The Linearized Laplace Approximation in Bayesian Neural networks

José Miguel Hernández–Lobato
Department of Engineering
University of Cambridge

http://jmhl.org, jmh233@cam.ac.uk

Outline for this tutorial

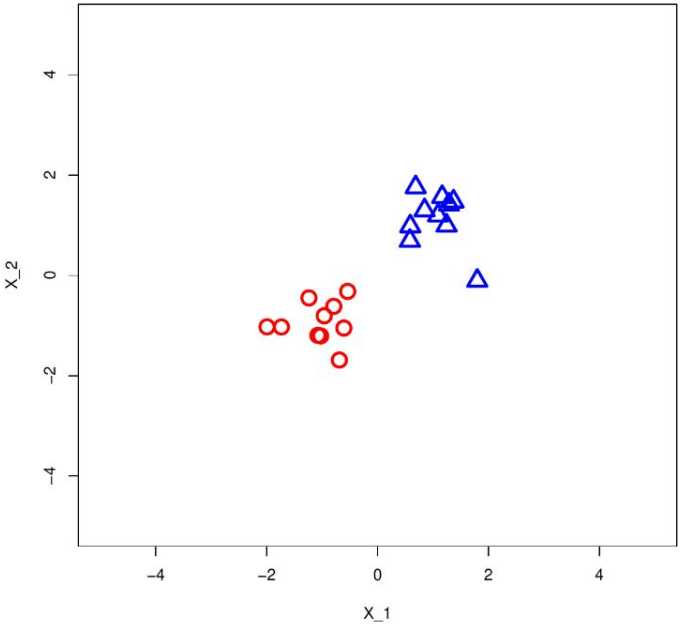
1. Motivation for a Bayesian approach
2. The basics of the Laplace approximation
3. The Laplace approximation in Bayesian neural networks
4. More recent works on the Laplace approximation
5. Subnetwork inference
6. Adapting the linearized Laplace model evidence
7. Case study: X-ray image reconstruction

Section 1

Motivation for a Bayesian Approach

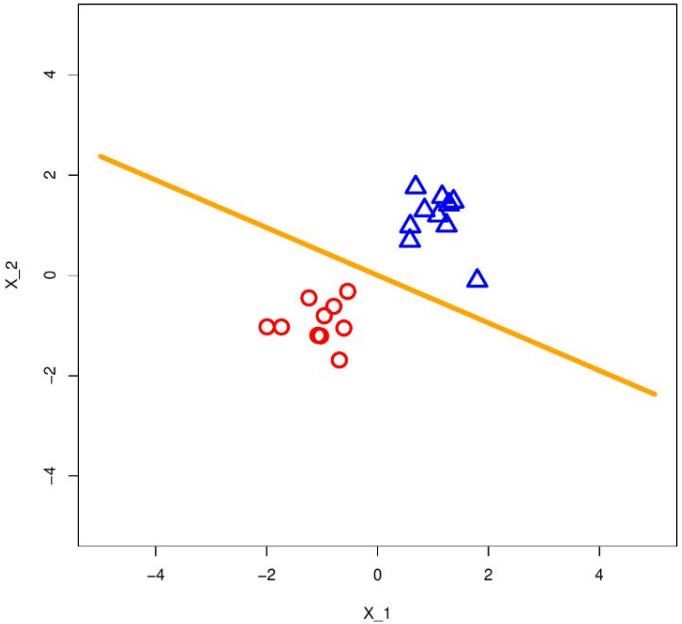
Consider linear classification with **linearly separable** data $\{y_n, \boldsymbol{x}_n\}_{n=1}^N$, $y_n \in \{-1, 1\}$, $\boldsymbol{x}_n \in \mathbb{R}^d$.

Many separating hyperplanes $\boldsymbol{w}$ fit the data equally well. Which one to choose?



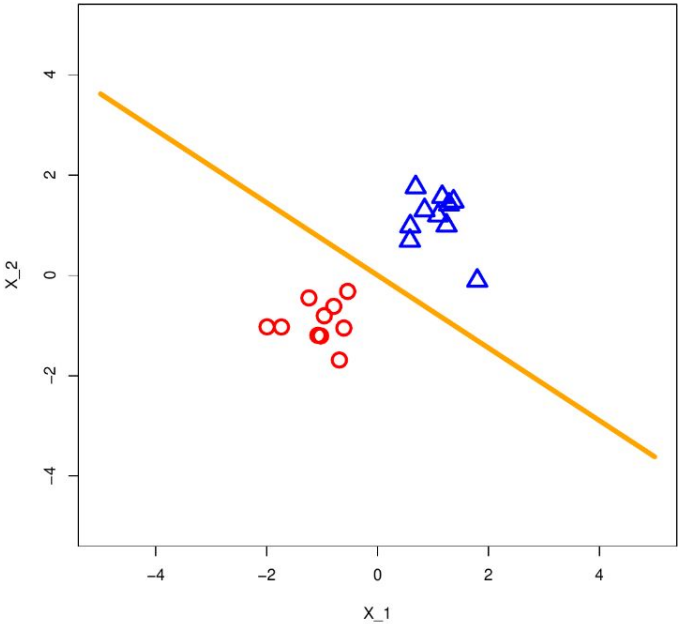
Consider linear classification with **linearly separable** data $\{y_n, \boldsymbol{x}_n\}_{n=1}^N$, $y_n \in \{-1, 1\}$, $\boldsymbol{x}_n \in \mathbb{R}^d$.

Many separating hyperplanes $\boldsymbol{w}$ fit the data equally well. Which one to choose?



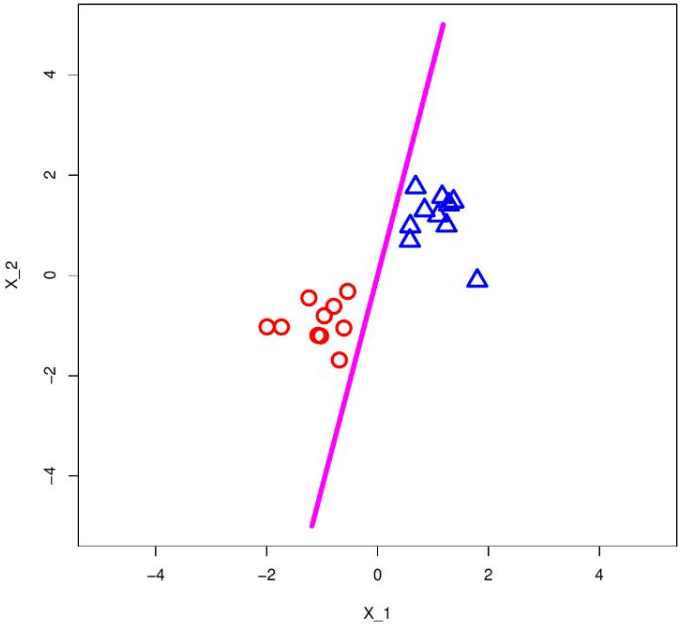
Consider linear classification with **linearly separable** data $\{y_n, \boldsymbol{x}_n\}_{n=1}^N$, $y_n \in \{-1, 1\}$, $\boldsymbol{x}_n \in \mathbb{R}^d$.

Many separating hyperplanes $\boldsymbol{w}$ fit the data equally well. Which one to choose?



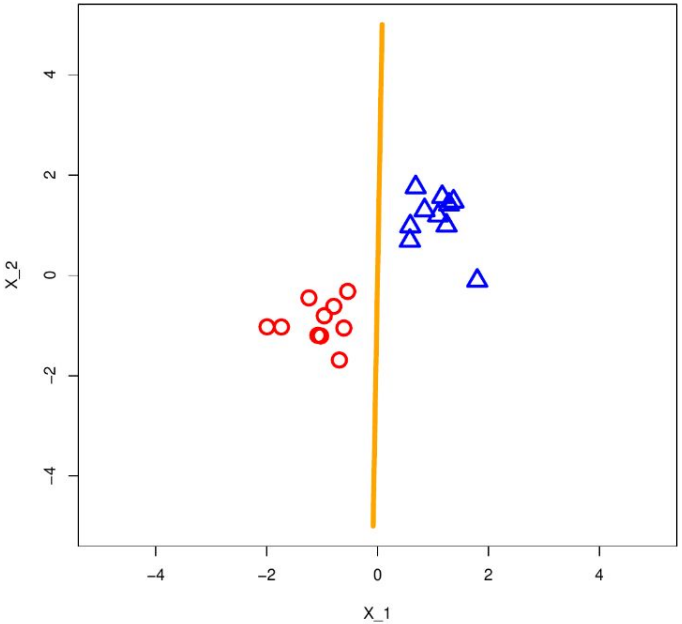
Consider linear classification with **linearly separable** data $\{y_n, \boldsymbol{x}_n\}_{n=1}^N$, $y_n \in \{-1, 1\}$, $\boldsymbol{x}_n \in \mathbb{R}^d$.

Many separating hyperplanes $\boldsymbol{w}$ fit the data equally well. Which one to choose?



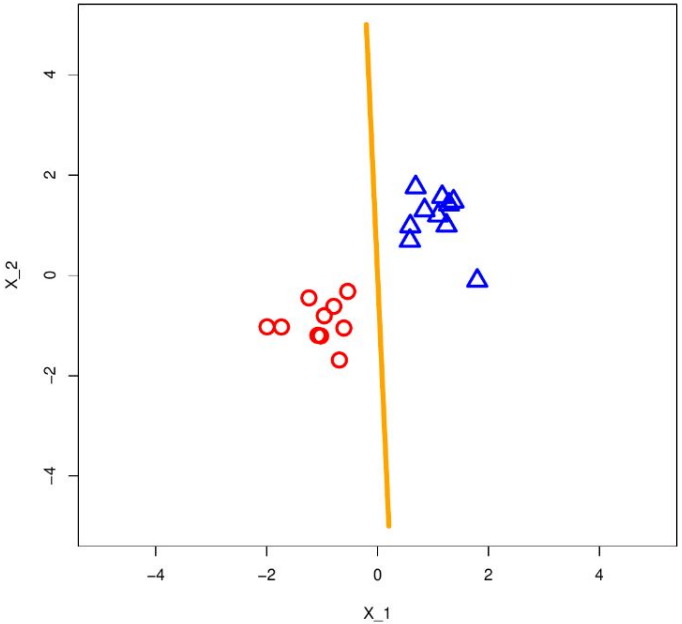
Consider linear classification with **linearly separable** data $\{y_n, \boldsymbol{x}_n\}_{n=1}^N$, $y_n \in \{-1, 1\}$, $\boldsymbol{x}_n \in \mathbb{R}^d$.

Many separating hyperplanes $\boldsymbol{w}$ fit the data equally well. Which one to choose?



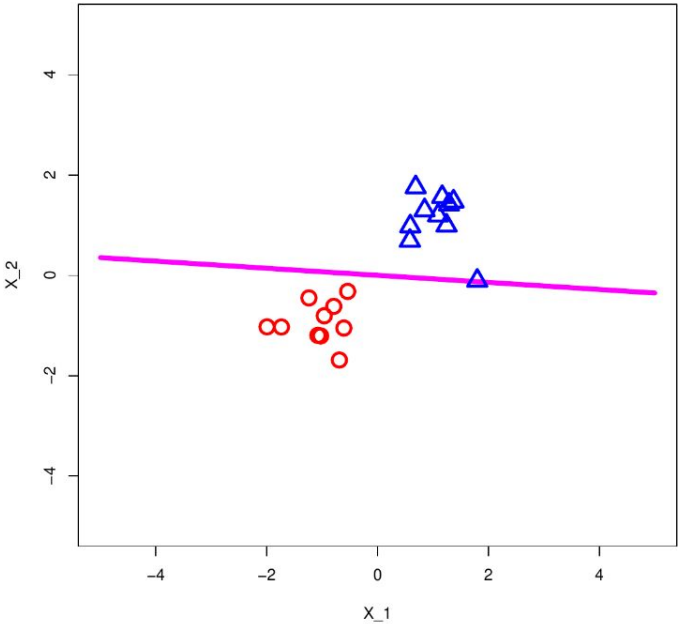
Consider linear classification with **linearly separable** data $\{y_n, \boldsymbol{x}_n\}_{n=1}^N$, $y_n \in \{-1, 1\}$, $\boldsymbol{x}_n \in \mathbb{R}^d$.

Many separating hyperplanes $\boldsymbol{w}$ fit the data equally well. Which one to choose?



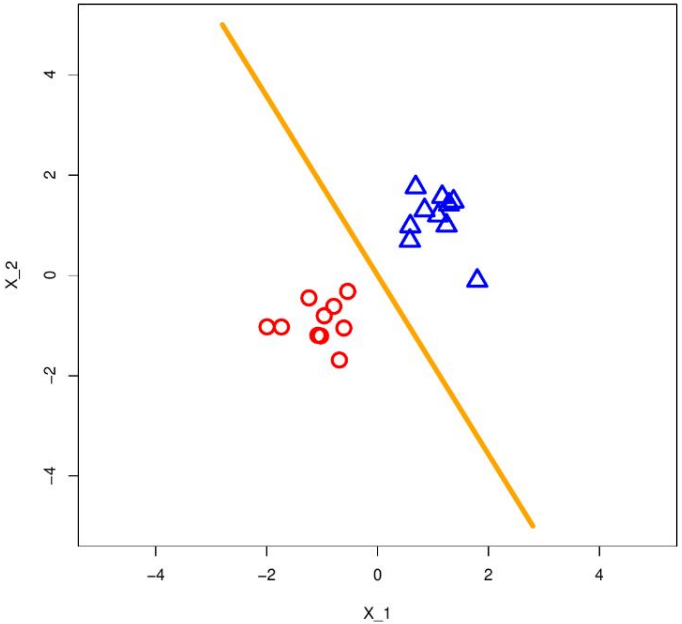
Consider linear classification with **linearly separable** data $\{y_n, \boldsymbol{x}_n\}_{n=1}^N$, $y_n \in \{-1, 1\}$, $\boldsymbol{x}_n \in \mathbb{R}^d$.

Many separating hyperplanes $\boldsymbol{w}$ fit the data equally well. Which one to choose?



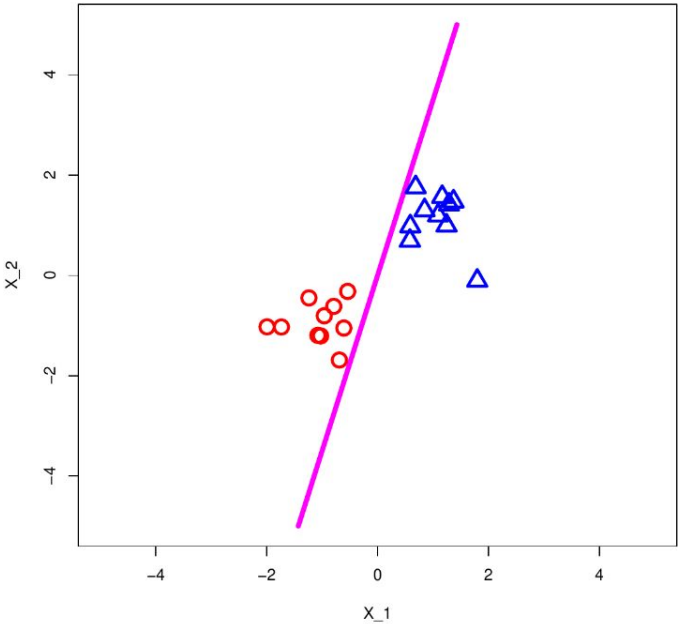
Consider linear classification with **linearly separable** data $\{y_n, \boldsymbol{x}_n\}_{n=1}^N$, $y_n \in \{-1, 1\}$, $\boldsymbol{x}_n \in \mathbb{R}^d$.

Many separating hyperplanes $\boldsymbol{w}$ fit the data equally well. Which one to choose?



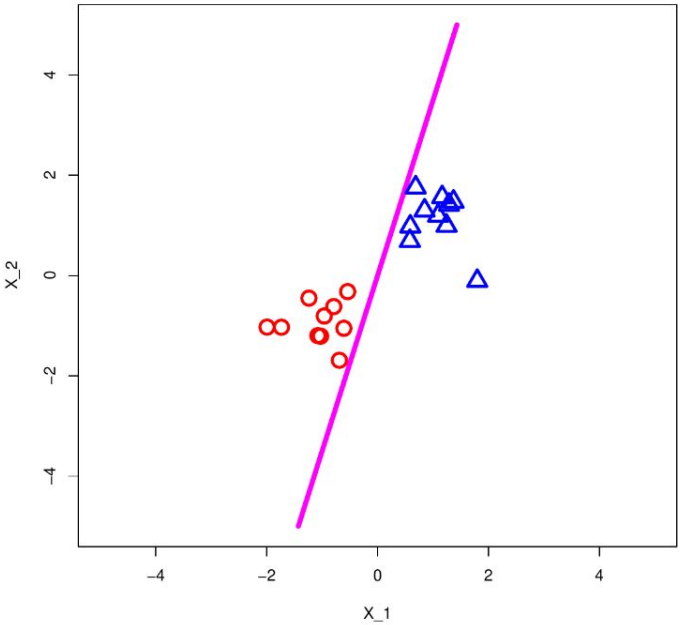
Consider linear classification with **linearly separable** data $\{y_n, \boldsymbol{x}_n\}_{n=1}^N$, $y_n \in \{-1, 1\}$, $\boldsymbol{x}_n \in \mathbb{R}^d$.

Many separating hyperplanes $\boldsymbol{w}$ fit the data equally well. Which one to choose?



Consider linear classification with **linearly separable** data $\{y_n, \boldsymbol{x}_n\}_{n=1}^N$, $y_n \in \{-1, 1\}$, $\boldsymbol{x}_n \in \mathbb{R}^d$.

Many separating hyperplanes $\boldsymbol{w}$ fit the data equally well. Which one to choose?



Solution: Bayesian inference.

The **prior** on $\boldsymbol{w}$ is $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|0, \alpha^{-1}\boldsymbol{I})$.

The **posterior** on $\boldsymbol{w}$ is then given by

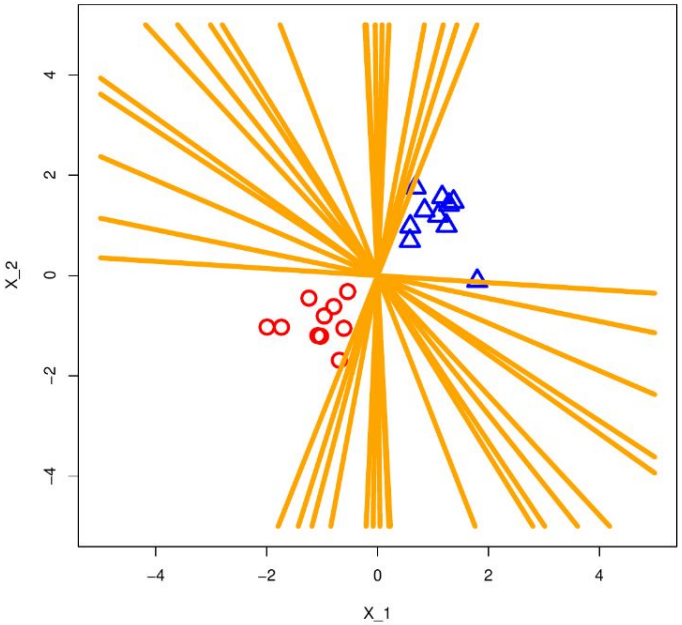
$$p(\boldsymbol{w}|\boldsymbol{y}, \boldsymbol{X}) \propto \left[\prod_{n_1}^{N} \sigma(y_n \boldsymbol{w}^\mathsf{T} \boldsymbol{x}_n)\right] p(\boldsymbol{w}),$$

where $\sigma(x) = 1/(1 + e^{-x})$. **Predictions** done with

$$p(y_\star|\boldsymbol{x}_\star, \boldsymbol{y}, \boldsymbol{X}) = \int \sigma(y_\star \boldsymbol{w}^\mathsf{T} \boldsymbol{x}_\star) p(\boldsymbol{w}|\boldsymbol{y}, \boldsymbol{X})\, d\boldsymbol{w},$$

Consider linear classification with **linearly separable** data $\{y_n, \boldsymbol{x}_n\}_{n=1}^N$, $y_n \in \{-1, 1\}$, $\boldsymbol{x}_n \in \mathbb{R}^d$.

Many separating hyperplanes $\boldsymbol{w}$ fit the data equally well. Which one to choose?



Solution: Bayesian inference.

The **prior** on $\boldsymbol{w}$ is $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|0, \alpha^{-1}\boldsymbol{I})$.

