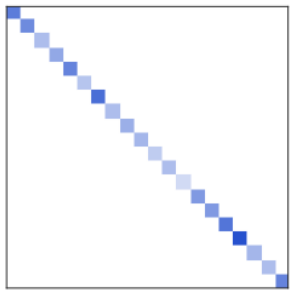
Assume that the negative-log-posterior's Hessian Λ is simply a diagonal matrix with diagonal elements equal the diagonal of the Fisher

$$\Lambda \approx -\gamma^{-2} I - \text{diag}(F)^{\top} I.$$

It requires storing only a length-D vector to represent F, and inverting Λ then costs just $\mathcal{O}(D)$ instead of $\mathcal{O}(D^3)$.



(a) Full Fisher



(b) Diagonal

KFAC

Kronecker-Factored Approximate Curvature (KFAC)

Consider an L-layer neural network with Fisher information matrix F over all parameters. KFAC first simplifies F by **ignoring cross-layer correlations**, yielding a block-diagonal approximation:

$$F \approx \text{diag}\left(F^{(1)}, \ldots, F^{(L)}\right),$$

where $F^{(l)}$ denotes the Fisher block associated with the parameters of layer l.

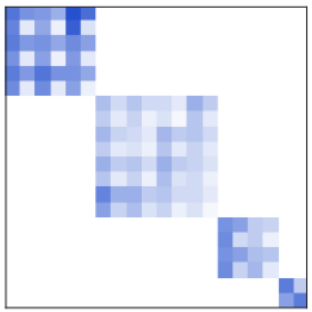


Figure 2: KFAC

KFAC: layer-wise structure

After approximating F as block-diagonal across layers, KFAC further simplifies each layer block $F^{(l)}$.

For layer $l = 1, \ldots, L$, let N_l be the number of units in layer l. The weight matrix

$$W^{(l)} \in \mathbb{R}^{N_l \times N_{l-1}}$$

has $N_l N_{l-1}$ parameters, so the corresponding Fisher block

$$F^{(l)} \in \mathbb{R}^{(N_l N_{l-1}) \times (N_l N_{l-1})}$$

would require $\mathcal{O}(N_l^2 N_{l-1}^2)$ memory to store in full.

KFAC: layer-wise structure

Define the per-sample quantities:

- $a^{(l)} \in \mathbb{R}^{N_l}$: output of layer l, i.e. the input to layer $l+1$.
- $g^{(l)} \in \mathbb{R}^{N_l}$: the log-likelihood gradient with respect to $a^{(l)}$.

KFAC: layer-wise structure

Define the per-sample quantities:

- $a^{(l)} \in \mathbb{R}^{N_l}$: output of layer l, i.e. the input to layer $l+1$.
- $g^{(l)} \in \mathbb{R}^{N_l}$: the log-likelihood gradient with respect to $a^{(l)}$.

For a feed-forward layer,

$$a^{(l)} = \sigma\left(W^{(l)} a^{(l-1)}\right)$$

where σ is the activation function.

KFAC: layer-wise structure

Define the per-sample quantities:

- $a^{(l)} \in \mathbb{R}^{N_l}$: output of layer l, i.e. the input to layer $l+1$.
- $g^{(l)} \in \mathbb{R}^{N_l}$: the log-likelihood gradient with respect to $a^{(l)}$.

For a feed-forward layer,

$$a^{(l)} = \sigma\left(W^{(l)} a^{(l-1)}\right)$$

where σ is the activation function. Therefore,

$$\frac{\partial \log p(y \mid f_\theta(x))}{\partial W_{ij}^{(l)}} = \frac{\partial \log p(y \mid f_\theta(x))}{\partial a_i^{(l)}} \frac{\partial a_i^{(l)}}{\partial W_{ij}^{(l)}} = g_i^{(l)} a_j^{(l-1)}$$

KFAC: layer-wise structure

Define the per-sample quantities:

- $a^{(l)} \in \mathbb{R}^{N_l}$: output of layer l, i.e. the input to layer $l+1$.
- $g^{(l)} \in \mathbb{R}^{N_l}$: the log-likelihood gradient with respect to $a^{(l)}$.

For a feed-forward layer,

$$a^{(l)} = \sigma\left(W^{(l)} a^{(l-1)}\right)$$

where σ is the activation function. Therefore,

$$\frac{\partial \log p(y \mid f_\theta(x))}{\partial W_{ij}^{(l)}} = \frac{\partial \log p(y \mid f_\theta(x))}{\partial a_i^{(l)}} \frac{\partial a_i^{(l)}}{\partial W_{ij}^{(l)}} = g_i^{(l)} a_j^{(l-1)}$$

Hence the outer product inside expectation in 1 can be written as

$$s(x_i, y)\, s(x_i, y)^\top = \mathrm{a}^{(l-1)} \mathrm{a}^{(l-1)\top} \otimes \mathrm{g}^{(l)} \mathrm{g}^{(l)\top}.$$

KFAC: layer-wise structure

Define the per-sample quantities:

- $a^{(l)} \in \mathbb{R}^{N_l}$: output of layer l, i.e. the input to layer $l+1$.
- $g^{(l)} \in \mathbb{R}^{N_l}$: the log-likelihood gradient with respect to $a^{(l)}$.

