

# 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](mailto: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

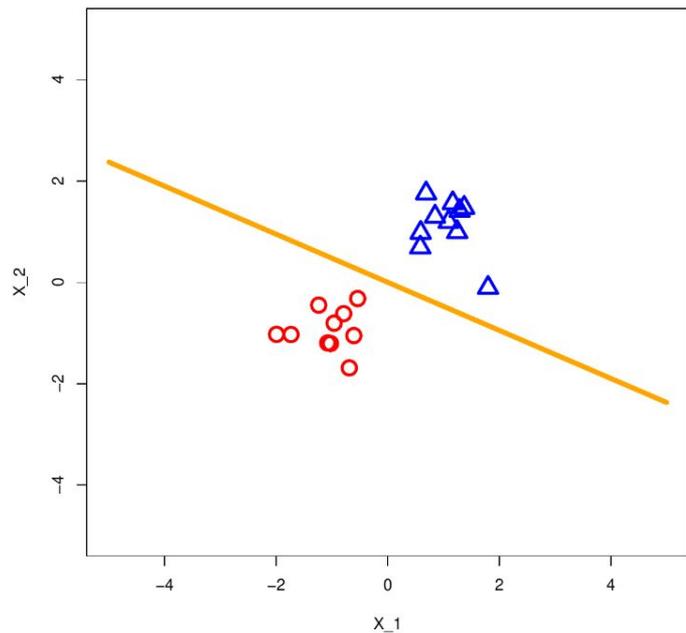
# Section 1

## Motivation for a Bayesian Approach



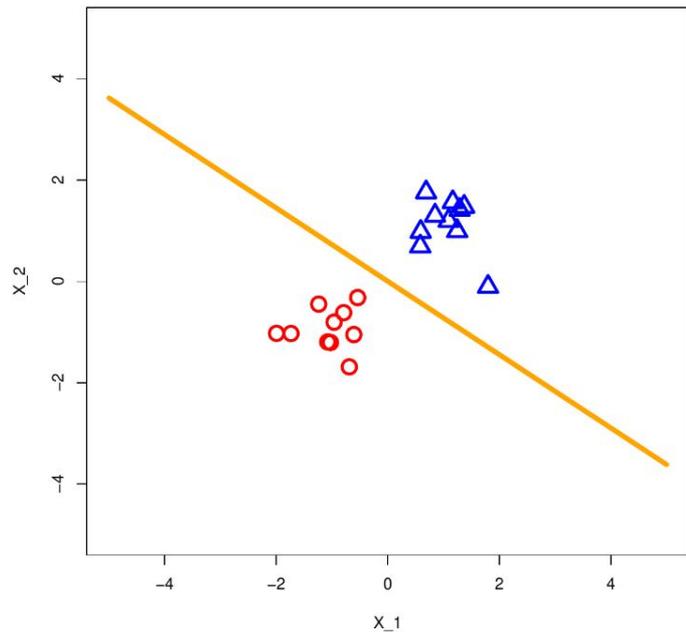
Consider linear classification with **linearly separable** data  $\{y_n, \mathbf{x}_n\}_{n=1}^N$ ,  $y_n \in \{-1, 1\}$ ,  $\mathbf{x}_n \in \mathbb{R}^d$ .

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



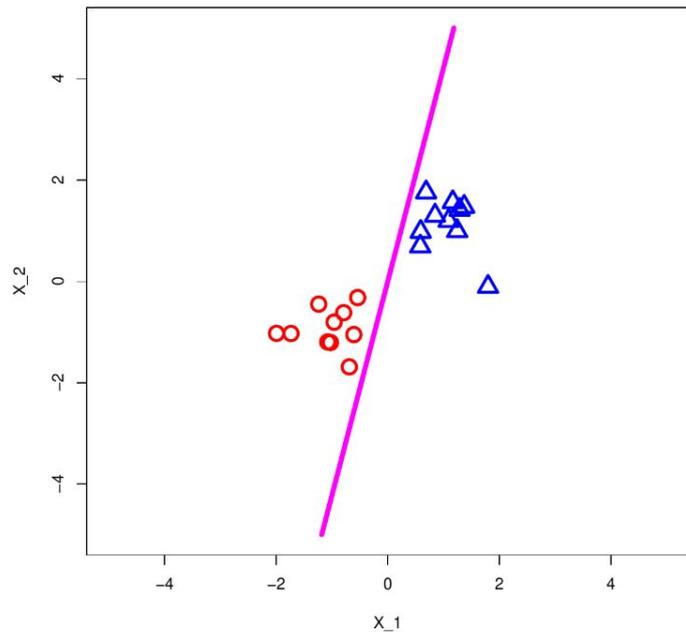
Consider linear classification with **linearly separable** data  $\{y_n, \mathbf{x}_n\}_{n=1}^N$ ,  $y_n \in \{-1, 1\}$ ,  $\mathbf{x}_n \in \mathbb{R}^d$ .