The **posterior** on $\boldsymbol{w}$ is then given by

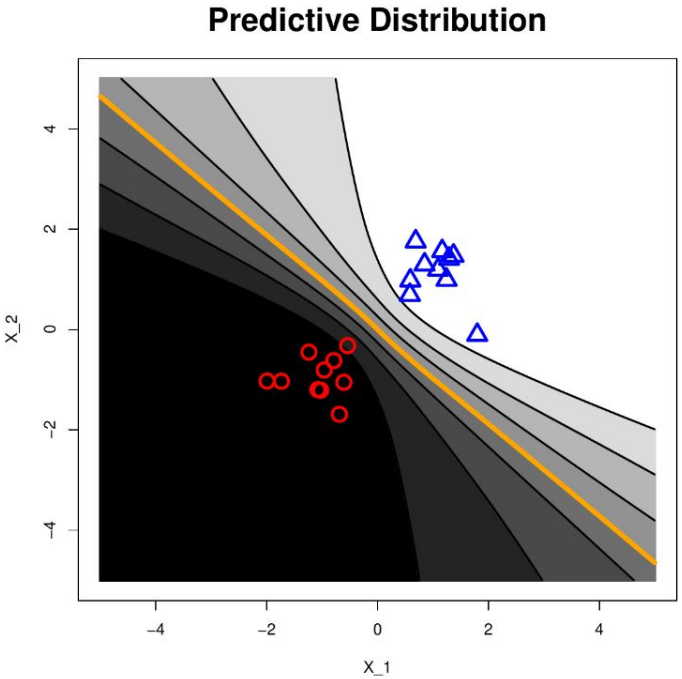
$$p(\boldsymbol{w}|\boldsymbol{y}, \boldsymbol{X}) \propto \left[\prod_{n_1}^{N} \sigma(y_n \boldsymbol{w}^\mathsf{T} \boldsymbol{x}_n)\right] p(\boldsymbol{w}),$$

where $\sigma(x) = 1/(1 + e^{-x})$. **Predictions** done with

$$p(y_\star|\boldsymbol{x}_\star, \boldsymbol{y}, \boldsymbol{X}) = \int \sigma(y_\star \boldsymbol{w}^\mathsf{T} \boldsymbol{x}_\star) p(\boldsymbol{w}|\boldsymbol{y}, \boldsymbol{X})\, d\boldsymbol{w},$$

Consider linear classification with **linearly separable** data $\{y_n, \boldsymbol{x}_n\}_{n=1}^N$, $y_n \in \{-1, 1\}$, $\boldsymbol{x}_n \in \mathbb{R}^d$.

Many separating hyperplanes $\boldsymbol{w}$ fit the data equally well. Which one to choose?



Solution: Bayesian inference.

The **prior** on $\boldsymbol{w}$ is $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|0, \alpha^{-1}\boldsymbol{I})$.

The **posterior** on $\boldsymbol{w}$ is then given by

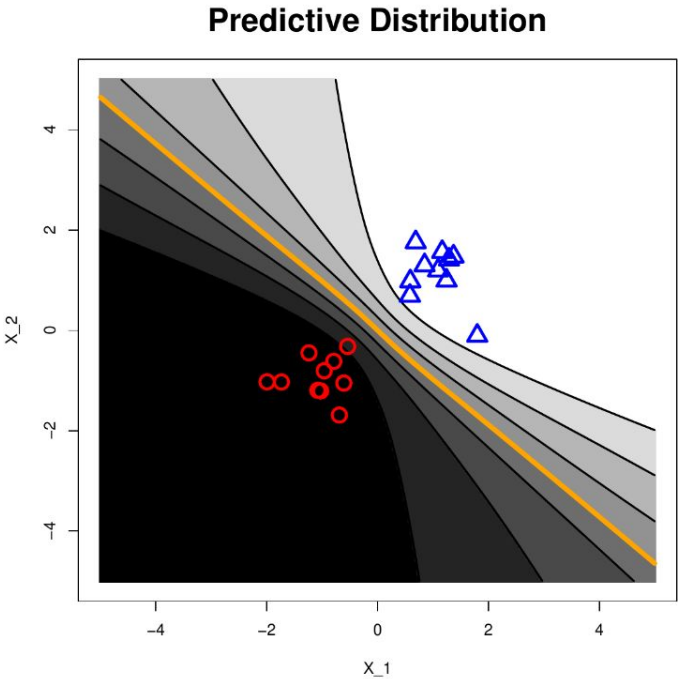
$$p(\boldsymbol{w}|\boldsymbol{y}, \boldsymbol{X}) \propto \left[\prod_{n_1}^{N} \sigma(y_n \boldsymbol{w}^\mathsf{T} \boldsymbol{x}_n)\right] p(\boldsymbol{w}),$$

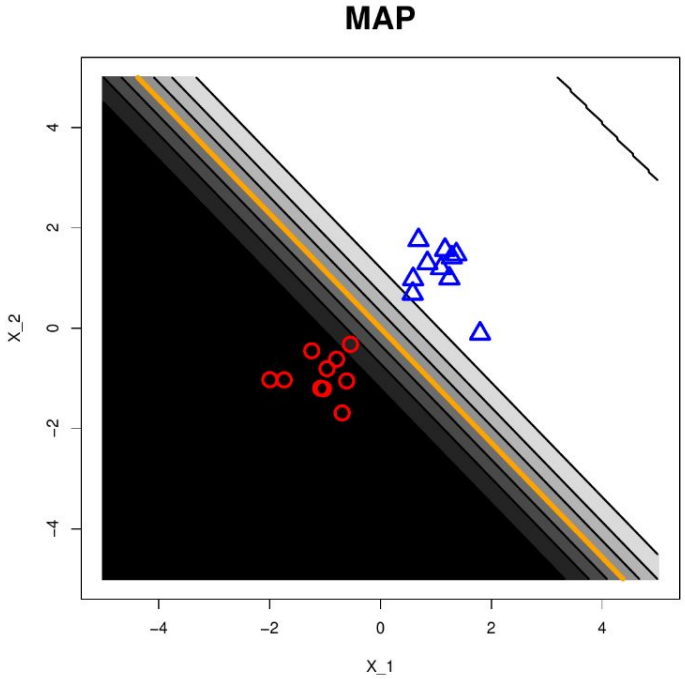
where $\sigma(x) = 1/(1 + e^{-x})$. **Predictions** done with

$$p(y_\star|\boldsymbol{x}_\star, \boldsymbol{y}, \boldsymbol{X}) = \int \sigma(y_\star \boldsymbol{w}^\mathsf{T} \boldsymbol{x}_\star) p(\boldsymbol{w}|\boldsymbol{y}, \boldsymbol{X})\, d\boldsymbol{w},$$

Consider linear classification with **linearly separable** data $\{y_n, \boldsymbol{x}_n\}_{n=1}^N$, $y_n \in \{-1, 1\}$, $\boldsymbol{x}_n \in \mathbb{R}^d$.

Many separating hyperplanes $\boldsymbol{w}$ fit the data equally well. Which one to choose?

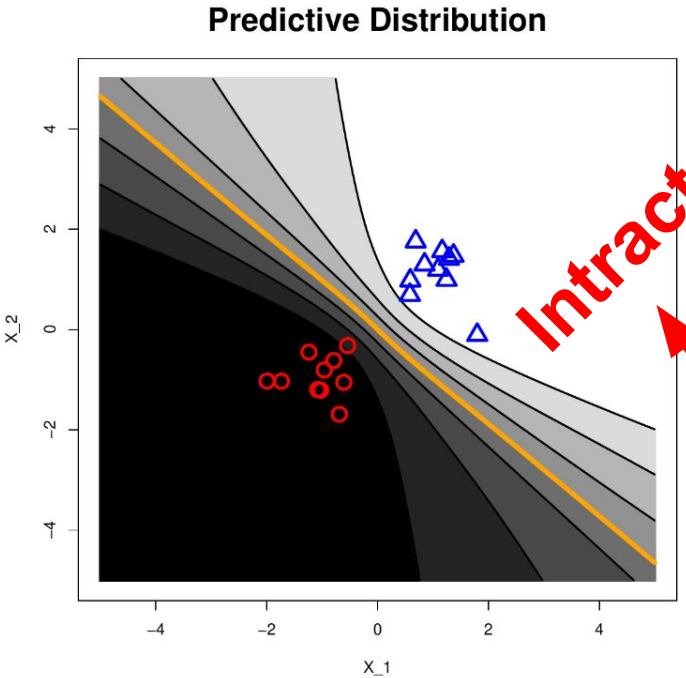




Difference w.r.t. *maximum a posteriori* (MAP) solution: **higher uncertainty** far from data.

Consider linear classification with **linearly separable** data $\{y_n, \boldsymbol{x}_n\}_{n=1}^N$, $y_n \in \{-1, 1\}$, $\boldsymbol{x}_n \in \mathbb{R}^d$.

Many separating hyperplanes $\boldsymbol{w}$ fit the data equally well. Which one to choose?



Intractable!

Solution: Bayesian inference.

The **prior** on $\boldsymbol{w}$ is $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|0, \alpha^{-1}\boldsymbol{I})$.

The **posterior** on $\boldsymbol{w}$ is then given by

$$p(\boldsymbol{w}|\boldsymbol{y}, \boldsymbol{X}) \propto \left[\prod_{n_1}^{N} \sigma(y_n \boldsymbol{w}^\mathsf{T} \boldsymbol{x}_n)\right] p(\boldsymbol{w}),$$

where $\sigma(x) = 1/(1 + e^{-x})$. **Predictions** done with

$$p(y_\star|\boldsymbol{x}_\star, \boldsymbol{y}, \boldsymbol{X}) = \int \sigma(y_\star \boldsymbol{w}^\mathsf{T} \boldsymbol{x}_\star) p(\boldsymbol{w}|\boldsymbol{y}, \boldsymbol{X})\, d\boldsymbol{w},$$

Difference w.r.t. *maximum a posteriori* (MAP) solution: **higher uncertainty** far from data.

Section 2

The basics of the Laplace approximation

Main idea behind Laplace approximation

Approximate complicated and possibly non-normalized distribution with a **Gaussian**.

One-dimensional Example

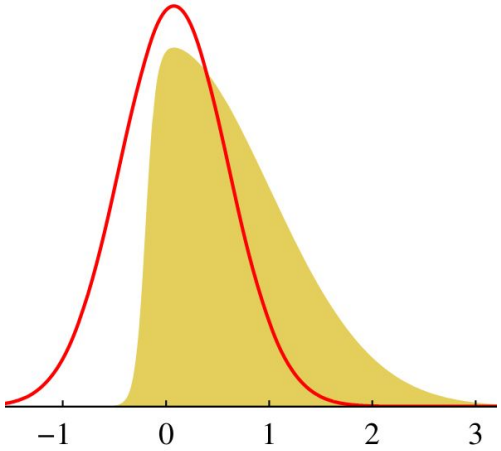
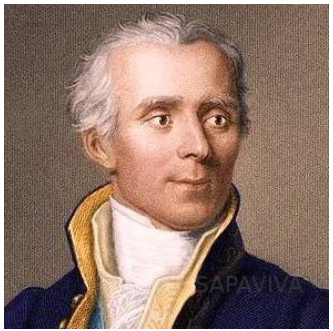
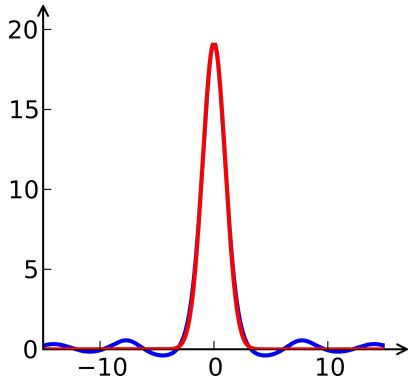


Figure source: Bishop, 2008.

Method named after **Pierre-Simon Laplace**, who used this approach to approximate integrals:

1. Replace integrand with Gaussian.
2. Integrate Gaussian approximation.





The Laplace approximation

Consider a scalar continuous w with

$$p(w|\mathcal{D}) = \frac{p(\mathcal{D}|w)p(w)}{p(\mathcal{D})} = \frac{p(\mathcal{D}, w)}{p(\mathcal{D})} = \frac{1}{Z} f(w)\,,$$

where $f(w) = p(w, \mathcal{D})$ for some data $\mathcal{D}$ and $Z = p(\mathcal{D}) = \int f(w) dw$.

Let $q(w) = \mathcal{N}(w|m, v)$ be the Gaussian approximation.

How to adjust m and v?

m can be the **MAP solution**, a **mode** w_{MAP} of $p(w|\mathcal{D})$:

$$\left.\frac{df(w)}{dw}\right|_{w=w_{\text{MAP}}} = 0$$

Any **optimization algorithm** can be used for finding w_{MAP}.

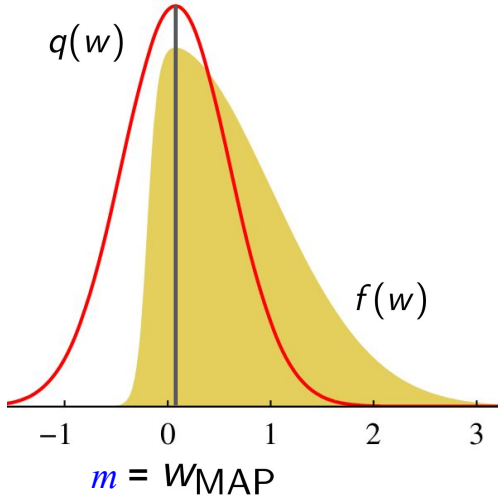


Figure source: Bishop, 2008.

The Laplace approximation

Given $m = w_{\text{MAP}}$, **what should the value of v be?**

We consider a **truncated Taylor expansion** of $\log f(w)$ center at w_{MAP}:

$$\log f(w) \approx \log f(w_{\text{MAP}}) - \frac{1}{2} a (w - w_{\text{MAP}})^2 , \qquad a = -\frac{d^2}{dw^2} \log f(w) \Big|_{w=w_{\text{MAP}}}$$

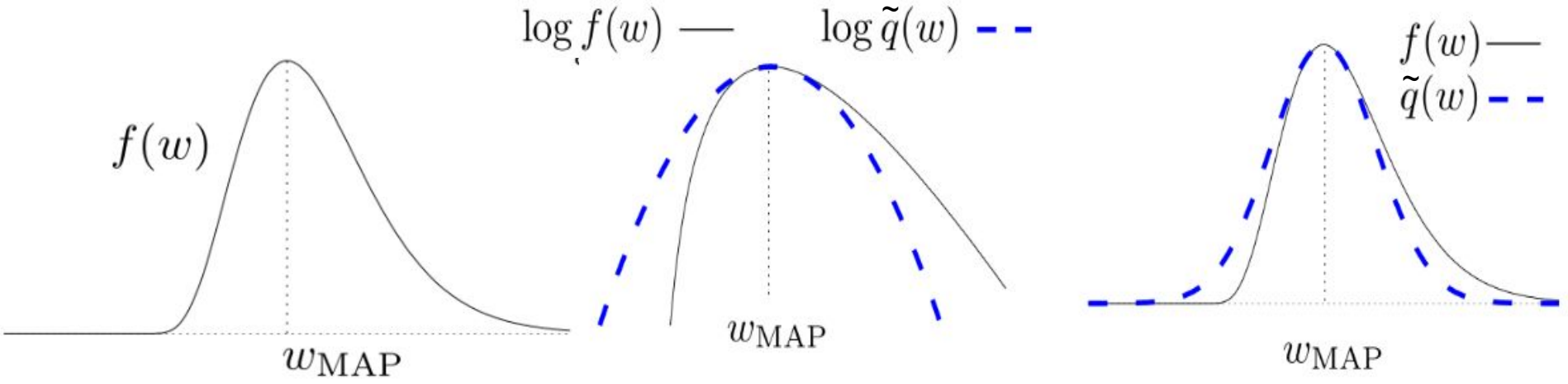
Taking the exponential we obtain:

$$f(w) \approx f(w_{\text{MAP}}) \exp\left\{ -\frac{a}{2} (w - w_{\text{MAP}})^2 \right\} = \tilde{q}(w) \qquad q(w) = \mathcal{N}(w | w_{\text{MAP}}, a^{-1}) ,$$

The exponentiated truncated series is **Gaussian**. Very easy to normalize!

The **normalizer** of $f(w)$ is approximated as $Z = \int f(w) dw \approx \int \tilde{q}(w) dw = f(w_{\text{MAP}}) \sqrt{2\pi / a}$.

1d example


$f(w)$
w_{MAP}
$\log f(w)$
$\log \tilde{q}(w)$
w_{MAP}
$f(w)$
$\tilde{q}(w)$
w_{MAP}

The multivariate case

Now $\boldsymbol{w}$ is a d-dimensional vector and $p(\boldsymbol{w}|\mathcal{D}) = \frac{1}{Z} f(\boldsymbol{w})$.

The same principle can be applied. The truncated Taylor series is

$$\log f(\boldsymbol{w}_{\text{MAP}}) \approx \log f(\boldsymbol{w}_{\text{MAP}}) - \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}_{\text{MAP}})^{\mathsf{T}} \boldsymbol{A} (\boldsymbol{w} - \boldsymbol{w}_{\text{MAP}}) \,,$$

where $\boldsymbol{A}$ is the **Hessian** of $-\log f(\boldsymbol{w})$ at $\boldsymbol{w}_{\text{MAP}}$, $\boldsymbol{A} = -\nabla^{\mathsf{T}} \nabla \log f(\boldsymbol{w})\big|_{\boldsymbol{w}=\boldsymbol{w}_{\text{MAP}}}$.

Taking the exponential we obtain a **multi-variate Gaussian** approximation:

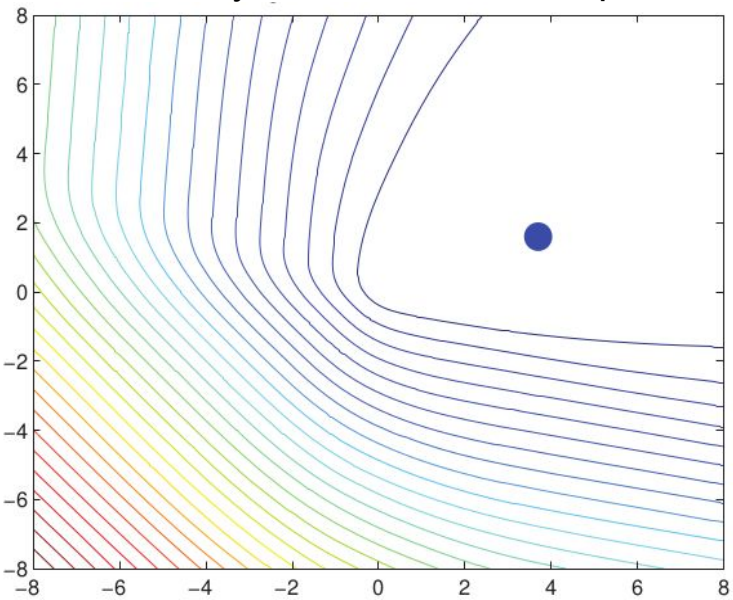
$$f(\boldsymbol{w}) \approx f(\boldsymbol{w}_{\text{MAP}}) \exp\left\{-\frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}_{\text{MAP}})^{\mathsf{T}} \boldsymbol{A} (\boldsymbol{w} - \boldsymbol{w}_{\text{MAP}}))\right\} = \tilde{q}(\boldsymbol{w}) \,,$$

$$q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{w}_{\text{MAP}}, \boldsymbol{A}^{-1}) \,,$$

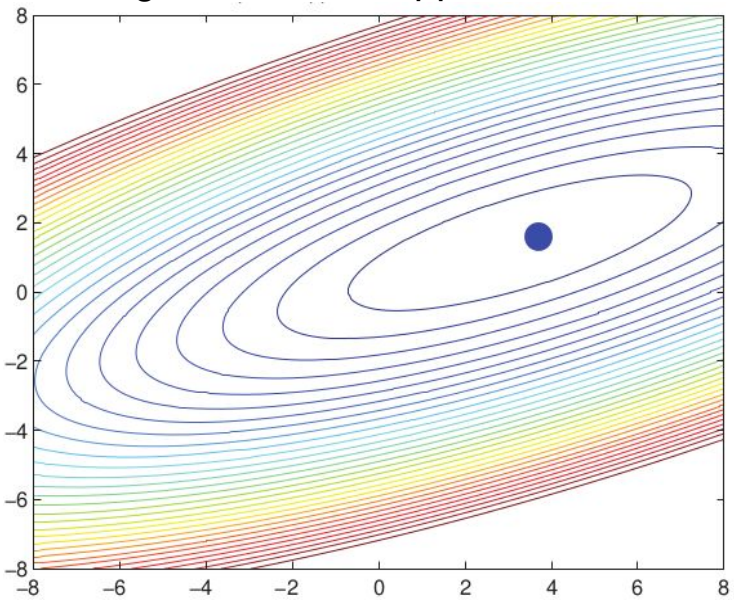
$$Z = \int f(\boldsymbol{w}) d\mathrm{w} \approx \int \tilde{q}(\boldsymbol{w}) d\boldsymbol{w} = f(\boldsymbol{w}_{\text{MAP}}) (2\pi)^{d/2} |\boldsymbol{A}|^{-1/2} \,.$$

2d example

Log un-nomralized posterior
in binary classification example



Log of Gaussian approximation



Figures source: Murphy 2012.

Example: Bayesian logistic regression

The posterior distribution is

$$p(\boldsymbol{w}|\mathcal{D}) \propto p(\boldsymbol{y}|\boldsymbol{w}, \boldsymbol{X})p(\boldsymbol{w}) = f(\boldsymbol{w}), \qquad \log f(\boldsymbol{w}) = \log p(\boldsymbol{y}|\boldsymbol{w}, \boldsymbol{X}) + \log p(\boldsymbol{w}).$$

Let $\sigma(x) = 1/(1 + e^x)$ be the logistic function. Then, we have that

$$\log f(\boldsymbol{w}) = \sum_{n=1}^{N} \log \sigma(y_n \boldsymbol{w}^\mathsf{T} \boldsymbol{x}_n) - \frac{\alpha}{2} \boldsymbol{w}^\mathsf{T} \boldsymbol{w} + \text{const}.$$

Let $\boldsymbol{w}_\text{MAP}$ maximize f. Using $\sigma'(x) = \sigma(x)\sigma(-x)$, the negative Hessian of $f(\boldsymbol{w})$ at $\boldsymbol{w}_\text{MAP}$ is

$$\boldsymbol{A} = \left[\sum_{n=1}^{N} \sigma(y_n \boldsymbol{w}^\mathsf{T} \boldsymbol{x}_n)\sigma(-y_n \boldsymbol{w}^\mathsf{T} \boldsymbol{x}_n) \boldsymbol{x}_n \boldsymbol{x}_n^\mathsf{T} \right] + \alpha \boldsymbol{I}.$$

We then have $q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{w}_\text{MAP}, \boldsymbol{A}^{-1})$ and $\log Z \approx \log f(\boldsymbol{w}_\text{MAP}) - \frac{1}{2} \log |\boldsymbol{A}| + \text{const}.$

Probit approximation to the predictive distribution

We need to evaluate $p(y_\star|\boldsymbol{x}_\star, \boldsymbol{y}) \approx \int p(y_\star|\boldsymbol{x}_\star, \boldsymbol{w})q(\boldsymbol{w})d\boldsymbol{w} = \int \sigma(y_\star \boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}_\star)\mathcal{N}(\boldsymbol{w}|\boldsymbol{w}_{\mathrm{MAP}}, \boldsymbol{A}^{-1})d\boldsymbol{w}$

We approximate $\sigma(x)$ with the standard Gaussian cdf $\Phi(x)$ or **probit function**.