For a feed-forward layer,

$$a^{(l)} = \sigma\left(W^{(l)} a^{(l-1)}\right)$$

where σ is the activation function. Therefore,

$$\frac{\partial \log p(y \mid f_\theta(x))}{\partial W_{ij}^{(l)}} = \frac{\partial \log p(y \mid f_\theta(x))}{\partial a_i^{(l)}} \frac{\partial a_i^{(l)}}{\partial W_{ij}^{(l)}} = g_i^{(l)} a_j^{(l-1)}$$

Hence the outer product inside expectation in 1 can be written as

$$s(x_i, y)\, s(x_i, y)^\top = \mathrm{a}^{(l-1)} \mathrm{a}^{(l-1)\top} \otimes \mathrm{g}^{(l)} \mathrm{g}^{(l)\top}.$$

Furthermore, we assume that $a^{(l-1)}$ is independent of $g^{(l)}$,

$$F^{(l)} \approx \mathbb{E}\left[\mathrm{a}^{(l-1)} \mathrm{a}^{(l-1)\top}\right] \otimes \mathbb{E}\left[\mathrm{g}^{(l)} \mathrm{g}^{(l)\top}\right] =: A^{(l-1)} \otimes G^{(l)}.$$

KFAC

For each layer l, we obtain the l-th diagonal block of Λ — denoted by $\Lambda^{(l)}$ — by

$$\Lambda^{(l)} \approx \left(A^{(l-1)} + \sqrt{\lambda}\, I\right) \otimes \left(G^{(l)} + \sqrt{\lambda}\, I\right) =: V^{(l)} \otimes U^{(l)}.$$

Storage benefit. This reduces the memory cost from $\mathcal{O}(N_l^2 N_{l-1}^2)$ (full block) to $\mathcal{O}(N_l^2 + N_{l-1}^2)$.

Low-rank block-diagonal KFAC

Low-rank approximation based on KFAC. Starting from the Kronecker-factored Fisher block

$$F^{(l)} \approx A^{(l-1)} \otimes G^{(l)},$$

we eigendecompose the two Kronecker factors:

$$A^{(l-1)} = U_A^{(l-1)} S_A^{(l-1)} U_A^{(l-1)^\top}, \qquad G^{(l)} = U_G^{(l)} S_G^{(l)} U_G^{(l)^\top}.$$

Low-rank block-diagonal KFAC

Low-rank approximation based on KFAC. Starting from the Kronecker-factored Fisher block

$$F^{(l)} \approx A^{(l-1)} \otimes G^{(l)},$$

we eigendecompose the two Kronecker factors:

$$A^{(l-1)} = U_A^{(l-1)} S_A^{(l-1)} U_A^{(l-1)^\top}, \qquad G^{(l)} = U_G^{(l)} S_G^{(l)} U_G^{(l)^\top}.$$

Using properties of the Kronecker product, this yields

$$F^{(l)} \approx \left(U_A^{(l-1)} \otimes U_G^{(l)}\right)\left(S_A^{(l-1)} \otimes S_G^{(l)}\right)\left(U_A^{(l-1)} \otimes U_G^{(l)}\right)^\top.$$

Low-rank block-diagonal KFAC

Low-rank approximation based on KFAC. Starting from the Kronecker-factored Fisher block

$$F^{(l)} \approx A^{(l-1)} \otimes G^{(l)},$$

we eigendecompose the two Kronecker factors:

$$A^{(l-1)} = U_A^{(l-1)} S_A^{(l-1)} U_A^{(l-1)^\top}, \qquad G^{(l)} = U_G^{(l)} S_G^{(l)} U_G^{(l)^\top}.$$

Using properties of the Kronecker product, this yields

$$F^{(l)} \approx \left(U_A^{(l-1)} \otimes U_G^{(l)}\right)\left(S_A^{(l-1)} \otimes S_G^{(l)}\right)\left(U_A^{(l-1)} \otimes U_G^{(l)}\right)^\top.$$

The eigenvalues of $F^{(l)}$ are products of eigenvalues of $A^{(l-1)}$ and $G^{(l)}$. Keeping only the top k eigenvalues yields an optimal rank-k approximation $F_k^{(l)}$, greatly reducing storage and computation.

Low-rank approximation

Instead of approximating each layer block separately, we can approximate the entire curvature matrix $F \in \mathbb{R}^{D \times D}$ by a low-rank matrix.