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



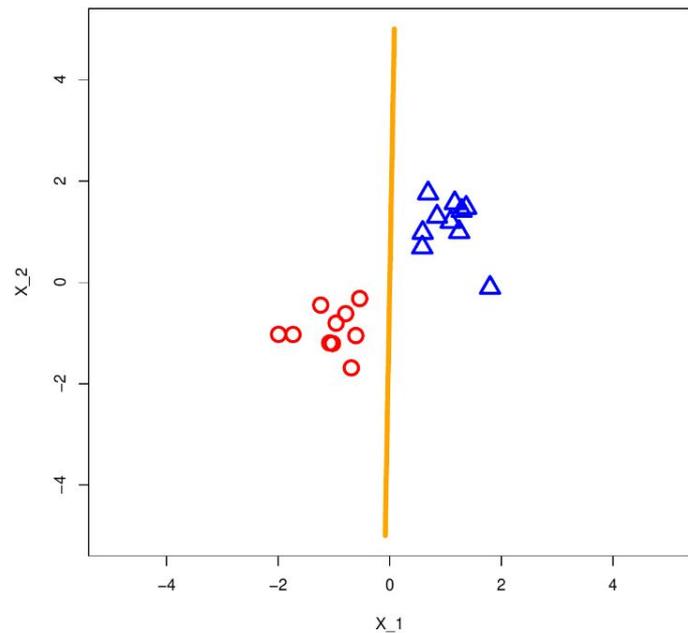
Consider linear classification with **linearly separable** data  $\{y_n, \mathbf{x}_n\}_{n=1}^N$ ,  $y_n \in \{-1, 1\}$ ,  $\mathbf{x}_n \in \mathbb{R}^d$ .

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



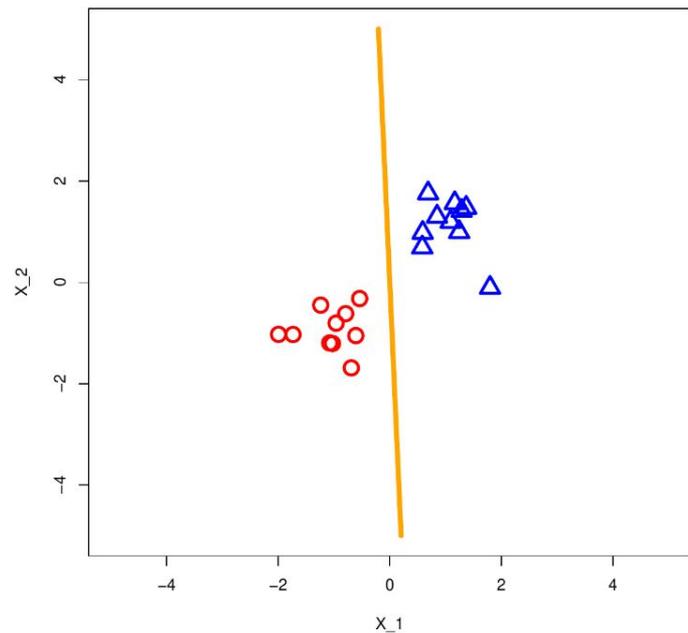
Consider linear classification with **linearly separable** data  $\{y_n, \mathbf{x}_n\}_{n=1}^N$ ,  $y_n \in \{-1, 1\}$ ,  $\mathbf{x}_n \in \mathbb{R}^d$ .