We scale $\Phi(x)$ to have same slope at origin as $\sigma(x)$:

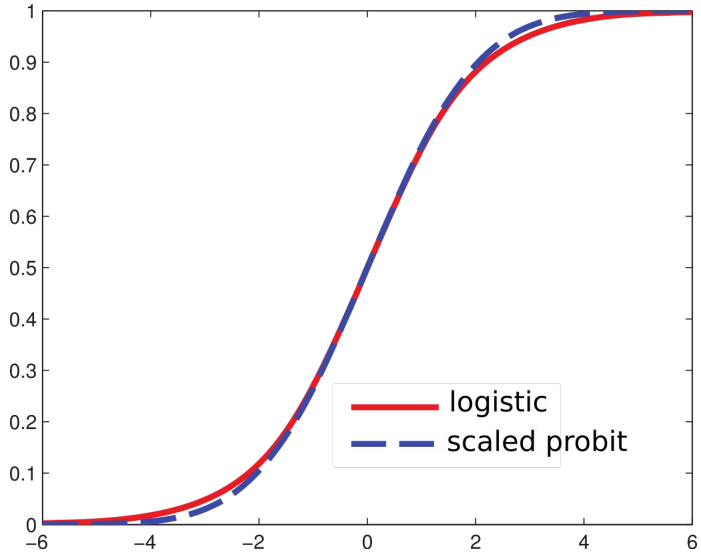
$$\sigma(x) \approx \Phi(\lambda x), \qquad \text{where } \lambda^2 = \pi / 8.$$

Note that

$$\int \Phi(\lambda x)\mathcal{N}(x|m, v)dx = \Phi(m/\sqrt{v + \lambda^{-2}})$$

Using $\Phi(\lambda x) \approx \sigma(x)$ on both sides above, we obtain

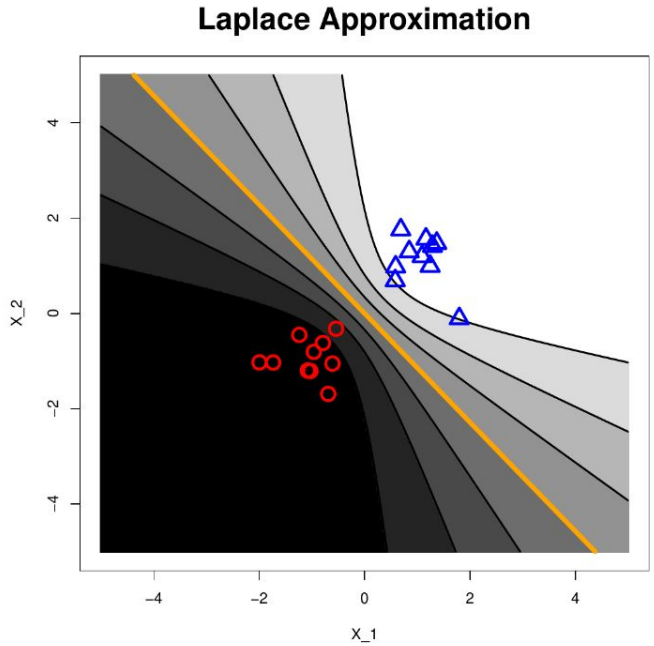
$$\int \sigma(x)\mathcal{N}(x|m, v)dx \approx \sigma(m/\sqrt{1 + \tfrac{\pi}{8}v})$$

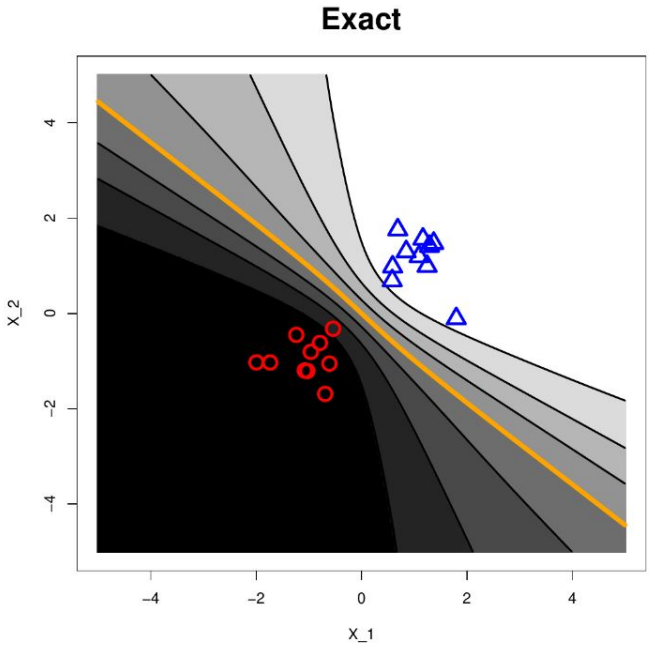


D. J. MacKay. The evidence framework applied to classification networks. Neural computation, 4(5), 720-736, 1992

Results in logistic regression example

$$p(y_\star|\boldsymbol{x}_\star, \boldsymbol{y}) \approx \int \sigma(y_\star \boldsymbol{w}^\mathsf{T} \boldsymbol{x}_\star) \mathcal{N}(\boldsymbol{w}|\boldsymbol{w}_{\mathrm{MAP}}, \boldsymbol{A}^{-1}) d\boldsymbol{w} \approx \sigma\left(\frac{y_\star \boldsymbol{w}_{\mathrm{MAP}}^\mathsf{T} \boldsymbol{x}_\star}{\sqrt{1 + \frac{\pi}{8} \boldsymbol{x}_\star^\mathsf{T} \boldsymbol{A}^{-1} \boldsymbol{x}_\star}}\right) .$$



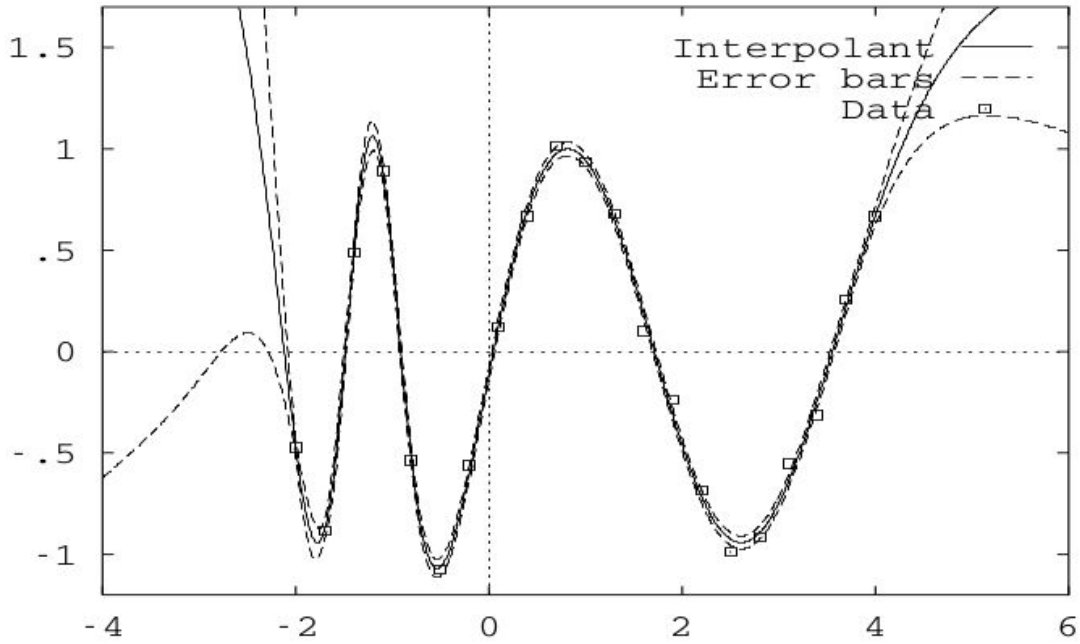


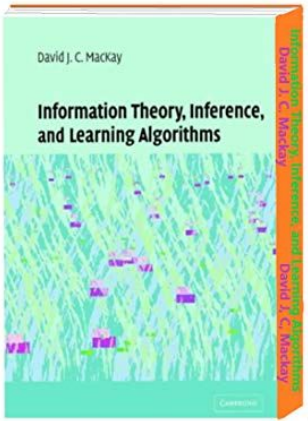
Section 3

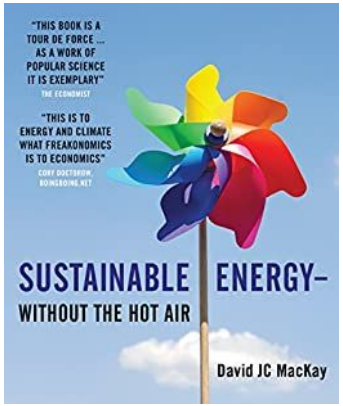
The Laplace approximation in Bayesian neural networks

Bayesian methods for adaptive models

David Mackay. PhD thesis. California Institute of Technology, 1992.







Posterior distribution in Bayesian neural networks

Regression problem with $\mathbf{y} \equiv (y_1, \ldots, y_N)^{\mathrm{T}}$, $\mathbf{X} \equiv (\mathbf{x}_1^{\mathrm{T}}, \ldots, \mathbf{x}_N^{\mathrm{T}})^{\mathrm{T}}$, $\mathbf{x}_n \in \mathbb{R}^d$, $y_n \in \mathbb{R}$.

Let $\mathbf{w} \equiv (w_1, \ldots, w_d)^{\mathrm{T}}$ be the weights of a neural network with output $f(\mathbf{x}; \mathbf{w})$ for input $\mathbf{x}$.

The likelihood function is $p(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \prod_{n=1}^{N} p(y_n|\mathbf{x}_n, \mathbf{w})$ where

$$p(y_n|\mathbf{x}_n, \mathbf{w}) = \left(\frac{\beta}{2\pi}\right)^{\frac{1}{2}} \exp\left\{-\frac{\beta}{2}(y_n - f(\mathbf{x}_n; \mathbf{w}))^2\right\}$$

and **β** is the **noise precision**. The prior on $\mathbf{w}$ is zero-mean Gaussian and ***α*** is the **prior precision**:

$$p(\mathbf{w}) = \left(\frac{\alpha}{2\pi}\right)^{\frac{d}{2}} \exp\left\{-\frac{\alpha}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}\right\}$$

The **log posterior** is then

$$\log p(\mathbf{w}|\mathbf{y}, \mathbf{X}) = -\frac{\beta}{2}\sum_{n=1}^{N}\{y_n - f(\mathbf{x}_n; \mathbf{w})\}^2 - \frac{\alpha}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} + \text{const}$$

The Laplace approximation in Bayesian neural networks

Let $\mathbf{w}_{\mathrm{MAP}}$ be the ***maximum a posteriori*** (MAP) estimate of the network weights. Then

$$p(\mathbf{w}|\mathbf{y},\mathbf{X}) \approx q(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{w}_{\mathrm{MAP}}, \mathbf{A}^{-1})$$

where

$$\begin{aligned}\mathbf{A} &= \nabla\nabla - \log p(\mathbf{w}|\mathbf{y},\mathbf{X})|_{\mathbf{w}_{\mathrm{MAP}}} \\ &= \beta\nabla\nabla\tfrac{1}{2}\textstyle\sum_{n=1}^{N}\{y_n - f(\mathbf{x}_n;\mathbf{w})\}^2\Big|_{\mathbf{w}_{\mathrm{MAP}}} + \alpha\mathbf{I}\end{aligned}$$

and the approximation to the logarithm of the **model evidence** is given by

$$\begin{aligned}\log p(\mathbf{y}|\mathbf{X}) \approx &-\frac{\beta}{2}\sum_{n=1}^{N}\{y_n - f(\mathbf{x}_n;\mathbf{w}_{\mathrm{MAP}})\}^2 + \frac{N}{2}\log\beta \\ &- \frac{\alpha}{2}\mathbf{w}_{\mathrm{MAP}}^{\mathrm{T}}\mathbf{w}_{\mathrm{MAP}} + \frac{d}{2}\log\alpha - \frac{1}{2}\log|\mathbf{A}| + \mathrm{const}\end{aligned}$$

The Laplace approximation in Bayesian neural networks

Let $\mathbf{w}_{\mathrm{MAP}}$ be the ***maximum a posteriori*** (MAP) estimate of the network weights. Then

$$p(\mathbf{w}|\mathbf{y},\mathbf{X}) \approx q(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{w}_{\mathrm{MAP}}, \mathbf{A}^{-1})$$

where

$$\mathbf{A} = \nabla\nabla - \log p(\mathbf{w}|\mathbf{y},\mathbf{X})|_{\mathbf{w}_{\mathrm{MAP}}}$$
$$= \beta\nabla\nabla\tfrac{1}{2}\textstyle\sum_{n=1}^{N}\{y_n - f(\mathbf{x}_n;\mathbf{w})\}^2\Big|_{\mathbf{w}_{\mathrm{MAP}}} + \alpha\mathbf{I}$$

Favours fit to the data

and the approximation to the logarithm of the **model evidence** is given by

$$\log p(\mathbf{y}|\mathbf{X}) \approx -\frac{\beta}{2}\sum_{n=1}^{N}\{y_n - f(\mathbf{x}_n;\mathbf{w}_{\mathrm{MAP}})\}^2 + \frac{N}{2}\log\beta$$
$$-\frac{\alpha}{2}\mathbf{w}_{\mathrm{MAP}}^{\mathrm{T}}\mathbf{w}_{\mathrm{MAP}} + \frac{d}{2}\log\alpha - \frac{1}{2}\log|\mathbf{A}| + \mathrm{const}$$

The Laplace approximation in Bayesian neural networks

Let $\mathbf{w}_{\mathrm{MAP}}$ be the ***maximum a posteriori*** (MAP) estimate of the network weights. Then

$$p(\mathbf{w}|\mathbf{y},\mathbf{X}) \approx q(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{w}_{\mathrm{MAP}}, \mathbf{A}^{-1})$$

where

$$\begin{aligned}\mathbf{A} &= \nabla\nabla - \log p(\mathbf{w}|\mathbf{y},\mathbf{X})|_{\mathbf{w}_{\mathrm{MAP}}} \\ &= \beta\nabla\nabla\tfrac{1}{2}\textstyle\sum_{n=1}^{N}\{y_n - f(\mathbf{x}_n;\mathbf{w})\}^2\Big|_{\mathbf{w}_{\mathrm{MAP}}} + \alpha\mathbf{I}\end{aligned}$$

Favours fit to the data

Penalizes model complexity

and the approximation to the logarithm of the **model evidence** is given by

$$\log p(\mathbf{y}|\mathbf{X}) \approx -\frac{\beta}{2}\sum_{n=1}^{N}\{y_n - f(\mathbf{x}_n;\mathbf{w}_{\mathrm{MAP}})\}^2 + \frac{N}{2}\log\beta$$

$$-\frac{\alpha}{2}\mathbf{w}_{\mathrm{MAP}}^{\mathrm{T}}\mathbf{w}_{\mathrm{MAP}} + \frac{d}{2}\log\alpha - \frac{1}{2}\log|\mathbf{A}| + \mathrm{const}$$

Approximating the predictive distribution

We can approximate the network function by its **linear expansion** around $\mathbf{w}_{\mathrm{MAP}}$.

$$f(\mathbf{x};\mathbf{w}) \approx f(\mathbf{x};\mathbf{w}_{\mathrm{MAP}}) + \nabla f(\mathbf{x};\mathbf{w})\big|^{\mathrm{T}}_{\mathbf{w}_{\mathrm{MAP}}} (\mathbf{w}-\mathbf{w}_{\mathrm{MAP}}) \equiv \hat{f}(\mathbf{x};\mathbf{w})$$

The **predictive distribution** can then be approximated as

$$\begin{aligned} p(y_\star|\mathbf{x}_\star,\mathbf{y},\mathbf{X}) &= \int p(y_\star|\mathbf{x}_\star,\mathbf{w}) p(\mathbf{w}|\mathbf{y},\mathbf{X})\, d\mathbf{w} \\ &\approx \int \mathcal{N}(y_\star|f(\mathbf{x}_\star;\mathbf{w}),\beta^{-1})\mathcal{N}(\mathbf{w}|\mathbf{w}_{\mathrm{MAP}},\mathbf{A}^{-1})\, d\mathbf{w} \\ &\approx \int \mathcal{N}(y_\star|\hat{f}(\mathbf{x}_\star;\mathbf{w}),\beta^{-1})\mathcal{N}(\mathbf{w}|\mathbf{w}_{\mathrm{MAP}},\mathbf{A}^{-1})\, d\mathbf{w} = \mathcal{N}(y_\star|m_\star,v_\star) \end{aligned}$$

where

$$m_\star = f(\mathbf{x}_\star;\mathbf{w}_{\mathrm{MAP}})$$

$$v_\star = \tfrac{1}{\beta} + \nabla f(\mathbf{x}_\star;\mathbf{w})\big|^{\mathrm{T}}_{\mathbf{w}_{\mathrm{MAP}}} \mathbf{A}^{-1} \nabla f(\mathbf{x}_\star;\mathbf{w})\big|_{\mathbf{w}_{\mathrm{MAP}}}$$

Approximating the predictive distribution

We can approximate the network function by its **linear expansion** around $\mathbf{w}_{\mathrm{MAP}}$.

$$f(\mathbf{x};\mathbf{w}) \approx f(\mathbf{x};\mathbf{w}_{\mathrm{MAP}}) + \nabla f(\mathbf{x};\mathbf{w})\big|^{\mathrm{T}}_{\mathbf{w}_{\mathrm{MAP}}} (\mathbf{w}-\mathbf{w}_{\mathrm{MAP}}) \equiv \hat{f}(\mathbf{x};\mathbf{w})$$

The **predictive distribution** can then be approximated as

$$\begin{aligned} p(y_\star|\mathbf{x}_\star,\mathbf{y},\mathbf{X}) &= \int p(y_\star|\mathbf{x}_\star,\mathbf{w}) p(\mathbf{w}|\mathbf{y},\mathbf{X})\, d\mathbf{w} \\ &\approx \int \mathcal{N}(y_\star|f(\mathbf{x}_\star;\mathbf{w}),\beta^{-1})\mathcal{N}(\mathbf{w}|\mathbf{w}_{\mathrm{MAP}},\mathbf{A}^{-1})\, d\mathbf{w} \\ &\approx \int \mathcal{N}(y_\star|\hat{f}(\mathbf{x}_\star;\mathbf{w}),\beta^{-1})\mathcal{N}(\mathbf{w}|\mathbf{w}_{\mathrm{MAP}},\mathbf{A}^{-1})\, d\mathbf{w} = \mathcal{N}(y_\star|m_\star,v_\star) \end{aligned}$$

where

$$m_\star = f(\mathbf{x}_\star;\mathbf{w}_{\mathrm{MAP}})$$

$$v_\star = \tfrac{1}{\beta} + \nabla f(\mathbf{x}_\star;\mathbf{w})\big|^{\mathrm{T}}_{\mathbf{w}_{\mathrm{MAP}}} \mathbf{A}^{-1} \nabla f(\mathbf{x}_\star;\mathbf{w})\big|_{\mathbf{w}_{\mathrm{MAP}}}$$

- **Mean is equal to prediction of MAP solution!**

Approximating the predictive distribution

We can approximate the network function by its **linear expansion** around $\mathbf{w}_{\mathrm{MAP}}$.

$$f(\mathbf{x};\mathbf{w}) \approx f(\mathbf{x};\mathbf{w}_{\mathrm{MAP}}) + \nabla f(\mathbf{x};\mathbf{w})\big|^{\mathrm{T}}_{\mathbf{w}_{\mathrm{MAP}}} (\mathbf{w}-\mathbf{w}_{\mathrm{MAP}}) \equiv \hat{f}(\mathbf{x};\mathbf{w})$$

The **predictive distribution** can then be approximated as

$$\begin{aligned} p(y_\star|\mathbf{x}_\star,\mathbf{y},\mathbf{X}) &= \int p(y_\star|\mathbf{x}_\star,\mathbf{w}) p(\mathbf{w}|\mathbf{y},\mathbf{X})\, d\mathbf{w} \\ &\approx \int \mathcal{N}(y_\star|f(\mathbf{x}_\star;\mathbf{w}),\beta^{-1})\mathcal{N}(\mathbf{w}|\mathbf{w}_{\mathrm{MAP}},\mathbf{A}^{-1})\, d\mathbf{w} \\ &\approx \int \mathcal{N}(y_\star|\hat{f}(\mathbf{x}_\star;\mathbf{w}),\beta^{-1})\mathcal{N}(\mathbf{w}|\mathbf{w}_{\mathrm{MAP}},\mathbf{A}^{-1})\, d\mathbf{w} = \mathcal{N}(y_\star|m_\star,v_\star) \end{aligned}$$

where

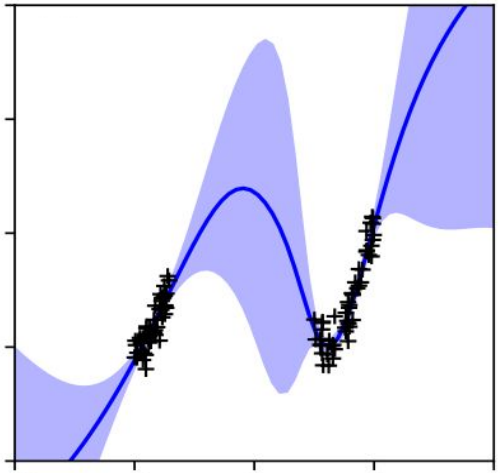
$$m_\star = f(\mathbf{x}_\star;\mathbf{w}_{\mathrm{MAP}})$$

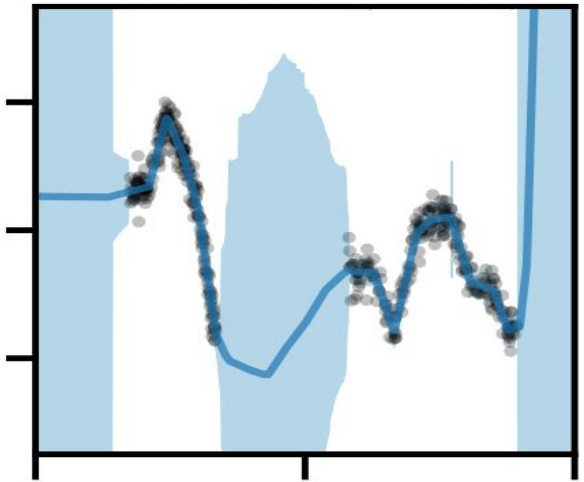
$$v_\star = \tfrac{1}{\beta} + \nabla f(\mathbf{x}_\star;\mathbf{w})\big|^{\mathrm{T}}_{\mathbf{w}_{\mathrm{MAP}}} \mathbf{A}^{-1} \nabla f(\mathbf{x}_\star;\mathbf{w})\big|_{\mathbf{w}_{\mathrm{MAP}}}$$

- **Mean is equal to prediction of MAP solution!**
- **Uncertainty high when gradient aligns with directions of low curvature**

Predictive distribution examples

1D toy regression problems. **Left**, 1 hidden layer, 50 neurls, tanh non-linearities. **Right**, 2 hidden layers, 50 neurons per layer and ReLU non-linearities.