Eigendecomposition. Since F is PSD,

$$F = Q\, L\, Q^\top,$$

where $Q = [q_1, \ldots, q_D]$ contains eigenvectors and $L = \operatorname{diag}(\lambda_1, \ldots, \lambda_D)$ with $\lambda_1 \geq \cdots \geq \lambda_D \geq 0$.

Low-rank approximation

Instead of approximating each layer block separately, we can approximate the entire curvature matrix $F \in \mathbb{R}^{D \times D}$ by a low-rank matrix.

Eigendecomposition. Since F is PSD,

$$F = Q\, L\, Q^\top,$$

where $Q = [q_1, \ldots, q_D]$ contains eigenvectors and $L = \operatorname{diag}(\lambda_1, \ldots, \lambda_D)$ with $\lambda_1 \geq \cdots \geq \lambda_D \geq 0$.

Rank-k truncation. Keep only the top k eigenpairs:

$$\hat{Q} = [q_1, \ldots, q_k] \in \mathbb{R}^{D \times k}, \qquad \hat{L} = \operatorname{diag}(\lambda_1, \ldots, \lambda_k) \in \mathbb{R}^{k \times k}.$$

Then the low-rank approximation is given by

$$F \approx F_k := \hat{Q}\, \hat{L}\, \hat{Q}^\top.$$

Storage benefit. This reduces the memory cost from $\mathcal{O}(D^2)$ (full block) to $\mathcal{O}(Dk)$.

Wrap-up

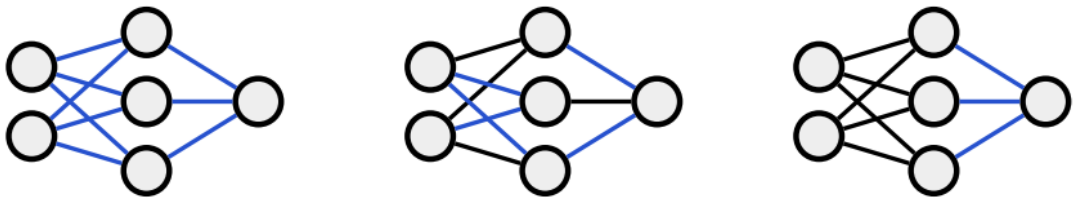
- Hessian approximation
  - Fisher information
  - Gauss-Newton matrix (GGN)
- Hessian factorization
  - Digagonal factorization
  - KFAC
    Low-rank block-diagonal KFAC
  - Low-rank approximation

Subnetwork

Subnetwork LA only applies the Laplace approximation to a subset of the model parameters, while keeping the remaining parameters fixed at their MAP estimates.
Last-Layer applies the LA to only the last linear layer of an L-layer NN. Suppose $W^{(L)}$ is the last-layer weight matrix of the network

$$p(W^{(L)} \mid \mathcal{D}) \approx \mathcal{N}(W^{(L)} \mid W^{(L)}_{\text{MAP}}, \Sigma^{(L)}).$$

① **Weights to be treated probabilistically with Laplace**



(a) All **(b)** Subnetwork **(c)** Last-Layer

```
Laplace(.., subset_of_weights='all'|'subnetwork'|'last_layer')
```

Monte Carlo Integration

Monte Carlo Integration

$$p(y \mid x^*, D) \approx \frac{1}{S} \sum_{s=1}^{S} p\big(y \mid f_{\theta_s}(x^*)\big), \quad \theta_s \sim q(\theta \mid D).$$

Approximate the distribution of netwrok outputs

We use **linearization** to approximate the network as

$$f_\theta(x^*) \approx f_{\theta_{\text{MAP}}}(x^*) + J_*^\top(\theta - \theta_{\text{MAP}}), \tag{2}$$

where $J_* := \nabla_\theta f_\theta(x^*)|_{\theta_{\text{MAP}}} \in \mathbb{R}^{d \times c}$ denotes the Jacobian of the network output with respect to the parameters, evaluated at θ_{MAP}. This way, under a Gaussian approximate posterior $q(\theta \mid \mathcal{D})$, the marginal distribution over the network output $f_* := f(x_*)$ is approximately Gaussian:

$$\begin{aligned} p(f_* \mid x_*, \mathcal{D}) &= \int \delta\big(f_* - f_\theta(x_*)\big)\, q(\theta \mid \mathcal{D})\, d\theta \\ &\approx \mathcal{N}\big(f_* \mid f_{\theta_{\text{MAP}}}(x_*),\ J_*^\top \Sigma J_*\big), \end{aligned}$$

where $J_* := \nabla_\theta f_\theta(x_*)\big|_{\theta=\theta_{\text{MAP}}}$.

References

Daxberger, E., Kristiadi, A., Immer, A., Eschenhagen, R., Bauer, M. and Hennig, P. (2021). Laplace redux-effortless bayesian deep learning, Advances in neural information processing systems **34**: 20089–20103.

Martens, J. (2020). New insights and perspectives on the natural gradient method, Journal of Machine Learning Research **21**(146): 1–76.