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



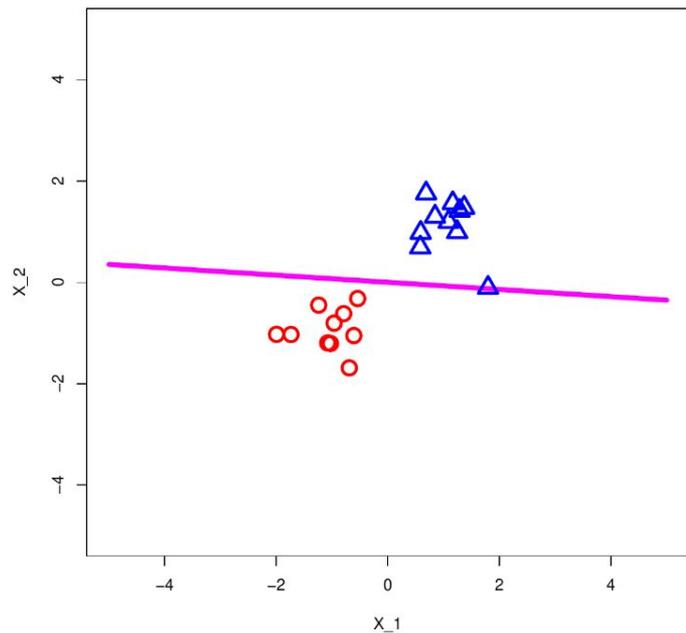
Consider linear classification with **linearly separable** data  $\{y_n, \mathbf{x}_n\}_{n=1}^N$ ,  $y_n \in \{-1, 1\}$ ,  $\mathbf{x}_n \in \mathbb{R}^d$ .

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



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

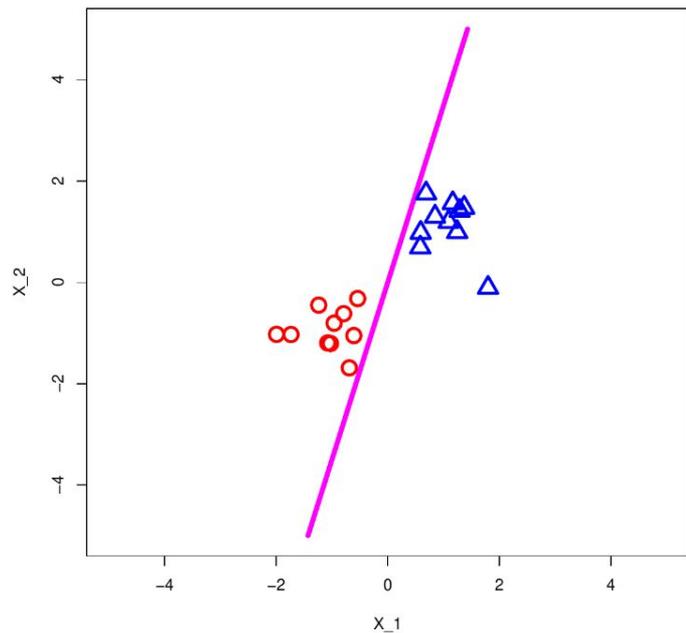
Many separating hyperplanes  $\mathbf{w}$  fit the data equally well. Which one to choose?





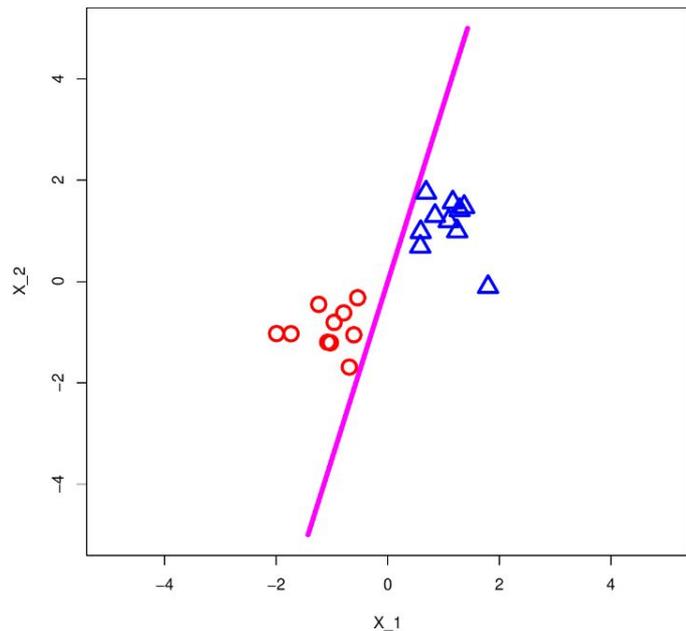
Consider linear classification with **linearly separable** data  $\{y_n, \mathbf{x}_n\}_{n=1}^N$ ,  $y_n \in \{-1, 1\}$ ,  $\mathbf{x}_n \in \mathbb{R}^d$ .