Predictive uncertainty is **high** in regions with a lack of training data.

Why use a linearized model for prediction?

We could approximate the predictive distribution by **sampling** from the Gaussian approximation using the original neural network for predictions **without linearization**.

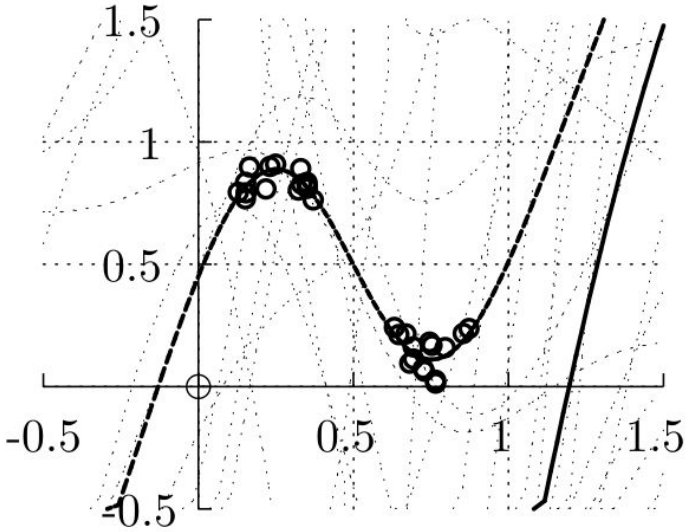
Neil D. Lawrence. Variational Inference in Probabilistic Models. PhD thesis, Cambridge, 2000.

Samples from Gaussian approximation

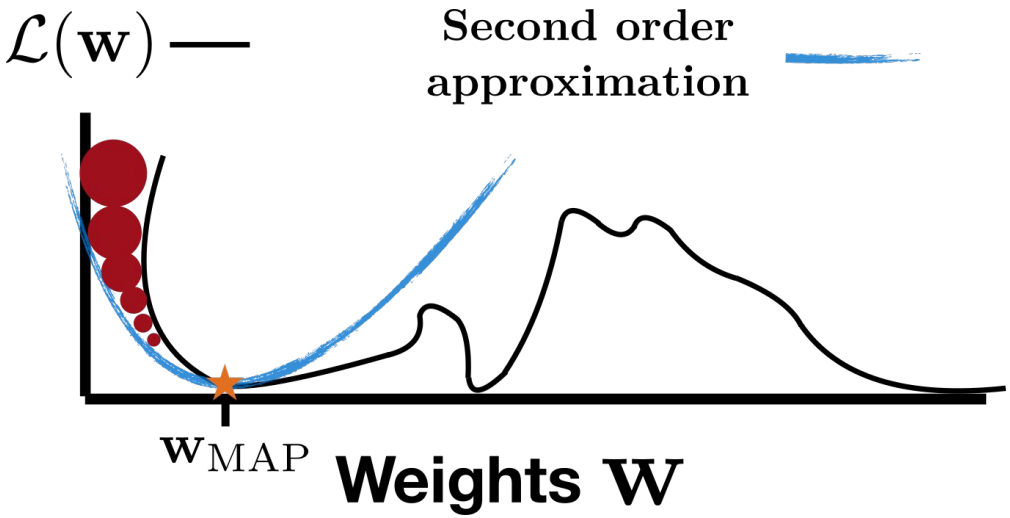
Average network output under generated samples, **most of which is off scale**

Output from network using $\mathbf{w}_{\mathrm{MAP}}$.

The linearized model approximation is critical for good performance in practice!



Some intuition on what is going on



Why are the predictions with the linearized model so good?

The linearized predictions do not change so rapidly as in original non-linear model.

More on this later!

A lot of mass from the Gaussian approximation falls into **regions of low posterior density**.

Samples from these regions will result in **very poor predictions** with the original non-linear model.

Figure source: Javier Antoran.

Tuning the prior precision $\boldsymbol{\alpha}$

Recall that $$\log p(\mathbf{y}|\mathbf{X}) \approx -\frac{\beta}{2}\sum_{n=1}^{N}\{y_n - f(\mathbf{x}_n;\mathbf{w}_{\mathrm{MAP}})\}^2 + \frac{N}{2}\log\beta - \frac{\alpha}{2}\mathbf{w}_{\mathrm{MAP}}^{\mathrm{T}}\mathbf{w}_{\mathrm{MAP}} + \frac{d}{2}\log\alpha - \frac{1}{2}\log|\mathbf{A}| + \mathrm{const}$$

We can optimize $\log p(\mathbf{y}|\mathbf{X})$ with respect to $\boldsymbol{\alpha}$ by noting that

$$\frac{d}{d\alpha}\log|\mathbf{A}| = \frac{d}{d\alpha}\sum_{i=1}^{d}\log(\lambda_i+\alpha) = \sum_{i=1}^{d}\frac{1}{\lambda_i+\alpha} = \mathrm{Trace}(\mathbf{A}^{-1})$$

We have assumed that the eigenvalues λ_i do not depend on $\boldsymbol{\alpha}$. $\log p(\mathbf{y}|\mathbf{X})$ is maximized when

$$\alpha = \frac{\gamma}{\mathbf{w}_{\mathrm{MAP}}^{\mathrm{T}}\mathbf{w}_{\mathrm{MAP}}} \quad \text{where} \quad \gamma \equiv \sum_{i=1}^{d}\frac{\lambda_i}{\lambda_i+\alpha} \quad \text{and we have assumed} \quad \frac{d}{d\alpha}\mathbf{w}_{\mathrm{MAP}} = 0 .$$

Tuning the prior precision $\boldsymbol{\alpha}$

Eigenvalues of

$$\mathbf{A} = \beta \nabla\nabla \tfrac{1}{2} \textstyle\sum_{n=1}^{N} \{y_n - f(\mathbf{x}_n; \mathbf{w})\}^2 \Big|_{\mathbf{w}_{\mathrm{MAP}}} + \alpha \mathbf{I}$$

Recall that $\log p(\mathbf{y}|\mathbf{X}) \approx -\frac{\beta}{2} \sum_{n=1}^{N} \{y_n - f(\mathbf{x}_n; \mathbf{w}_{\mathrm{MAP}})\}^2 + \frac{N}{2} \log \beta$

$$- \frac{\alpha}{2} \mathbf{w}_{\mathrm{MAP}}^{\mathrm{T}} \mathbf{w}_{\mathrm{MAP}} + \frac{d}{2} \log \alpha - \frac{1}{2} \log |\mathbf{A}| + \mathrm{const}$$

We can optimize $\log p(\mathbf{y}|\mathbf{X})$ with respect to $\boldsymbol{\alpha}$ by noting that

$$\frac{d}{d\alpha} \log |\mathbf{A}| = \frac{d}{d\alpha} \sum_{i=1}^{d} \log(\lambda_i + \alpha) = \sum_{i=1}^{d} \frac{1}{\lambda_i + \alpha} = \mathrm{Trace}(\mathbf{A}^{-1})$$

We have assumed that the eigenvalues λ_i do not depend on $\boldsymbol{\alpha}$. $\log p(\mathbf{y}|\mathbf{X})$ is maximized when

$$\alpha = \frac{\gamma}{\mathbf{w}_{\mathrm{MAP}}^{\mathrm{T}} \mathbf{w}_{\mathrm{MAP}}} \quad \text{where} \quad \gamma \equiv \sum_{i=1}^{d} \frac{\lambda_i}{\lambda_i + \alpha} \quad \text{and we have assumed} \quad \frac{d}{d\alpha} \mathbf{w}_{\mathrm{MAP}} = 0 .$$

Tuning the prior precision $\boldsymbol{\alpha}$

Eigenvalues of

$$\mathbf{A} = \beta \nabla\nabla \tfrac{1}{2}\textstyle\sum_{n=1}^{N}\{y_n - f(\mathbf{x}_n;\mathbf{w})\}^2\Big|_{\mathbf{w}_{\mathrm{MAP}}} + \alpha\mathbf{I}$$

Recall that $\log p(\mathbf{y}|\mathbf{X}) \approx -\frac{\beta}{2}\sum_{n=1}^{N}\{y_n - f(\mathbf{x}_n;\mathbf{w}_{\mathrm{MAP}})\}^2 + \frac{N}{2}\log\beta$

$$- \frac{\alpha}{2}\mathbf{w}_{\mathrm{MAP}}^{\mathrm{T}}\mathbf{w}_{\mathrm{MAP}} + \frac{d}{2}\log\alpha - \frac{1}{2}\log|\mathbf{A}| + \mathrm{const}$$

Effective number of well-determined parameters

We can optimize $\log p(\mathbf{y}|\mathbf{X})$ with respect to $\boldsymbol{\alpha}$ by noting that

$$\frac{d}{d\alpha}\log|\mathbf{A}| = \frac{d}{d\alpha}\sum_{i=1}^{d}\log(\lambda_i + \alpha) = \sum_{i=1}^{d}\frac{1}{\lambda_i + \alpha} = \mathrm{Trace}(\mathbf{A}^{-1})$$

We have assumed that the eigenvalues λ_i do not depend on $\boldsymbol{\alpha}$. $\log p(\mathbf{y}|\mathbf{X})$ is maximized when

$$\alpha = \frac{\gamma}{\mathbf{w}_{\mathrm{MAP}}^{\mathrm{T}}\mathbf{w}_{\mathrm{MAP}}} \quad \text{where} \quad \gamma \equiv \sum_{i=1}^{d}\frac{\lambda_i}{\lambda_i + \alpha} \quad \text{and we have assumed} \quad \frac{d}{d\alpha}\mathbf{w}_{\mathrm{MAP}} = 0$$

Tuning the noise precision β

Recall that $\log p(\mathbf{y}|\mathbf{X}) \approx -\frac{\beta}{2}\sum_{n=1}^{N}\{y_n - f(\mathbf{x}_n;\mathbf{w}_{\mathrm{MAP}})\}^2 + \frac{N}{2}\log\beta$

$$- \frac{\alpha}{2}\mathbf{w}_{\mathrm{MAP}}^{\mathrm{T}}\mathbf{w}_{\mathrm{MAP}} + \frac{d}{2}\log\alpha - \frac{1}{2}\log|\mathbf{A}| + \mathrm{const}$$

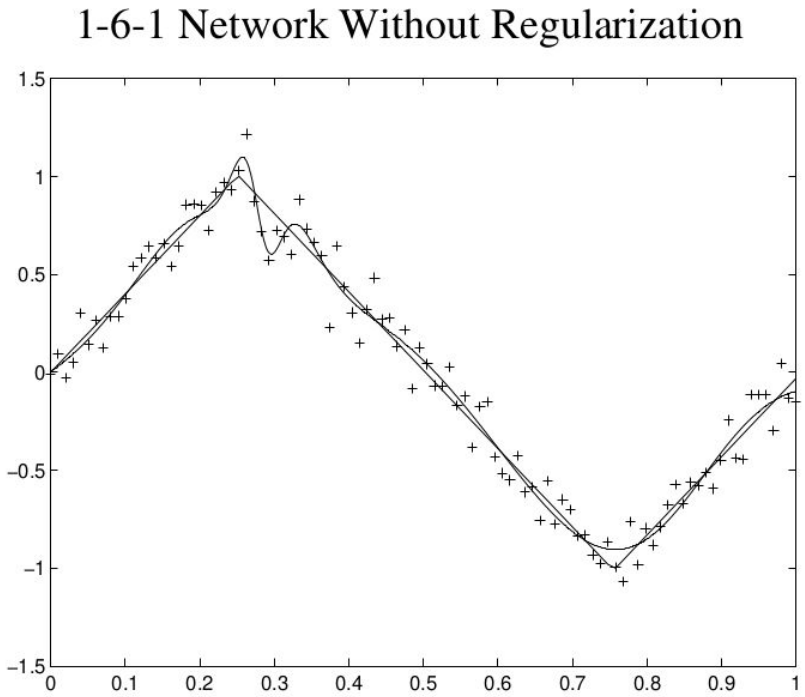
We can optimize $\log p(\mathbf{y}|\mathbf{X})$ w.r.t. **β** by noting that the eigenvalues λ_i are proportional to **β**, hence

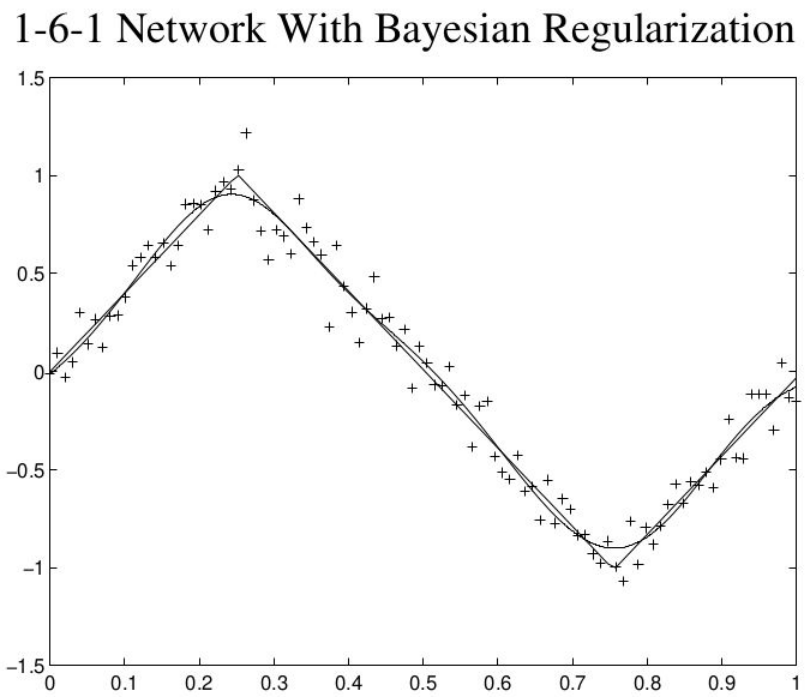
$$\frac{d\lambda_i}{d\beta} = \frac{\lambda_i}{\beta} \qquad \text{and thus} \qquad \frac{d}{d\beta}\log|\mathbf{A}| = \frac{d}{d\beta}\sum_{i=1}^{d}\log(\lambda_i+\alpha) = \frac{1}{\beta}\sum_{i=1}^{d}\frac{\lambda_i}{\lambda_i+\alpha}$$

$\log p(\mathbf{y}|\mathbf{X})$ is maximized when

$$\beta = \frac{N-\gamma}{\sum_{n=1}^{N}\{y_n - f(\mathbf{x}_n;\mathbf{w}_{\mathrm{MAP}})\}^2}$$

Example of hyperparameter tuning for regression





F. D. Foresee and T. H. Martin. Gauss-Newton approximation to Bayesian learning. In ICNN, 1997.

Generalized Gauss-Newton approximation

The Hessian $\mathbf{A} = \nabla\nabla - \log p(\mathbf{w}|\mathbf{y}, \mathbf{X})|_{\mathbf{w}_{\mathrm{MAP}}}$ **may not be positive definite!** What can we do?

We can again approximate the network function by its linear expansion:

$$f(\mathbf{x}; \mathbf{w}) \approx f(\mathbf{x}; \mathbf{w}_{\mathrm{MAP}}) + \nabla f(\mathbf{x}; \mathbf{w})|^{\mathrm{T}}_{\mathbf{w}_{\mathrm{MAP}}} (\mathbf{w} - \mathbf{w}_{\mathrm{MAP}}) \equiv \hat{f}(\mathbf{x}; \mathbf{w})$$

Using this in the Hessian equation gives the **generalized Gauss-Newton (GGN) approximation**:

$$\mathbf{A} \approx \beta \nabla\nabla \tfrac{1}{2} \textstyle\sum_{n=1}^{N} \{y_n - \hat{f}(\mathbf{x}_n; \mathbf{w})\}^2 \Big|_{\mathbf{w}_{\mathrm{MAP}}} + \alpha \mathbf{I} = \boxed{\textstyle\sum_{n=1}^{N} \mathbf{J}_n^{\mathrm{T}} \mathbf{G}_n \mathbf{J}_n + \alpha \mathbf{I}}$$

where $\mathbf{J}_n$ is the *n_outputs* × *d* Jacobian matrix given by $\mathbf{J}_n = \nabla f(\mathbf{x}_n; \mathbf{w})|_{\mathbf{W}_{\mathrm{MAP}}}$ and $\mathbf{G}_n$ is the *n_outputs* × *n_outputs* matrix $\mathbf{G}_n = \nabla\nabla - \log p(y_n|\hat{f}(\mathbf{x}_n; \mathbf{w}_{\mathrm{MAP}}))\Big|_{\hat{f}(\mathbf{x}_n; \mathbf{w}_{\mathrm{MAP}})} = \beta$

The GGN is **guaranteed to be positive definite** when $-\log p(y|f(\mathbf{x}; \mathbf{w}))$ s convex in $f(\mathbf{x}; \mathbf{w})$.

J. Martens and I. Sutskever, Learning recurrent neural networks with hessian-free optimization. In ICML, 2011,
F. D. Foresee and T. H. Martin. Gauss-Newton approximation to Bayesian learning. In ICNN, 1997.

Practical implementation

1. Choose initial values for the hyperparameters $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and network weights $\mathbf{w}$
2. Train the network using a standard optimization algorithm to obtain $\mathbf{w}_{\mathrm{MAP}}$
3. Every few cycles of optimization:
   a. Update $\mathbf{A} = \mathbf{J}^{\mathrm{T}}\mathbf{G}\mathbf{J} + \alpha\mathbf{I}$ given current estimate value of $\mathbf{w}$
   b. Compute $\boldsymbol{\gamma}$ using eigen-decomposition of **A**, that is, $\gamma = \sum_{i=1}^{d} \lambda_i/(\lambda_i + \alpha)$
   c. Update $\boldsymbol{\alpha}$ using $\alpha = \gamma/\mathbf{w}^{\mathrm{T}}\mathbf{w}$
   d. Update $\boldsymbol{\beta}$ using $\beta = (N - \gamma)/\sum_{n=1}^{N}\{y_n - f(\mathbf{x}_n; \mathbf{w})\}^2$
4. Iterate 2 and 3 until convergence

If final $\boldsymbol{\gamma}$ is close to d, then network may not be large enough: add more neurons.

If larger network has same final $\boldsymbol{\gamma}$, then the smaller network was enough.

Classification problems

For C classes, the network has C outputs, $\mathbf{y}_n$ is a **one-hot-encoding** vector and

$$p(\mathbf{y}_n|\mathbf{f}(\mathbf{x}_n;\mathbf{w})) = \text{Categorical}\left[\mathbf{y}_n\,;\text{Softmax}\{\mathbf{f}(\mathbf{x}_n;\mathbf{w})\}\right]$$

where

$$\text{Categorical}\left[\mathbf{y}_n|\mathbf{p}\right] = \sum_{i=1}^{C} p_i y_{n,i} \quad \text{and} \quad \left[\text{Softmax}\{\mathbf{f}(\mathbf{x}_n;\mathbf{w})\}\right]_i = \frac{\exp\{f_i(\mathbf{x}_n;\mathbf{w})\}}{\sum_{i=1}^{C}\exp\{f_i(\mathbf{x}_n;\mathbf{w})\}}$$

Similarly as in the **probit approximation** of the logistic function, we can use

$$p(\mathbf{y}_\star|\mathbf{x}_\star,\mathbf{Y},\mathbf{X}) \approx \text{Categorical}\left[\mathbf{y}_\star;\text{Softmax}\{\mathbf{f}(\mathbf{x}_\star;\mathbf{w}_{\text{MAP}})/\sqrt{1+\pi/8\text{diag}(\boldsymbol{\Sigma}(\mathbf{x}_\star))}\}\right]$$

where we use entry-wise vector operations and $\boldsymbol{\Sigma}(\mathbf{x}_\star) = \nabla\mathbf{f}(\mathbf{x}_\star;\mathbf{w})|^{\text{T}}_{\mathbf{w}_{\text{MAP}}}\mathbf{A}^{-1}\nabla\mathbf{f}(\mathbf{x}_\star;\mathbf{w})|_{\mathbf{w}_{\text{MAP}}}$

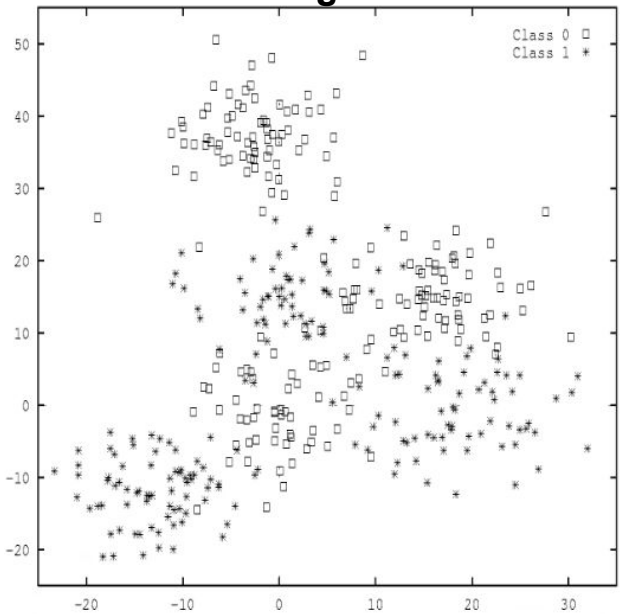
We could use sampling, but this is less expensive and works well in practice.

Gibbs, M. N. Bayesian Gaussian Processes for Regression and Classification. PhD thesis, University of Cambridge, September 1997

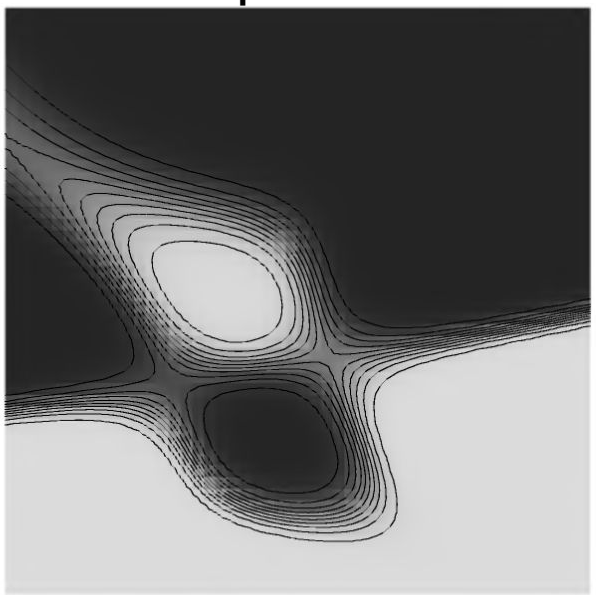
Classification example

MAP solution is **overly confident** far from data. Laplace approximation is **more conservative**.

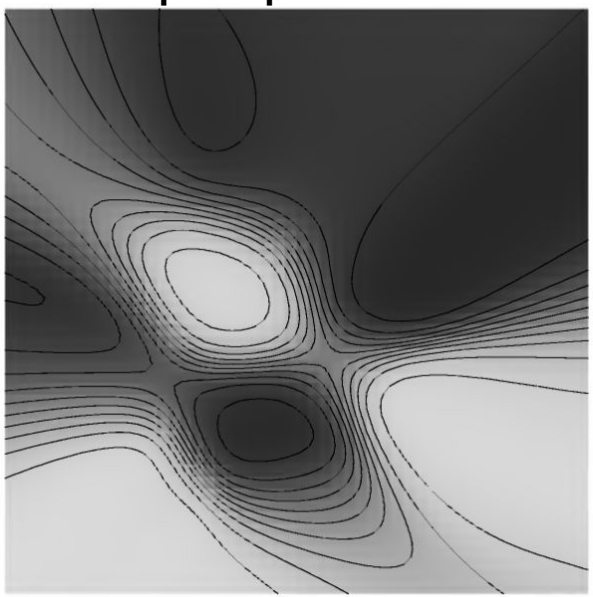
Training Data



MAP predictions



Laplace predictions



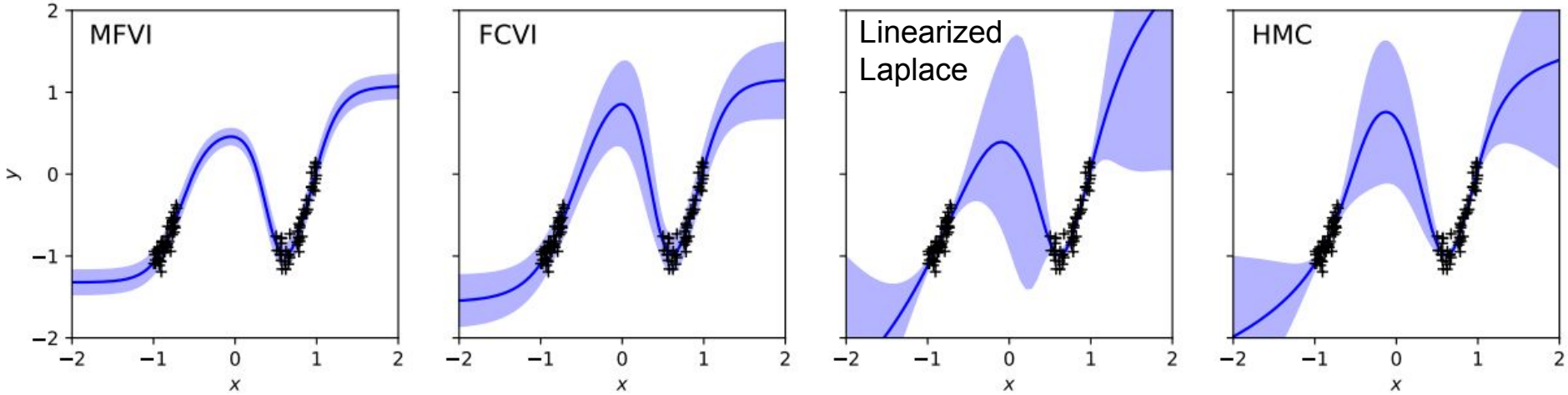
David J. C. MacKay. The evidence framework applied to classification networks. Neural computation 4.5 (1992): 720-736.

Section 4

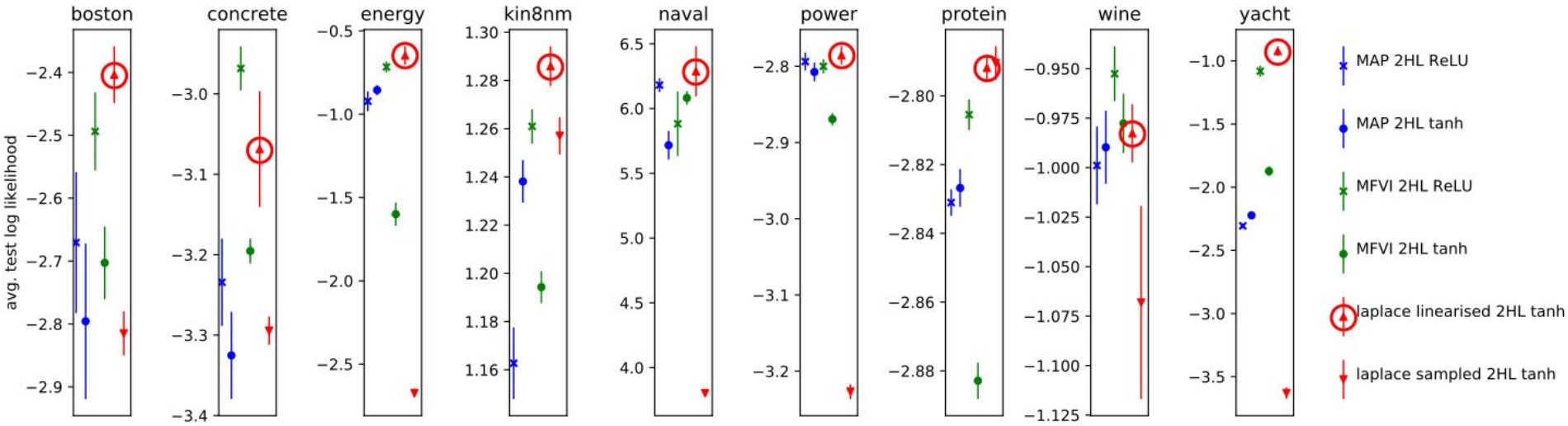
More recent works on the Laplace approximation

Comparison with baselines

Andrew Y. K. Foong, Yingzhen Li, José Miguel Hernández-Lobato, Richard E. Turner. 'In-Between' Uncertainty in Bayesian Neural Networks. arXiv:1906.11537, 2019

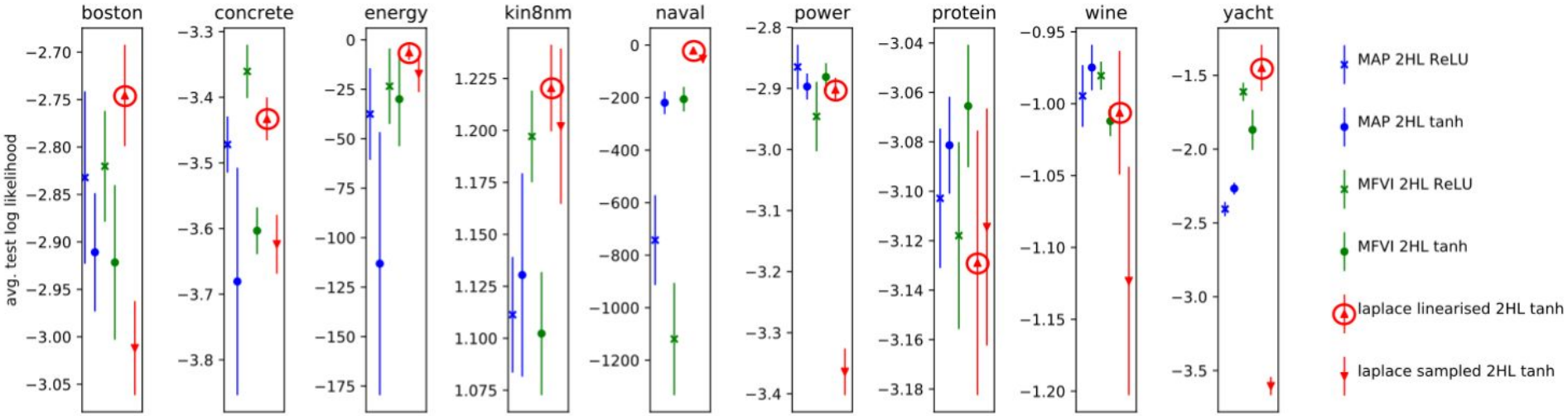


Results on standard splits of UCI datasets



Linearized Laplace performs very well on the standard splits, while **sampled Laplace does not**.

Results on gap splits of UCI datasets



Linearized Laplace avoids catastrophic results on gap splits for energy and naval problems.

Scaling up with Kronecker products

The **GGN** can be approximated in a scalable way using **Kronecker products** $\otimes$:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix}, \quad \begin{aligned} &(\mathbf{A} \otimes \mathbf{B})^{-1} = \mathbf{A}^{-1} \otimes \mathbf{B}^{-1}. \quad (\mathbf{A} \otimes \mathbf{B})^{\mathsf{T}} = \mathbf{A}^{\mathsf{T}} \otimes \mathbf{B}^{\mathsf{T}}. \\ &(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = (\mathbf{AC}) \otimes (\mathbf{BD}). \\ &\mathrm{vec}(\mathbf{A}) = [a_{1,1}, \ldots, a_{m,1}, a_{1,2}, \ldots, a_{m,2}, \ldots, a_{1,n}, \ldots, a_{m,n}]^{\mathrm{T}} \end{aligned}$$

Let $\mathbf{h}_\ell$, $\mathbf{a}_\ell$ and $\mathbf{W}_\ell$ be the **pre-activations**, **activations** and **weights** for layer ℓ, $1 \leq \ell \leq L$. Then

$\mathbf{h}_\ell = \mathbf{W}_\ell \mathbf{a}_{\ell-1} = (\mathbf{a}_{\ell-1}^{\mathrm{T}} \otimes \mathbf{I})\mathrm{vec}(\mathbf{W}_\ell)$, $\mathbf{a}_\ell = f_\ell(\mathbf{h}_\ell)$ and $f_\ell(\cdot)$ is the element-wise non-linearity.

Let $\mathcal{L}_n$ be the **minus log-likelihood** for the n-th data point and let $\mathcal{G}_L^n = \dfrac{\partial^2 \mathcal{L}_n}{\partial h_i^L \partial h_j^L}$. Then

the GGN for layer ℓ is $\mathbf{G}_\ell^n = \dfrac{\partial \mathbf{h}_L}{\partial \mathrm{vec}(\mathbf{W}_\ell)}^{\mathrm{T}} \mathcal{G}_L^n \dfrac{\partial \mathbf{h}_L}{\partial \mathrm{vec}(\mathbf{W}_\ell)} = (\mathbf{a}_{\ell-1}\mathbf{a}_{\ell-1}^{\mathrm{T}}) \otimes \left(\dfrac{\partial \mathbf{h}_L}{\partial \mathbf{h}_\ell}^{\mathrm{T}} \mathcal{G}_L^n \dfrac{\partial \mathbf{h}_L}{\partial \mathbf{h}_\ell} \right)$

Botev, A. The Gauss-Newton matrix for Deep Learning models and its applications. PhD thesis, University College London, 2020.

Scaling up with Kronecker products

Pre-activation Gauss Newton matrix

$$\mathcal{G}_\ell^n$$

The **GGN** can be approximated in a scalable way using **Kronecker products** $\otimes$:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix}, \quad \begin{array}{l} (\mathbf{A} \otimes \mathbf{B})^{-1} = \mathbf{A}^{-1} \otimes \mathbf{B}^{-1}. \quad (\mathbf{A} \otimes \mathbf{B})^\mathsf{T} = \mathbf{A}^\mathsf{T} \otimes \mathbf{B}^\mathsf{T}. \\ (\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = (\mathbf{AC}) \otimes (\mathbf{BD}). \\ \mathrm{vec}(\mathbf{A}) = [a_{1,1}, \dots, a_{m,1}, a_{1,2}, \dots, a_{m,2}, \dots, a_{1,n}, \dots, a_{m,n}]^\mathrm{T} \end{array}$$

Let $\mathbf{h}_\ell$, $\mathbf{a}_\ell$ and $\mathbf{W}_\ell$ be the **pre-activations**, **activations** and **weights** for layer ℓ, $1 \leq \ell \leq L$. Then

$\mathbf{h}_\ell = \mathbf{W}_\ell \mathbf{a}_{\ell-1} = (\mathbf{a}_{\ell-1}^\mathrm{T} \otimes \mathbf{I})\mathrm{vec}(\mathbf{W}_\ell)$, $\mathbf{a}_\ell = f_\ell(\mathbf{h}_\ell)$ and $f_\ell(\cdot)$ is the element-wise non-linearity.

Let $\mathcal{L}_n$ be the **minus log-likelihood** for the n-th data point and let $\mathcal{G}_L^n = \dfrac{\partial^2 \mathcal{L}_n}{\partial h_i^L \partial h_j^L}$. Then

the GGN for layer ℓ is $\mathbf{G}_\ell^n = \dfrac{\partial \mathbf{h}_L}{\partial \mathrm{vec}(\mathbf{W}_\ell)}^\mathrm{T} \mathcal{G}_L^n \dfrac{\partial \mathbf{h}_L}{\partial \mathrm{vec}(\mathbf{W}_\ell)} = (\mathbf{a}_{\ell-1}\mathbf{a}_{\ell-1}^\mathrm{T}) \otimes \left(\dfrac{\partial \mathbf{h}_L}{\partial \mathbf{h}_\ell}^\mathrm{T} \mathcal{G}_L^n \dfrac{\partial \mathbf{h}_L}{\partial \mathbf{h}_\ell} \right)$

Ritter, H., Botev, A., & Barber, D. A scalable laplace approximation for neural networks. In ICLR, 2018.
Botev, A. The Gauss-Newton matrix for Deep Learning models and its applications. PhD thesis, University College London, 2020.

Scaling up with Kronecker products

The **pre-activation Gauss Newton matrix** can be computed recursively using

$$\boldsymbol{\mathcal{G}}_\ell^n = \mathbf{B}_\ell \mathbf{W}_{\ell+1}^{\mathrm{T}} \boldsymbol{\mathcal{G}}_{\ell+1}^n \mathbf{W}_{\ell+1} \mathbf{B}_\ell$$

The diagonal matrices $\mathbf{B}_\ell$ are defined as $\mathbf{B}_\ell = \mathrm{diag}(f'_\ell(\mathbf{h}_\ell))$.

For the whole dataset, the GGN for layer ℓ can be approximated as

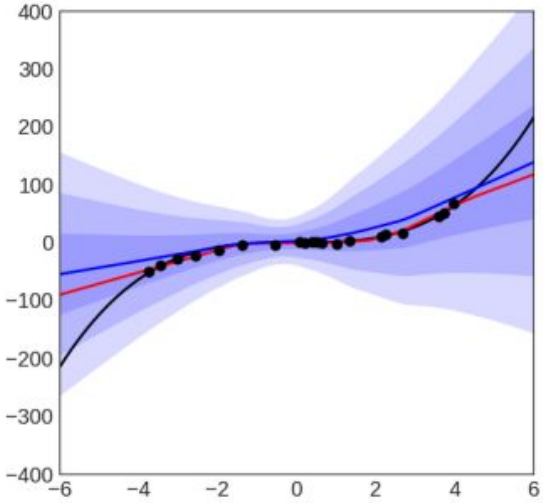
$$\mathbf{G}_\ell = \sum_{n=1}^{N} \left[\left(\mathbf{a}_{\ell-1} \mathbf{a}_{\ell-1}^{\mathrm{T}} \right) \otimes \boldsymbol{\mathcal{G}}_\ell^n \right] \approx \left(\sum_{n=1}^{N} \mathbf{a}_{\ell-1} \mathbf{a}_{\ell-1}^{\mathrm{T}} \right) \otimes \left(\sum_{n=1}^{N} \boldsymbol{\mathcal{G}}_\ell^n \right)$$

The posterior covariance $\mathbf{A}$ is then assumed **block diagonal**, one block per layer given by

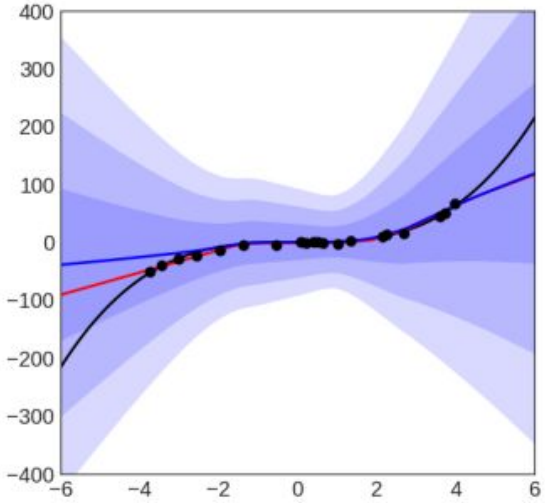
$\mathbf{A}_\ell = [\mathbf{G}_\ell + \alpha \mathbf{I}]^{-1}$ which can be computed efficiently due to the Kronecker factorization.

Immer, A., Korzepa, M., & Bauer, M. Improving predictions of Bayesian neural nets via local linearization. In AISTATS, 2021.