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

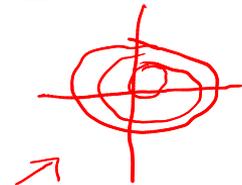


Consider linear classification with **linearly separable** data  $\{y_n, \mathbf{x}_n\}_{n=1}^N$ ,  $y_n \in \{-1, 1\}$ ,  $\mathbf{x}_n \in \mathbb{R}^d$ .

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



**Solution: Bayesian inference.**



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

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

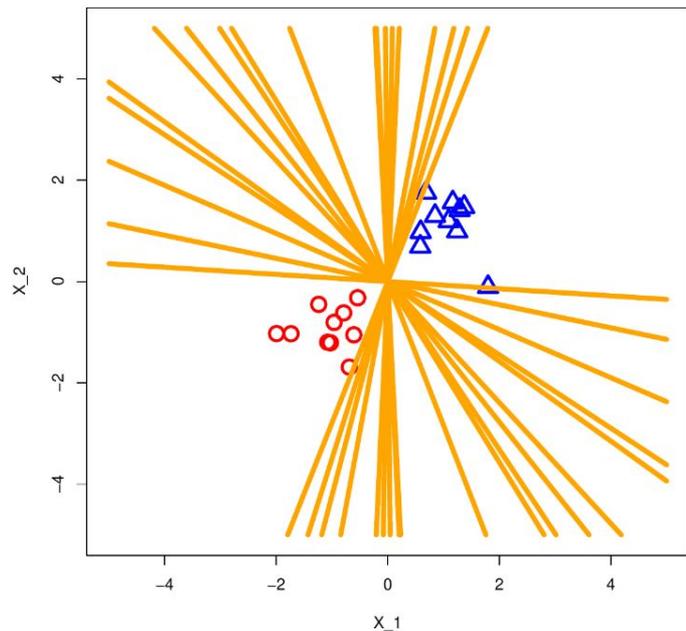
$$p(\mathbf{w}|\mathbf{y}, \mathbf{X}) \propto \left[ \prod_{n_1}^N \sigma(y_n \mathbf{w}^T \mathbf{x}_n) \right] p(\mathbf{w}),$$

$\sigma(-x) = 1 - \sigma(x)$

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

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

Consider linear classification with **linearly separable** data  $\{y_n, \mathbf{x}_n\}_{n=1}^N$ ,  $y_n \in \{-1, 1\}$ ,  $\mathbf{x}_n \in \mathbb{R}^d$ . Many separating hyperplanes  $\mathbf{w}$  fit the data equally well. Which one to choose?



**Solution: Bayesian inference.**

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

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

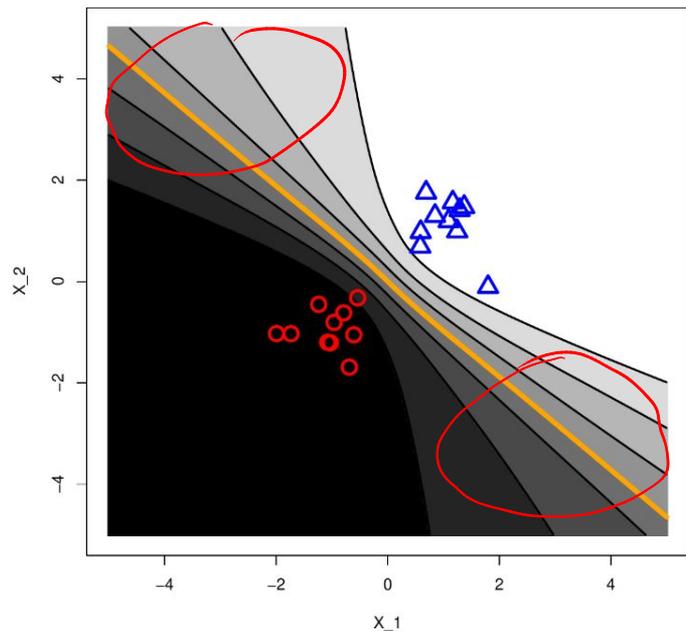
$$p(\mathbf{w}|\mathbf{y}, \mathbf{X}) \propto \left[ \prod_{n=1}^N \sigma(y_n \mathbf{w}^T \mathbf{x}_n) \right] p(\mathbf{w}),$$