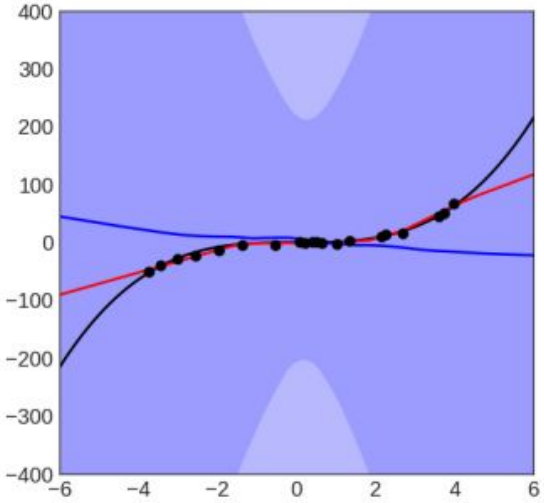
Results on toy problem



(a) KF Laplace



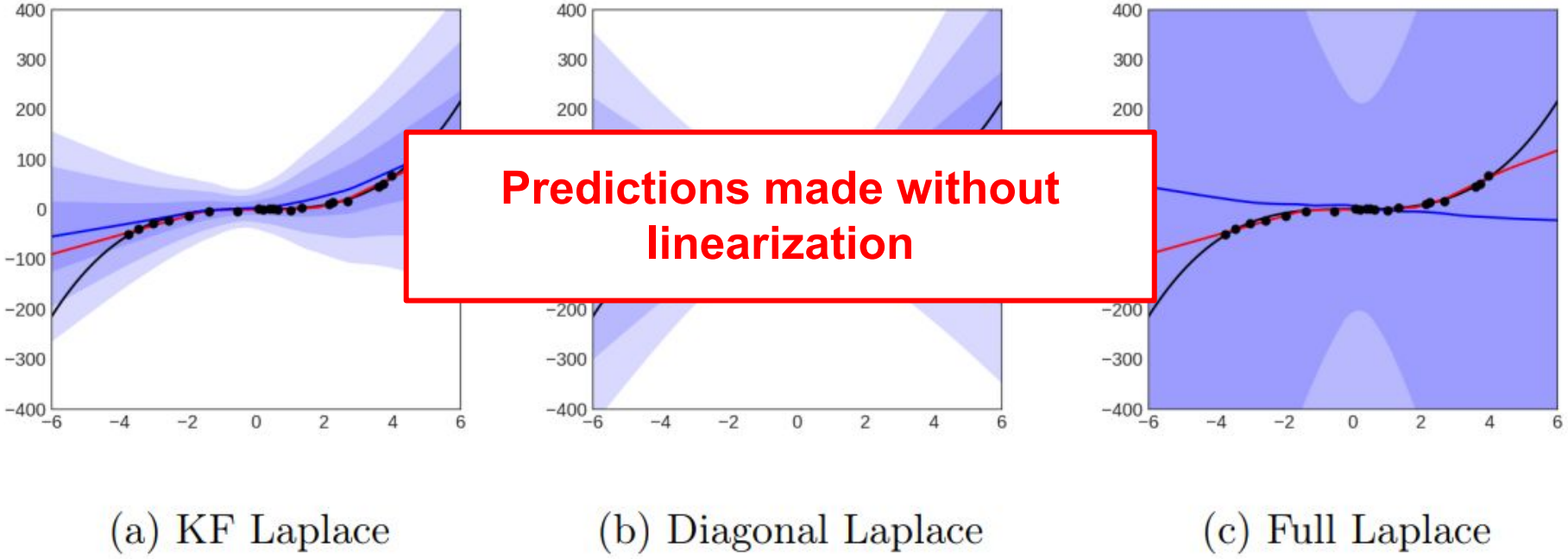
(b) Diagonal Laplace



(c) Full Laplace

Botev, A. The Gauss-Newton matrix for Deep Learning models and its applications. PhD thesis, University College London, 2020.

Results on toy problem



Botev, A. The Gauss-Newton matrix for Deep Learning models and its applications. PhD thesis, University College London, 2020.

Justification for predictions with linearized model

Immer, A., Korzepa, M., & Bauer, M. Improving predictions of Bayesian neural nets via local linearization. In AISTATS, 2021.

The **GGN** approximation to the Hessian **assumes a linearization** of the neural network.

GGN is a model approximation, independent of the inference method, e.g. Laplace or VI.

If we use the linearized model for inference, **we should also make predictions using it**.

The linearized model is $f(\mathbf{x}; \mathbf{w}) \approx f(\mathbf{x}; \mathbf{w}_{\text{MAP}}) + \nabla f(\mathbf{x}; \mathbf{w})|^{\text{T}}_{\mathbf{w}_{\text{MAP}}} (\mathbf{w} - \mathbf{w}_{\text{MAP}}) \equiv \hat{f}(\mathbf{x}; \mathbf{w})$.

The log-posterior is $\log p(\mathbf{w}|\mathcal{D}) = \sum_{n=1}^{N} \log p(\mathbf{y}_n|\hat{f}(\mathbf{x}_n; \mathbf{w})) + \log p(\mathbf{w})$.

The **predictive distribution** is approximated using sampling:

$$p(\mathbf{y}_\star|\mathbf{x}_\star, \mathcal{D}) \approx \int p(\mathbf{y}_\star|\hat{f}(\mathbf{x}_\star; \mathbf{w}))q(\mathbf{w})\, d\mathbf{w} \approx \frac{1}{K} \sum_{k=1}^{K} p(\mathbf{y}_\star|\hat{f}(\mathbf{x}_\star; \mathbf{w}_k)), \quad \mathbf{w}_k \sim q(\mathbf{w})$$

Results Fashion-MNIST and CIFAR10

FMNIST: 3 conv. layers and 3 FC layers. CIFAR10: 9 conv. blocks and average pooling.

Results using Kronecker factor approximations. **Prior variance tuned using validation set**.

Dataset	Method	Accuracy ↑	NLL ↓	ECE ↓	OOD-AUC ↑
FMNIST	MAP	$91.39_{\pm 0.11}$	$0.258_{\pm 0.004}$	$0.017_{\pm 0.001}$	$0.864_{\pm 0.014}$
	Laplace, no linearized	$84.42_{\pm 0.12}$	$0.942_{\pm 0.016}$	$0.411_{\pm 0.008}$	$0.945_{\pm 0.002}$
	Laplace, linearized	$\mathbf{92.25_{\pm 0.10}}$	$\mathbf{0.244_{\pm 0.003}}$	$0.012_{\pm 0.003}$	$\mathbf{0.955_{\pm 0.006}}$
CIFAR10	MAP	$80.92_{\pm 0.32}$	$0.605_{\pm 0.007}$	$0.066_{\pm 0.004}$	$0.792_{\pm 0.008}$
	Laplace, no linearized	$21.74_{\pm 0.80}$	$2.114_{\pm 0.021}$	$0.095_{\pm 0.012}$	$0.689_{\pm 0.020}$
	Laplace, linearized	$\mathbf{81.37_{\pm 0.15}}$	$0.601_{\pm 0.008}$	$0.084_{\pm 0.010}$	$\mathbf{0.843_{\pm 0.016}}$

Laplace, linearized improves significantly over MAP solution and non-linearized Laplace.

Tuning hyper-parameters as well

Immer, A., Bauer, M., Fortuin, V., Rätsch, G., & Emtiyaz, K. M. Scalable marginal likelihood estimation for model selection in deep learning. In ICML, 2021.

After some epochs, update layerwise precision hyperparameters by gradient based optimization of

$$\log p(\mathbf{y}, \mathbf{X}) \approx \log p(\mathbf{y}|\mathbf{X}, \mathbf{w}) + \log p(\mathbf{w}) - \frac{1}{2} \log |\mathbf{A}|$$

where $\mathbf{A}$ approximated using Kronecker products. Done every F epochs after burn-in of B epochs.

For ResNets, they use **fixup** instead of **batchnorm**.

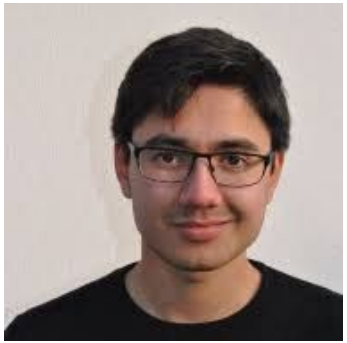
		cross-validation		KFAC marginal likelihood	
Dataset	**Model**	**accuracy**	**logLik**	**accuracy**	**logLik**
MNIST	**MLP**	98.22	−0.061	98.38	−0.053
	CNN	99.40	**−0.017**	**99.46**	**−0.016**
FMNIST	**MLP**	88.09	−0.347	89.83	−0.305
	CNN	91.39	−0.258	**92.06**	**−0.233**
CIFAR10	**CNN**	77.41	−0.680	80.46	−0.644
	ResNet	83.73	−1.060	**86.11**	−0.595

Section 5

Subnetwork inference

Daxberger E., Nalisnick E., Allingham J., Antoran J. and Hernández-Lobato J. M. Bayesian Deep Learning via Subnetwork Inference. In ICML, 2021.

Collaborators



Erik Daxberger

Eric Nalisnick

Javier Antoran

James Allingham

Motivation

Goal: Infer posterior distribution over weights $\mathbf{W}$ of deep neural network (DNN)

$$p(\mathbf{W}|\boldsymbol{y}, \boldsymbol{X}) \quad \propto \quad p(\boldsymbol{y}|\boldsymbol{X}, \mathbf{W})\, p(\mathbf{W})$$

posterior likelihood prior

Problem: Modern DNNs are too big! Even approximate inference is hard!

Existing approaches: Make strong/unrealistic assumptions on posterior,
e.g. full factorization $p(\mathbf{W}|\boldsymbol{y}, \boldsymbol{X}) \approx \prod_{d=1}^{D} q(\mathrm{w}_d)$.

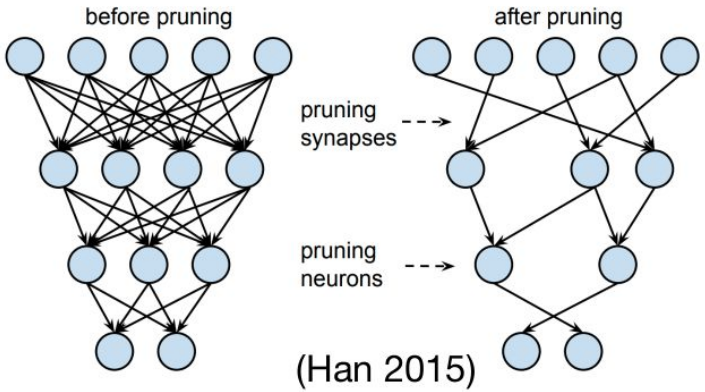
This deteriorates the quality of resulting uncertainty estimates!
(Ovadia 2019, Fort 2019, Foong 2019, Ashukha 2020)

Question: Do we really need to estimate a posterior over **ALL** the weights?

Proposed Idea

Due to overparameterization, a DNNs accuracy is well-preserved by a small subnetwork.

How to find those subnetworks? E.g. by pruning (Frankle & Carbin 2019).



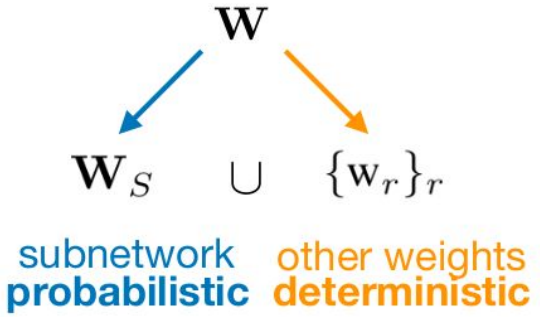
Can a full DNN's model uncertainty be well-preserved by a small subnetwork's model uncertainty?

Answer: Yes!

Subnetwork inference

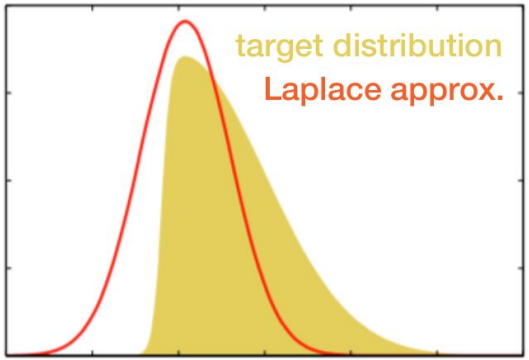
Proposed posterior approximation:

$$p(\mathbf{W}|\boldsymbol{y}, \boldsymbol{X}) \approx q(\mathbf{W}) = p(\mathbf{W}_S|\boldsymbol{y}, \boldsymbol{X}) \prod_r \delta(\mathrm{w}_r - \mathrm{w}_r^*)$$
$$\approx q(\mathbf{W}_S) \prod_r \delta(\mathrm{w}_r - \mathrm{w}_r^*)$$



1. How do we obtain the subnetwork posterior approximation $q(\mathbf{W}_S)$?
2. How do we set the fixed values $\mathrm{w}_r^* \in \mathbb{R}$ of all remaining weights $\{\mathrm{w}_r\}_r$?
3. How do we select the subnetwork weights $\mathbf{W}_S$?

Laplace Approximation



(Bishop 2006)

1. Obtain point estimate over all weights via (MAP) inference:

$$\boldsymbol{W}_{MAP} = \arg\max_{\boldsymbol{W}} \left[\log p(\mathbf{y}|\boldsymbol{X}, \boldsymbol{W}) + \log p(\boldsymbol{W})\right]$$

2. Infer full-covariance Gaussian posterior over $\mathbf{W}$:

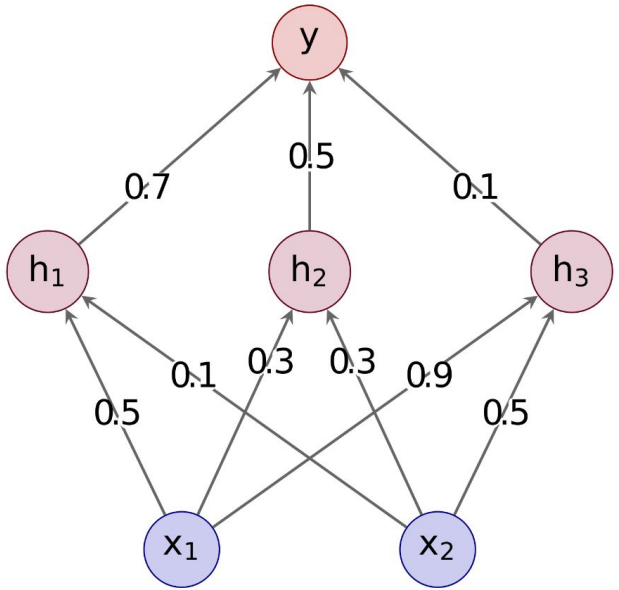
$$p(\mathbf{W}|\boldsymbol{y}, \boldsymbol{X}) \approx q(\mathbf{W}) = \mathcal{N}\left(\mathbf{W}; \boldsymbol{W}_{MAP}, H^{-1}\right)$$

$\boldsymbol{H}$ is the Gauss-Newton approximation of the **Hessian** of the loss.

- **Practically appealing due to simplicity of MAP estimation.**
- **Not scalable, as H is typically too big in large networks!**
- **Solution:** Select a subnetwork with weights $\mathbf{W}_S$ on which we apply the Laplace approximation and keep other weights equal to their MAP point estimates.

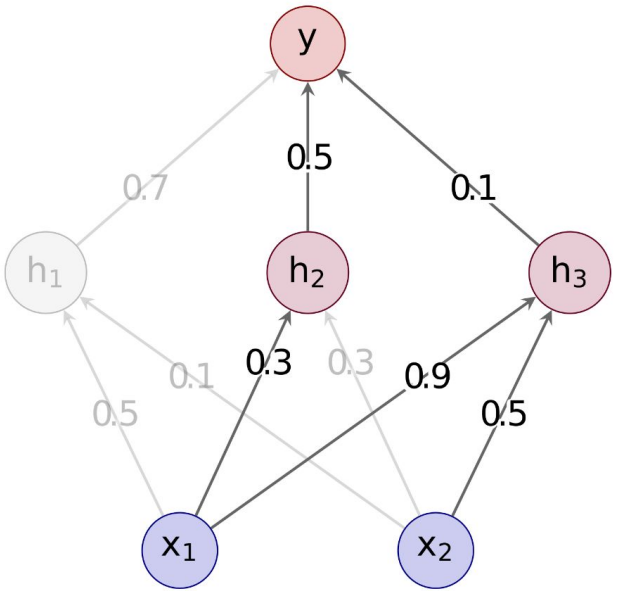
Subnetwork inference

1 MAP Estimation



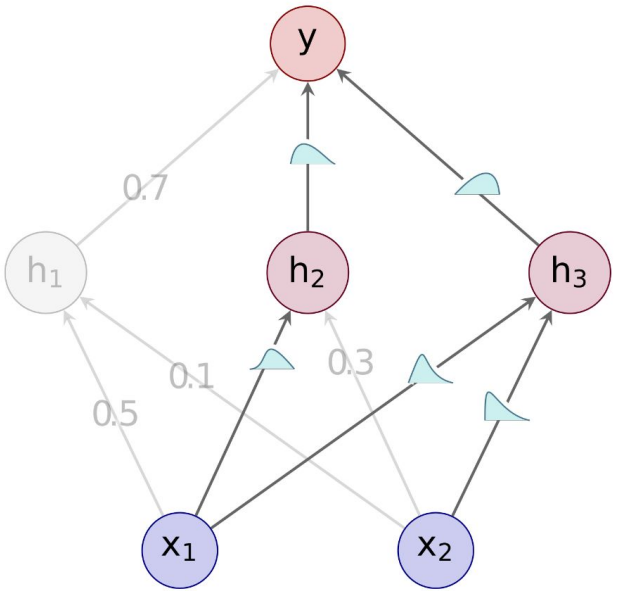
Subnetwork inference

1. MAP Estimation
2. Subnet Selection



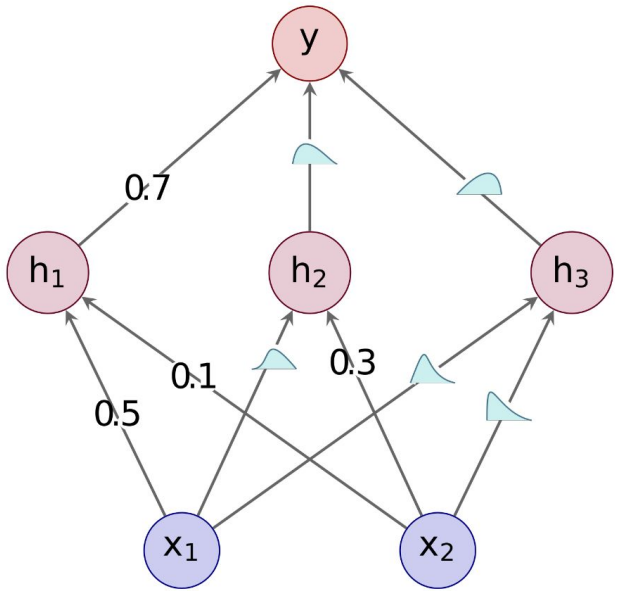
Subnetwork inference

1. MAP Estimation
2. Subnet Selection
3. Bayes. Inference



Subnetwork inference

1. MAP Estimation
2. Subnet Selection
3. Bayes. Inference
4. **Prediction**



Questions

1. How do we obtain the subnetwork posterior approximation $q(\mathbf{W}_S)$?

   Full-covariance Gaussian approximation via Laplace approximation.

2. How do we set the fixed values $\mathbf{w}_r^* \in \mathbb{R}$ of all the deterministic weights $\{\mathbf{w}_r\}_r$?

   Just leave them at their MAP estimates.

3. How do we select the subnetwork weights $\mathbf{w}_S$?

   Let's see in the next slide!

Subnetwork selection

Goal: Find subnetwork whose posterior is closest to the full network posterior.

$$
\begin{aligned}
&\min \mathrm{Wass}[\ \text{full posterior} \parallel \text{subnet posterior}\ ] \\
&= \min \mathrm{Wass}[\ p(\mathbf{W}|\boldsymbol{y}, \boldsymbol{X}) \parallel q(\mathbf{W})\ ] \\
&\approx \min \mathrm{Wass}[\ \mathcal{N}\left(\mathbf{W}; \boldsymbol{W}_{MAP}, H^{-1}\right) \parallel \mathcal{N}\left(\mathbf{W}_S; \boldsymbol{W}^S_{MAP}, H_S^{-1}\right) \prod_r \delta(\mathrm{w}_r - \mathrm{w}^r_{MAP})\ ]
\end{aligned}
$$

Intractable: this depends on all entries of the full network Hessian approximation H.

Assume that posterior factorizes: *H is diagonal*.

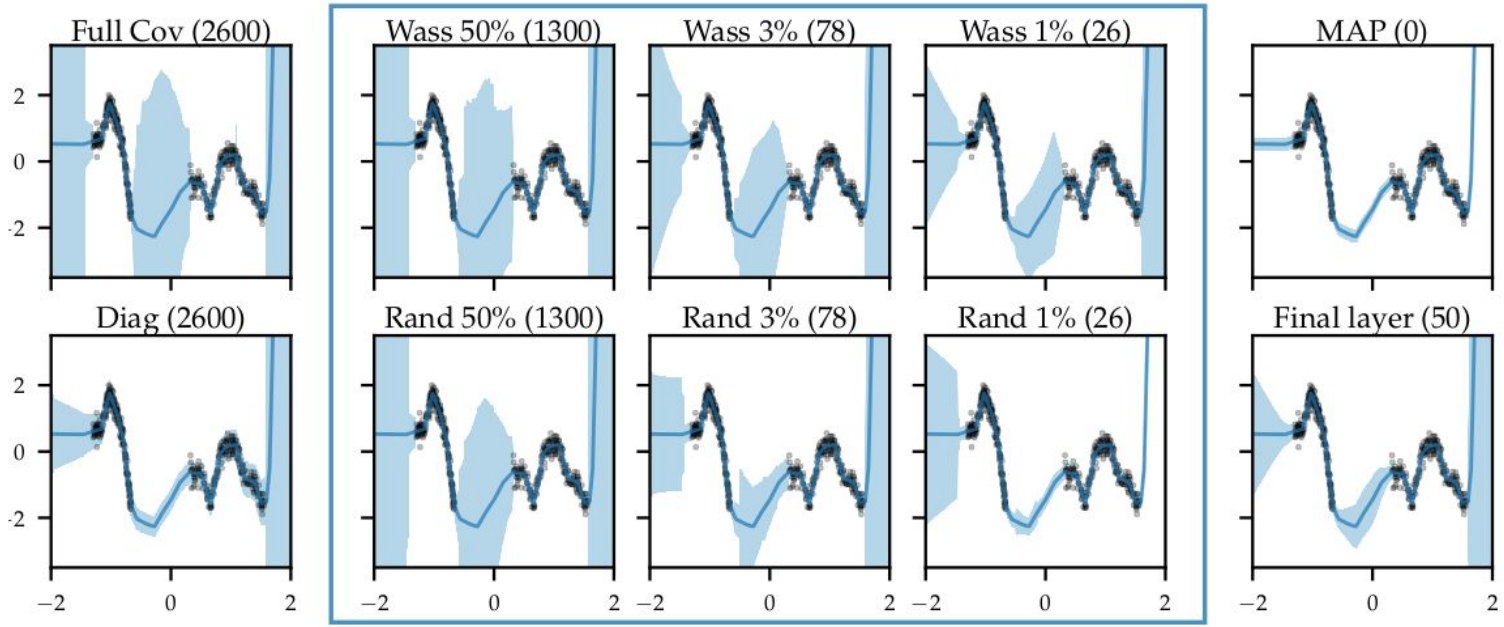
Diagonal assumption for subnetwork selection ≫ diag. assumption for inference

Wasserstein subnetwork selection:

1. Estimate a **factorized Gaussian** posterior over all weights (with diagonal SWAG).
2. Subnetwork = weights with **largest marginal variances** in factorized approx.