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

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

Consider linear classification with **linearly separable** data  $\{y_n, \mathbf{x}_n\}_{n=1}^N$ ,  $y_n \in \{-1, 1\}$ ,  $\mathbf{x}_n \in \mathbb{R}^d$ .  
 Many separating hyperplanes  $\mathbf{w}$  fit the data equally well. Which one to choose?

**Predictive Distribution**



**Solution: Bayesian inference.**

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

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

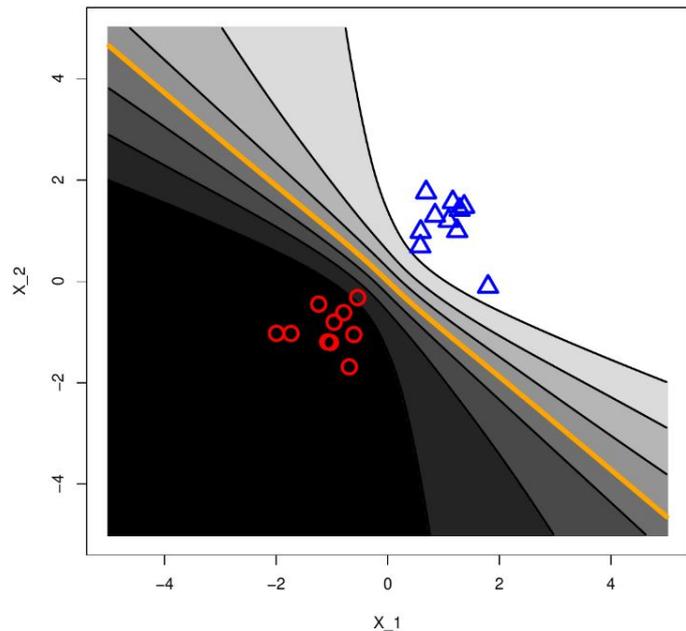
$$p(\mathbf{w}|\mathbf{y}, \mathbf{X}) \propto \left[ \prod_{n_1}^N \sigma(y_n \mathbf{w}^T \mathbf{x}_n) \right] p(\mathbf{w}),$$

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

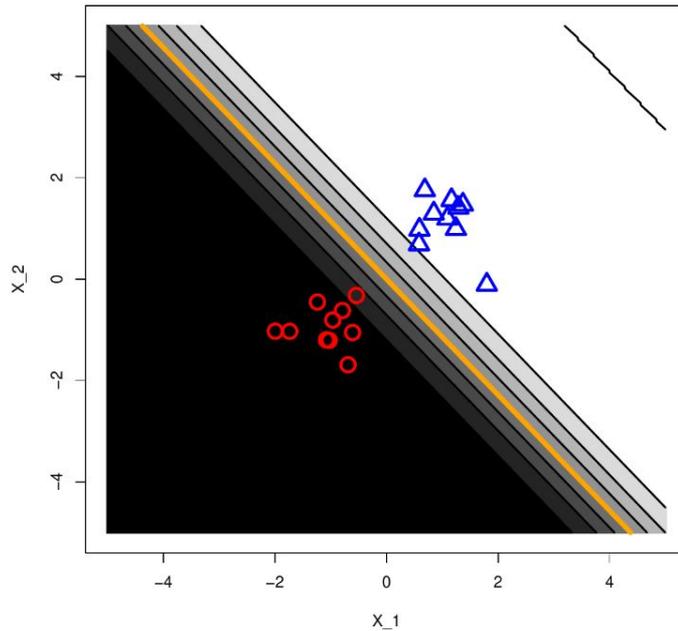
$$p(y_*|\mathbf{x}_*, \mathbf{y}, \mathbf{X}) = \int \sigma(y_* \mathbf{w}^T \mathbf{x}_*) p(\mathbf{w}|\mathbf{y}, \mathbf{X}) d\mathbf{w},$$

Consider linear classification with **linearly separable** data  $\{y_n, \mathbf{x}_n\}_{n=1}^N$ ,  $y_n \in \{-1, 1\}$ ,  $\mathbf{x}_n \in \mathbb{R}^d$ . Many separating hyperplanes  $\mathbf{w}$  fit the data equally well. Which one to choose?