Results on 1D regression

Model: 2-layer NN with 2600 weights. **Goal**: test ‘in-between’ uncertainty (Foong 2019).



Expressive inference over a small subnetwork preserves **more uncertainty** than crude inference over the full network!

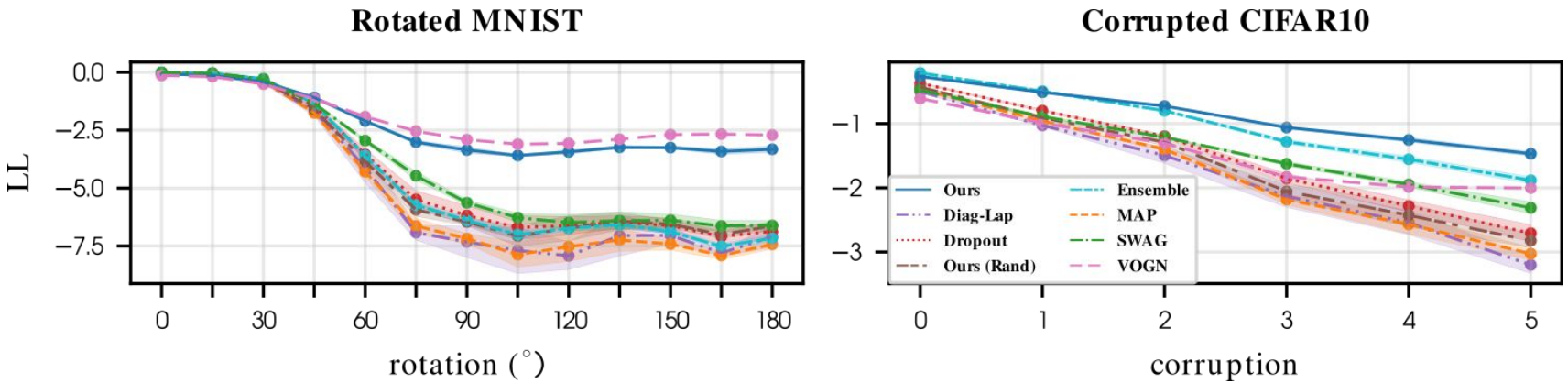
Image classification under distribution shift

Model: ResNet-18 with 11M weights. **Subnetwork**: just 42K (0.38%) weights.

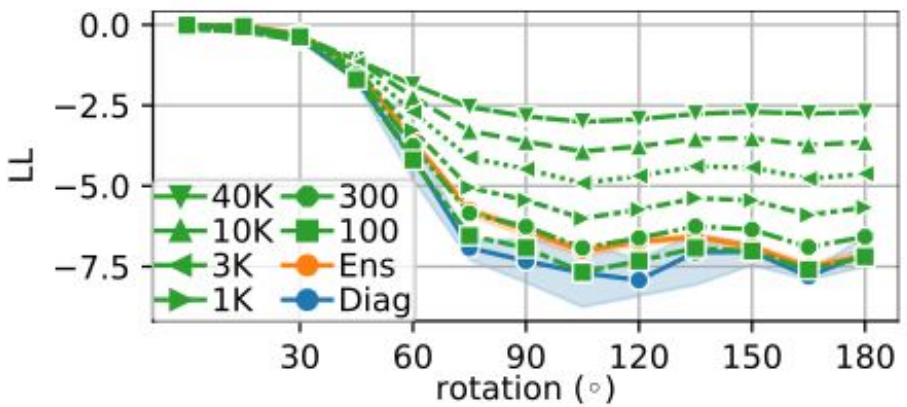
Baselines: MAP, Diagonal Laplace, MC Dropout, Deep Ensembles, SWAG, VOGN.

Rotated MNIST (Ovadia 2019)

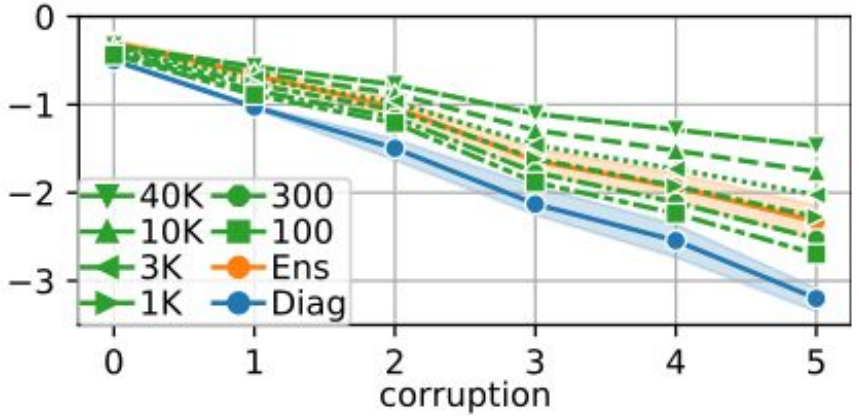
Corrupted CIFAR10 (Ovadia 2019)



Additional results for smaller subnetworks



(a) Rotated MNIST



(b) Corrupted CIFAR10

Take-home message

Linearized Laplace subnetwork inference, a new Bayesian deep learning method that

1. can be **easily applied** post-hoc to any pre-trained model.
2. is **more accurate** than crude approximations over the full network weights.
3. applies to **large neural networks** without sacrificing much its quality.
4. can outperform some **of the best existing methods** for uncertainty quantification.

Section 6

Adapting the linearized Laplace model evidence

Slides prepared by Javier Antoran

Antoran J., Janz D., Allingham J. A., Daxberger E., Barbano R., Nalisnick E. and Hernandez-Lobato J. M. Adapting the Linearised Laplace Model Evidence for Modern Deep Learning. In ICML, 2022.

Collaborators

Javiere Antoran

David Janz

James Allingham

Erik Daxberger

Riccardo Barbano

Eric Nalisnick

Preliminaries: the linearised Laplace method

- A Gaussian can be a very poor approximation to the NN posterior
- But it is a very good posterior for a linear model (in some cases exact)
  - $h(\theta, \cdot) = f(\tilde{\theta}, \cdot) + \partial_\theta f(\tilde{\theta}, \cdot)(\theta - \tilde{\theta})$

Affine in θ

Jacobian acts as basis expansion

- New loss $\mathcal{L}_h(\theta)$ is convex and $\approx \mathcal{G}_h(\theta) = \mathcal{L}_h(\tilde{\theta}) + ||\theta - \tilde{\theta}||^2_{\partial^2_\theta \mathcal{L}_h(\tilde{\theta})}$
- This **conjugate Gaussian-linear model** has:
  - Feature expansion $J(\cdot) = \partial_\theta f(\tilde{\theta}, \cdot)$, Design matrix $H = \partial^2_\theta \mathcal{L}_h(\tilde{\theta})$
  - Closed form predictive posterior and marginal likelihood

[Mackay, 1992]

Some questions one might have

- But wait, how did you find a mode of the NN loss to expand around?
  - We didn't, we used **S**GD and hoped for the best
- How did you deal with modern architecture elements, like batchnorm?
  - We used them and hoped for the best
- How did you tune hyperparameters?
  - Cross validation — in fact, the choice of Gaussian prior precision Λ makes a large difference in performance; it controls the size of the errorbars
- What about the model evidence?
  - We could not get it to work, it consistently choose prior precisions that overestimated uncertainty

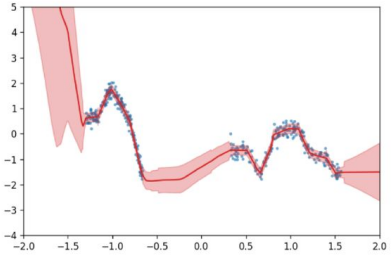
— Why doesn't it work?

Problem illustration

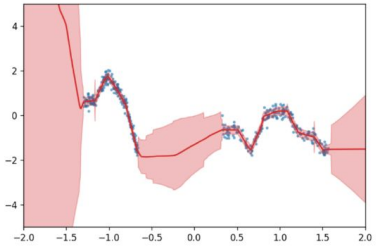
$$\Lambda = \lambda I$$

2 hidden layer, 2600 parameter, MLP with batchnorm

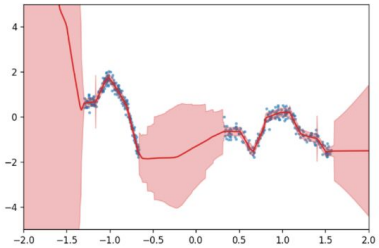
$\lambda = 100$



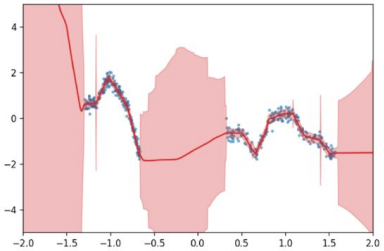
$\lambda = 10$



$\lambda = 5$

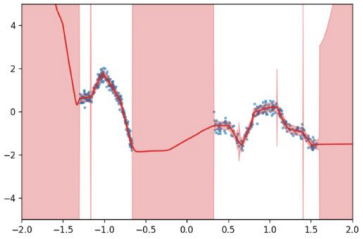


$\lambda = 1$



Largest $\mathcal{M}_{\tilde{\theta}}$

$\lambda = 0.1$



Why is this happening?

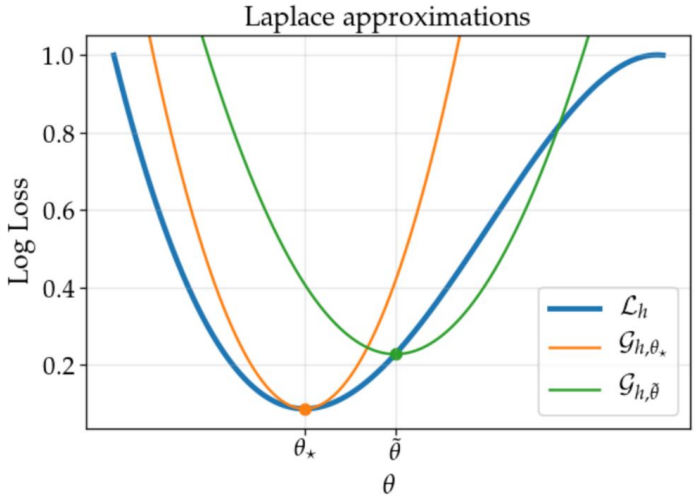
On the difficulty of finding a mode of the loss

- In 1992 Mackay did not use stochastic optimisation, early stopping or normalisation layers
- In modern settings $\tilde{\theta}$ is **not** a stationary point of $\mathscr{L}_f$
- $\partial_\theta \mathscr{L}_f(\tilde{\theta}) \neq 0 \implies \partial_\theta \mathscr{L}_h(\tilde{\theta}) \neq 0$

$$\mathscr{M}_{\tilde{\theta}}(\Lambda) =$$

$$\frac{-1}{2}(||\tilde{\theta}||^2_\Lambda + \text{logdet}(\Lambda^{-1}H + I)) + C$$

Wrong! **Not that wrong?**



The basis function linear model has a well defined optima

- We know that for any regularisation strength Λ, $\mathscr{L}_{h,\Lambda}(\theta)$ is convex
- We know that for any linearisation point θ, $\mathscr{M}_{\theta}(\Lambda)$ is concave

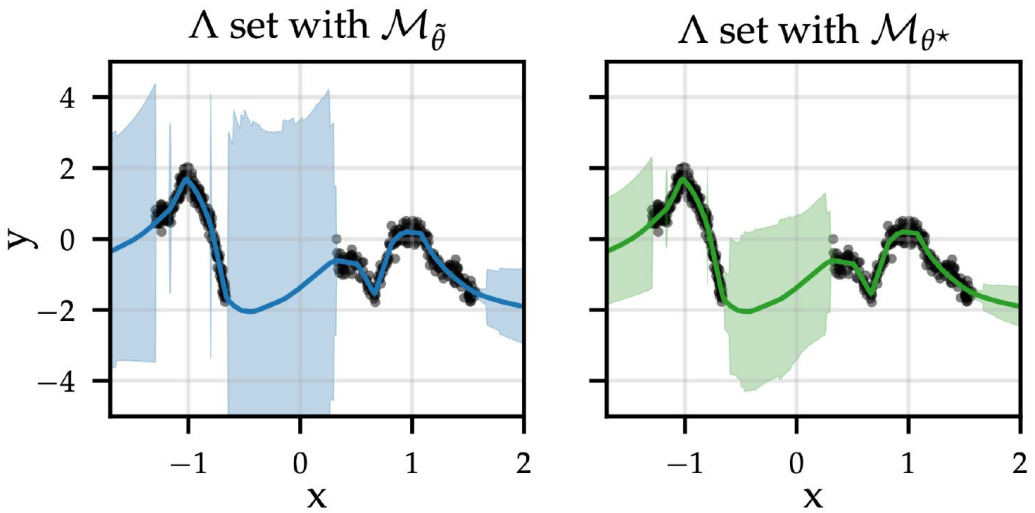
$\implies$ We can find joint stationary point $(\theta_\star, \Lambda_\star)$

$$\theta_\star \in \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}_{h,\Lambda_\star}(\theta) \quad \text{and} \quad \Lambda_\star \in \underset{\Lambda}{\operatorname{argmax}}\, \mathcal{M}_{\theta_\star}(\Lambda).$$

Recommendation 1: Keep $\tilde{\theta}$ as linearisation point but

$$\mathscr{M}_{\tilde{\theta}}(\Lambda) = \frac{-1}{2}(||\tilde{\theta}||^2_\Lambda + \text{logdet}(\Lambda^{-1}H + I)) + C \quad \rightarrow \quad \mathscr{M}_{\theta_\star}(\Lambda) = \frac{-1}{2}(||\theta_\star||^2_\Lambda + \text{logdet}(\Lambda^{-1}H + I)) + C$$

Following recommendation 1 improves errorbars



Could we find $\tilde{\theta} = \theta_\star$ by optimising our network better?

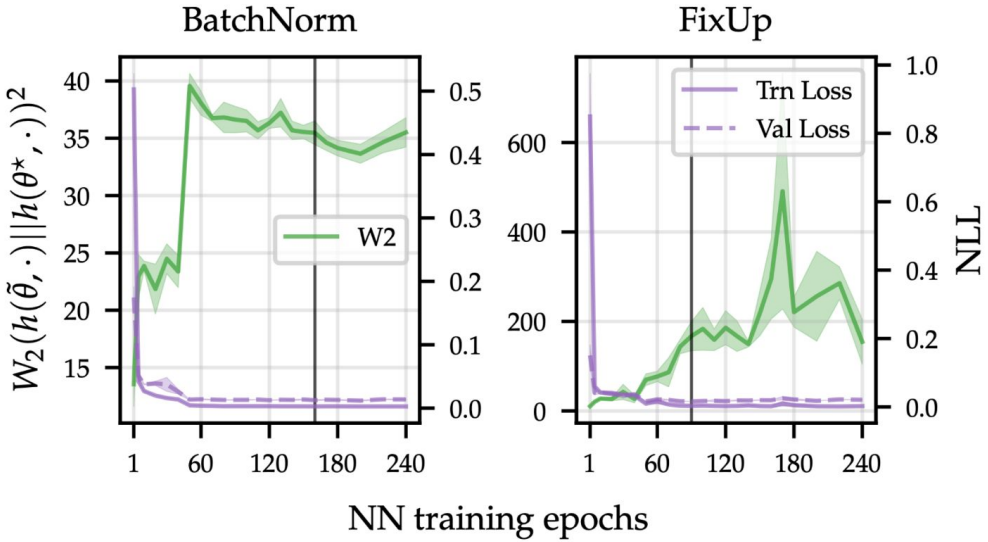


Figure 5. Wasserstein divergence between distributions obtained when employing $\mathcal{M}_{\tilde{\theta}}$ and $\mathcal{M}_{\theta_\star}$ as NN training progresses. The vertical black line indicates optimal early stopping.

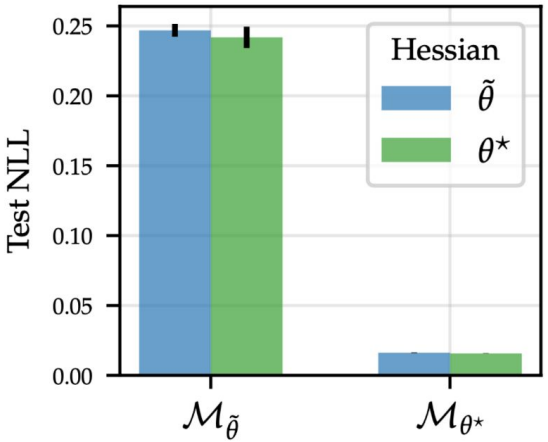
What about the choice of Hessian evaluation point?

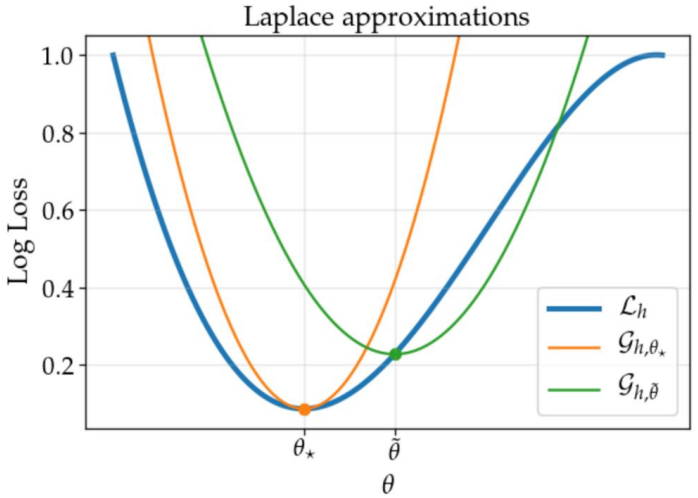
$$\mathscr{M}_{\theta_\star}(\Lambda) =$$

$$\frac{-1}{2}(||\theta_\star||^2_\Lambda + \text{logdet}(\Lambda^{-1}H + I)) + C$$

Not that wrong?

In fact, correct for regression!





ResNet image classification task:
most gains come from setting correct posterior mean in $\mathscr{M}$

Studying the effect of normalisation layers

Normalised networks:

Let $\theta = \theta' + \theta''$ with θ' having zero entries in the place of weights to which normalisation is applied and the opposite is true for θ'', then

$$f(\theta' + \theta'', \cdot) = f(\theta' + k\theta'', \cdot) \quad \text{for} \quad k > 0$$

Definition applies to:

- Batch norm
- Layer norm
- Group norm
- Normalisation-free ResNets

The MAP solution does not exist for normalised networks

Normalisation introduces scale invariance

$$f(\theta' + \theta'', \cdot) = f(\theta' + k\theta'', \cdot) \quad \text{for} \quad k > 0$$

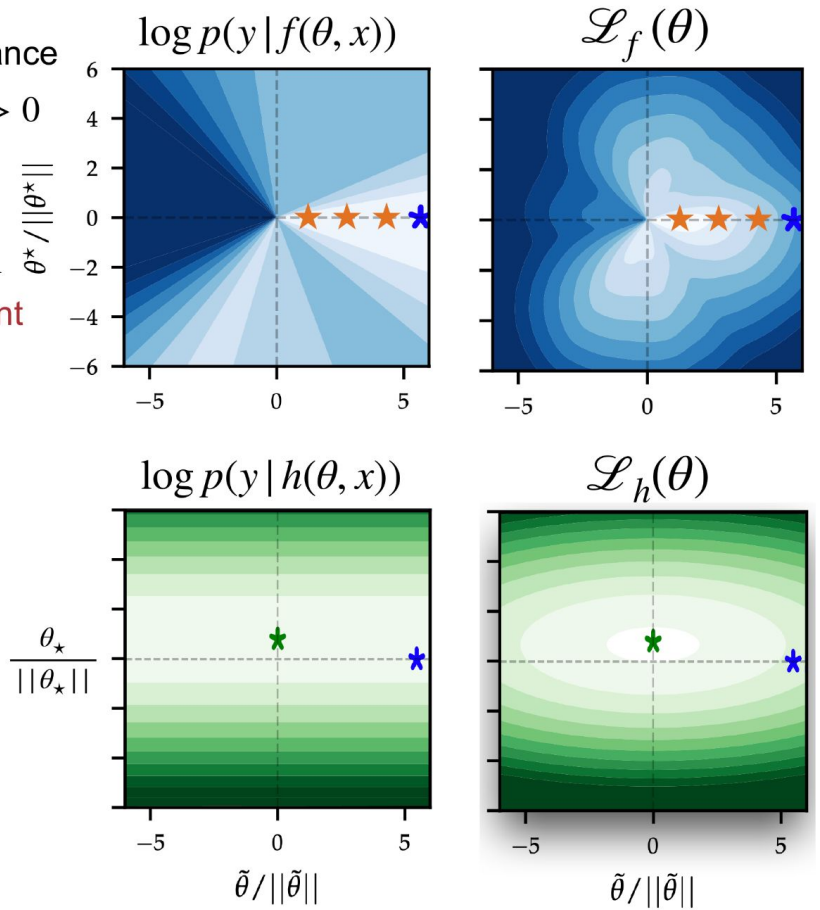
$$\mathscr{L}_f(\theta) = \underbrace{\log p(y \,|\, f(\theta, x))}_{\text{invariant}} + \underbrace{||\theta||^2_\Lambda}_{\text{not invariant}}$$

$$\implies \mathscr{L}_f(\theta' + \theta'') > \mathscr{L}_f(\theta' + \frac{1}{2}\theta'')$$

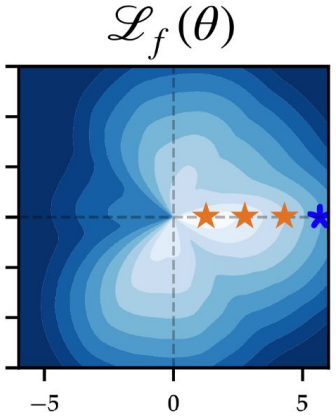
Linearisation point $\tilde{\theta}$ ★ can never be a mode of the posterior since the posterior has no modes

$\mathscr{L}_h(\theta)$ has a well defined mode $\theta_\star$

→ **apply recommendation 1**

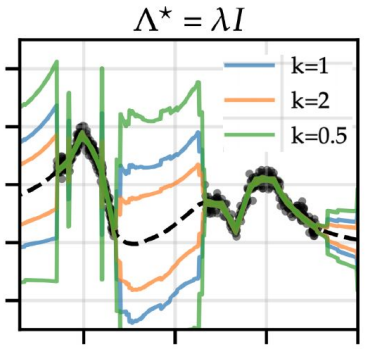


Dependence on scale of linearisation point k

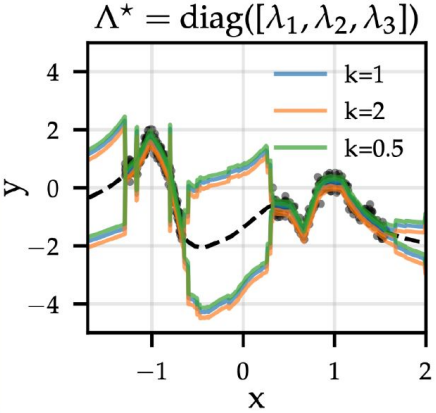


k does not affect NN predictions so it should not affect the predictive variance!

However, in general, it does!



Proposition 3. *For normalised neural networks, using a regulariser of the form* $\|\theta'\|^2_{\Lambda'} + \|\theta''\|^2_{\Lambda''}$ *with* Λ' *and* Λ'' *parametrised independently and chosen according to recommendation 1, the predictive posterior* $h(\theta, \cdot)$, $\theta \sim Q$ *induced by a linearisation point* $\theta' + k\theta''$ *is independent of the choice of* $k > 0$.



Recommendation 2: learn an independent regulariser for each normalised group of weights $\theta^{(n)}$, i.e.

$$\log \pi(\theta) \propto \lambda' ||\theta'||^2 + \sum_n \lambda^{(n)} ||\theta^{(n)}||^2$$

Systematic analysis (46k param models)

Table 1. Validation of recommendations across architectures. All results are reported as negative log-likelihoods (lower is better). In each column, the best performing method is bolded. For each $\mathcal{M}$, if single or multiple λ optimisation performs better it is underlined.

		T-FORMER	CNN	RESNET	PRE-RESNET	FIXUP ★	U-NET
$\mathcal{M}_{\theta_\star}$	single λ	**0.162** $\pm$ 0.042	**0.025** $\pm$ 0.000	**0.017** $\pm$ 0.000	0.017 $\pm$ 0.000	**0.055** $\pm$ 0.006	—
	multiple λs	**0.162** $\pm$ 0.042	**0.025** $\pm$ 0.000	**0.016** $\pm$ 0.001	**0.016** $\pm$ 0.000	0.061 $\pm$ 0.005	**-2.240** $\pm$ 0.027
$\mathcal{M}_{\tilde{\theta}}$	single λ	0.310 $\pm$ 0.060	0.253 $\pm$ 0.001	0.252 $\pm$ 0.006	0.220 $\pm$ 0.004	0.153 $\pm$ 0.021	—
	multiple λs	**0.162** $\pm$ 0.042	0.205 $\pm$ 0.002	0.236 $\pm$ 0.005	0.239 $\pm$ 0.004	0.200 $\pm$ 0.018	-1.703 $\pm$ 0.023

Recommendation 1 + Recommendation 2 is best in all cases

* Fixup is a non-scale invariant alternative to normalisation layers so recommendation 2 does not apply

Validation on ResNet-50 (23M parameters)

Table 2. Test negative log-likelihoods for ResNet-50 on CIFAR10.

		BatchNorm	FixUp
$\mathcal{M}_{\theta_\star,\text{simple}}$	single λ	**-0.773** $\pm$ 0.004	-0.744 $\pm$ 0.000
	multiple λs	**-0.778** $\pm$ 0.003	**-0.801** $\pm$ 0.000
$\mathcal{M}_{\theta_\star,\text{full}}$	single λ	-0.645 $\pm$ 0.005	-0.563 $\pm$ 0.002
	multiple λs	-0.639 $\pm$ 0.009	-0.641 $\pm$ 0.001
$\mathcal{M}_{\tilde{\theta}}$	single λ	-0.269 $\pm$ 0.004	-0.387 $\pm$ 0.000
	multiple λs	-0.271 $\pm$ 0.004	-0.437 $\pm$ 0.000

Recommendation 1 + Recommendation 2 is best in all cases

We employ standard KFAC approximation for scalable Hessian computations

Wrapping up: treat your NNs as kernels!

- Linearised Laplace should not be naively applied to modern NNs.
  - Every linearisation point $\tilde{\theta}$ defines a tangent linear model. Linearised Laplace uses this model to provide errorbars. Choosing hyperparameters using this model's evidence avoids pathologies.

- Is the tangent linear model a good surrogate for the NN?
  - For NNs with linear dense output layers, $f(\tilde{\theta}, \cdot)$ is in the linear span of the Jacobian basis expansion $J(\cdot) = \partial_\theta f(\tilde{\theta}, \cdot)$

- Furthermore, for normalised networks with dense output layers:
  - Linearisation simplifies to $h(\theta, \cdot) = J(\cdot)\theta,\ \theta \sim \mathcal{N}(0, \Lambda)$ and thus induces a GP prior $f \sim \mathrm{GP}(0, J\Lambda^{-1}J^T)$

Section 7

Case study: X-ray image reconstruction

Slides prepared by Javier Antoran

Antorán J., Barbano R., Leuschner J., Hernández-Lobato J. M., Jin B. A Probabilistic Deep Image Prior for Computational Tomography, arXiv:2203.00479, 2022

Collaborators

Javier Antoran

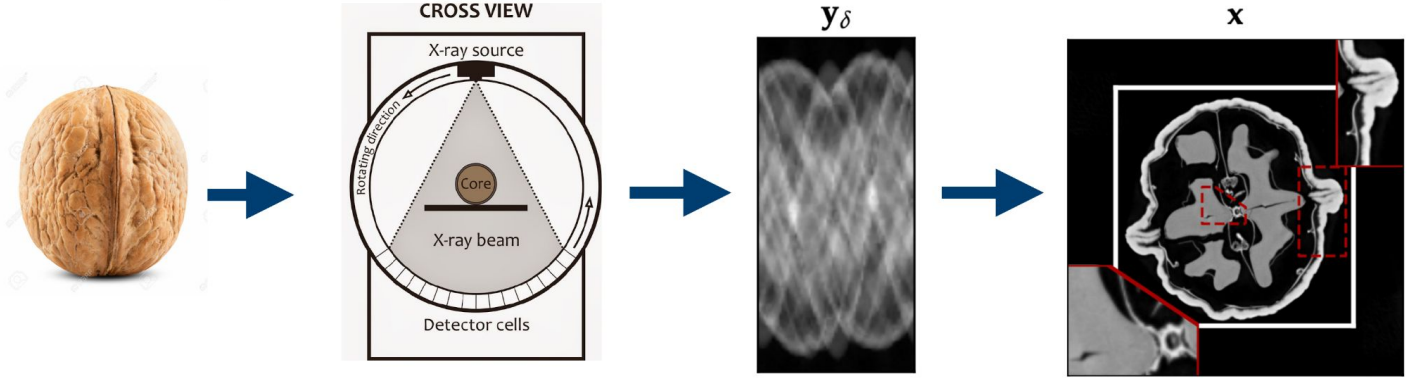
Riccardo Barbano

Johannes Leuschner

Bangti Jin

A brief primer on inverse problems

- Consider the setting $y_\delta = Ax + \eta, \quad \eta \sim \mathcal{N}(0, \sigma_y^2 I)$ and
- We observe $y_\delta \in \mathbb{R}^{d_y}$ and are tasked with recovering $x \in \mathbb{R}^{d_x}$, and $d_x \gg d_y$



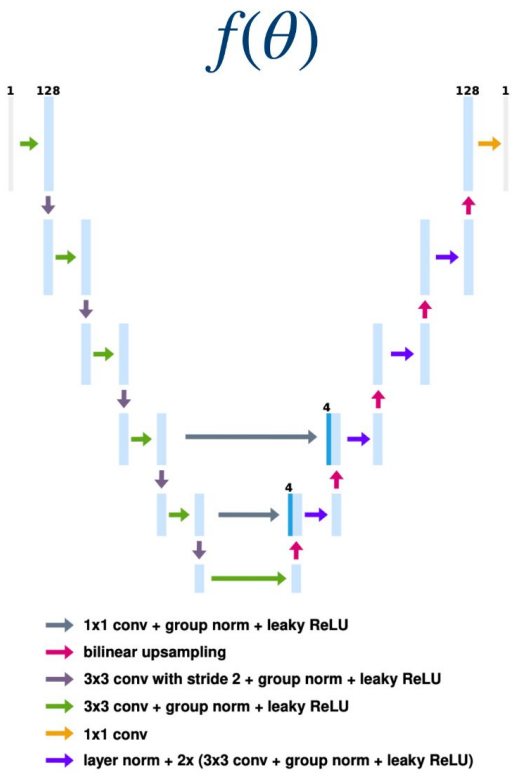
- Clearly the problem is ill posed
  - Traditionally, x is estimated through regularised reconstruction
  - Can we design a Bayesian prior $p(x)$ to solve this task?

The “deep image prior” for inverse problems

- Standard solution: “Deep image prior”

$$\operatorname{argmin}_\theta \underbrace{(y_\delta - Af(\theta))^2}_{\text{data fit}} + \lambda \underbrace{\mathrm{TV}(f(\theta))}_{\text{classical regulariser}}$$

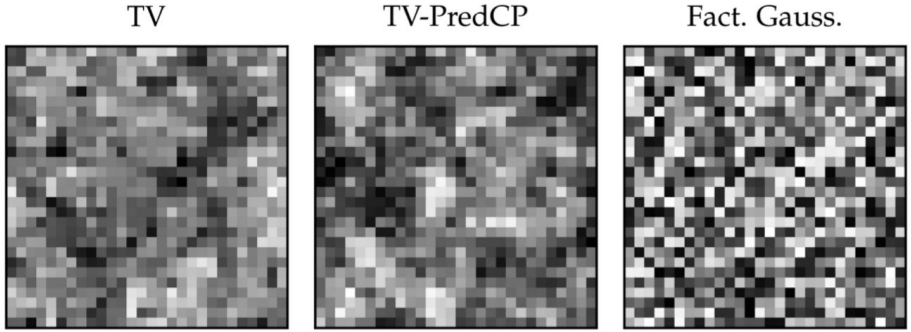
Can be interpreted as a MAP objective given a prior that constrains reconstructions to be the output of a U-net and have low TV



From regularised reconstruction to Bayesian inference

- Can build unnormalised prior $p(f) \propto \exp(-\lambda \mathrm{TV}(f))$
  - Normalising constant does not admit closed form
  - Hessian is 0 almost everywhere $\Longrightarrow$ can't use Laplace
- **Idea**: build surrogate Gaussian prior with a covariance kernel that enforces TV smoothness

$$f \sim N(0, K(\Lambda)), \quad \Lambda \sim p(\Lambda) = \mathrm{Exp}(TV(f); \lambda) \left| \frac{\partial \mathrm{TV}(f)}{\partial \Lambda} \right|$$



Building a probabilistic deep image prior

1. Train U-net with standard objective: $(y_\delta - Af(\theta))^2 + \mathrm{TV}(f(\theta))$

2. Linearise around some acceptable parameter setting $\tilde{\theta}$

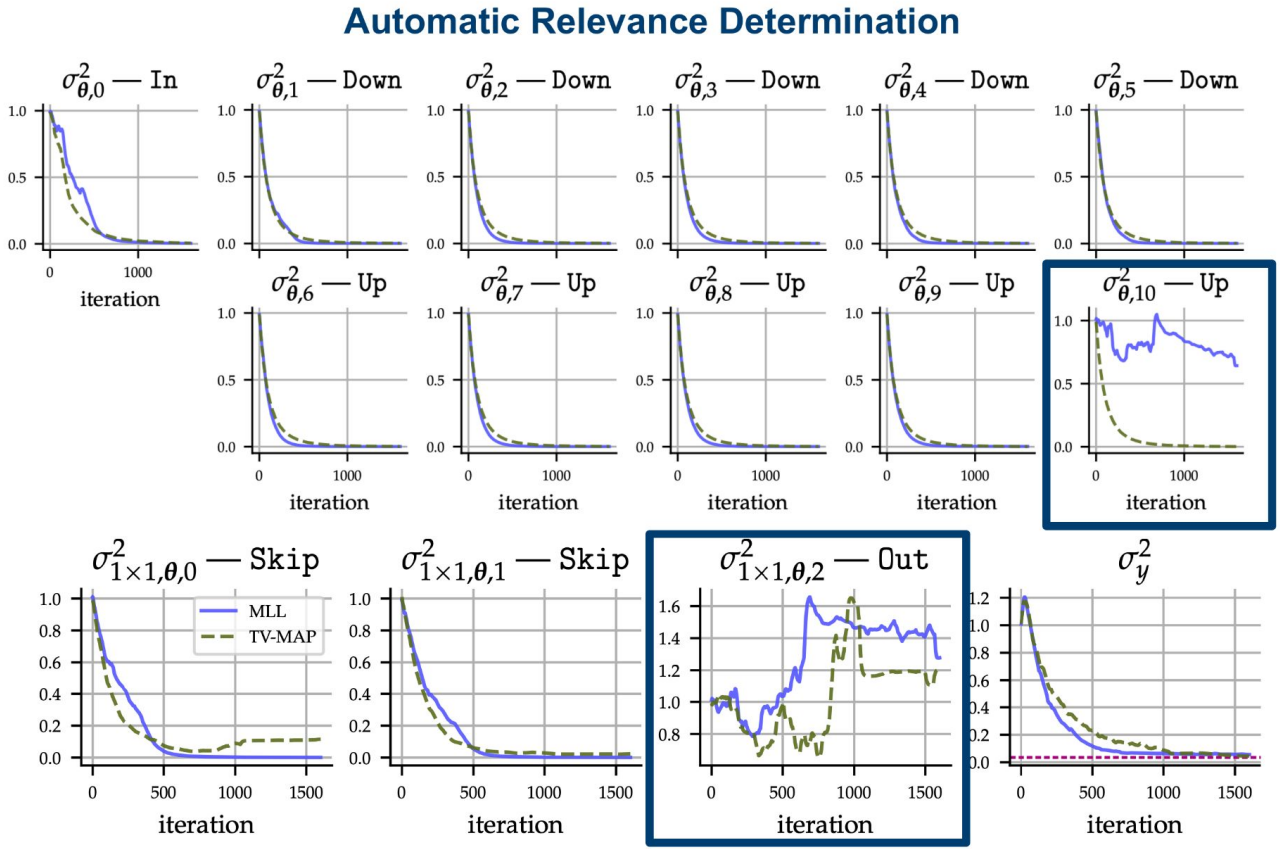
3. Build Bayesian hierarchical model

$$y_\delta \sim \mathcal{N}(Af,\, \sigma_y^2 I), \quad f \sim \mathcal{N}(0,\, J\Lambda^{-1}J^T), \quad \Lambda \sim p(\Lambda) = \mathrm{Exp}(TV(f);\lambda) \left| \frac{\partial \mathrm{TV}(f)}{\partial \Lambda} \right|$$

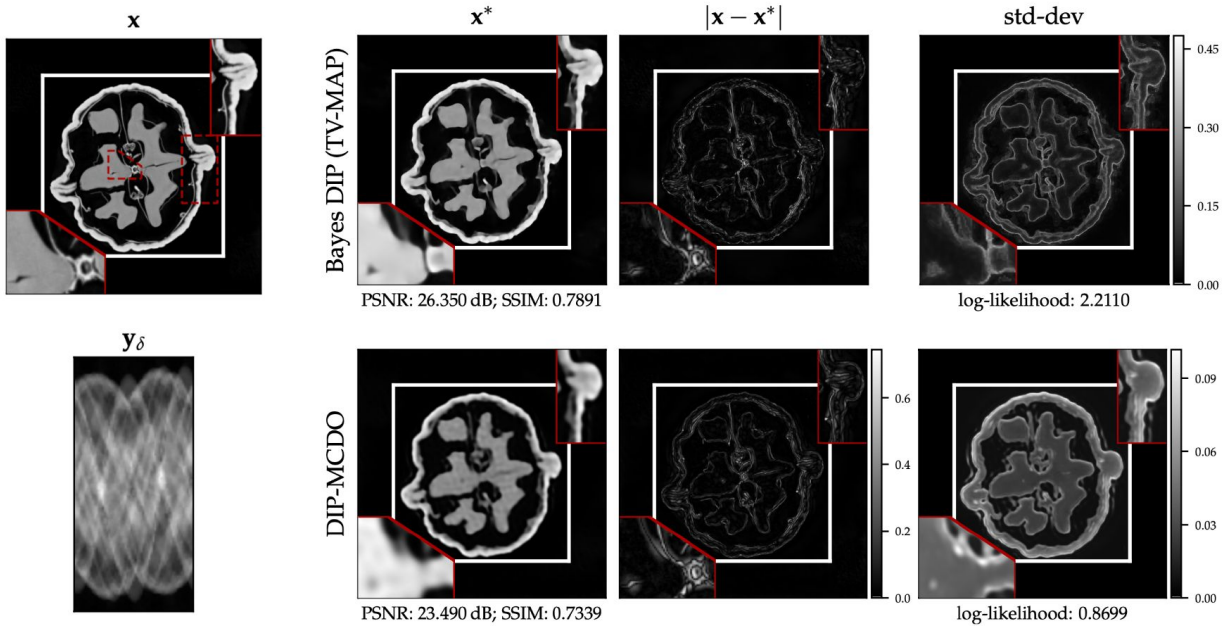
4. Optimise hyperparameters with marginal likelihood

5. Make predictions (cheap because $d_y \ll d_x, d_\theta$)

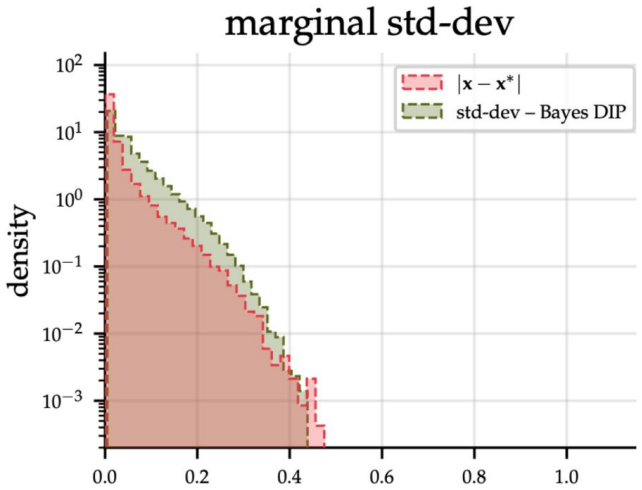
Optimising hyperparameters with the marginal likelihood



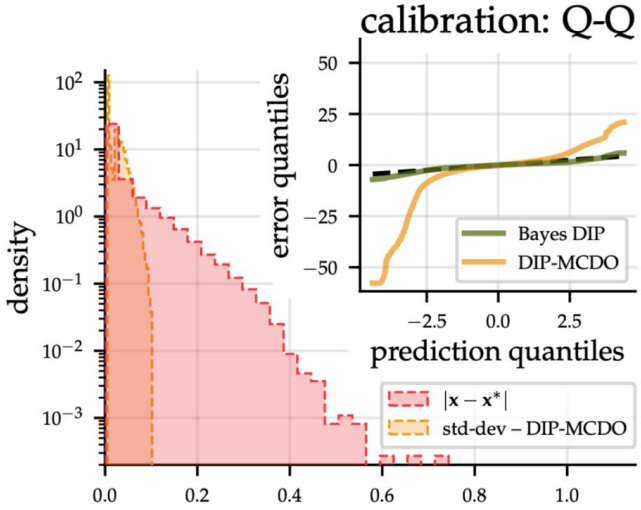
Some results



Calibration comparison



Lin Laplace



Dropout

Take-home message 2: use deep learning to build specialised kernels

- We can obtain very powerful task-specific kernels by training a NN to solve a task and then linearising it.

- Once the network is trained, we the tangent linear model $f \sim \mathrm{GP}(0,\, J\Lambda^{-1}J^T)$ provides us with uncertainty estimates and a model selection objective.

Thanks!