**Predictive Distribution**

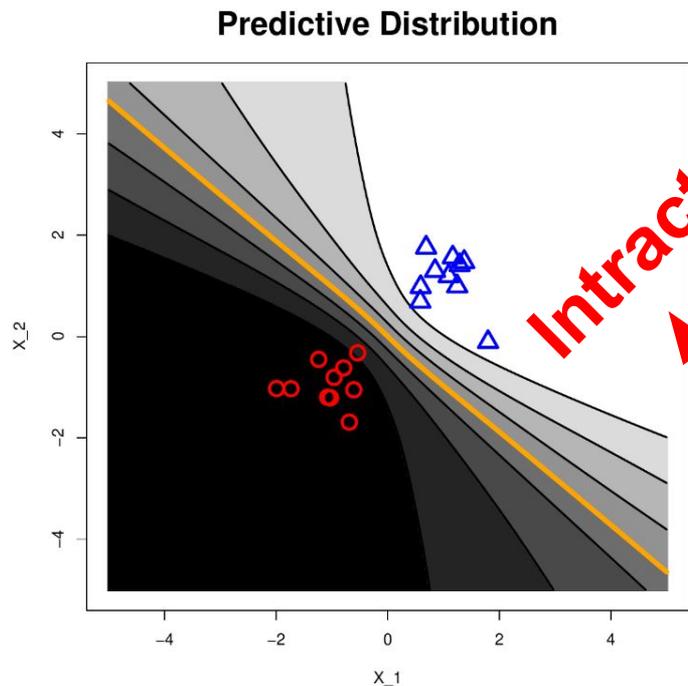


**MAP**



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

Consider linear classification with **linearly separable** data  $\{y_n, \mathbf{x}_n\}_{n=1}^N$ ,  $y_n \in \{-1, 1\}$ ,  $\mathbf{x}_n \in \mathbb{R}^d$ .  
 Many separating hyperplanes  $\mathbf{w}$  fit the data equally well. Which one to choose?



**Intractable!**

**Solution: Bayesian inference.**

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

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

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

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

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

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

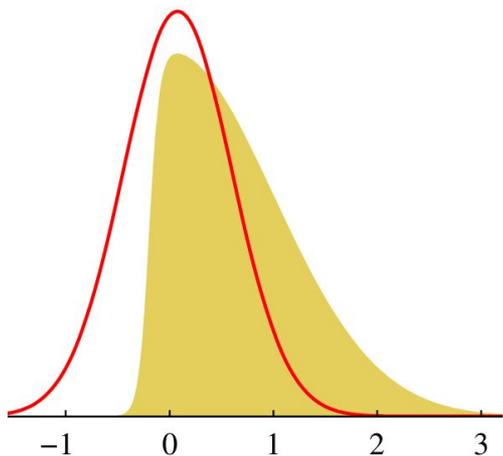
## **Section 2**

# **The basics of the Laplace approximation**

# Main idea behind Laplace approximation

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

One-dimensional Example



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

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

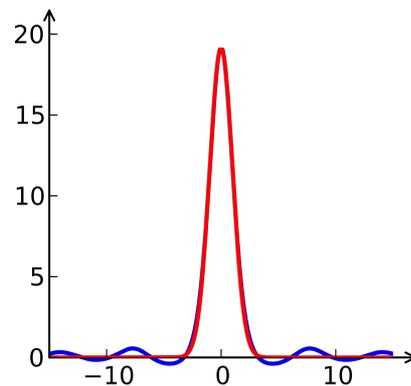
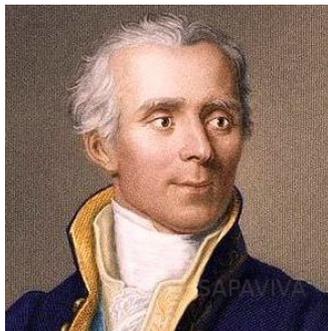


Figure source: Bishop, 2008.

# 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.

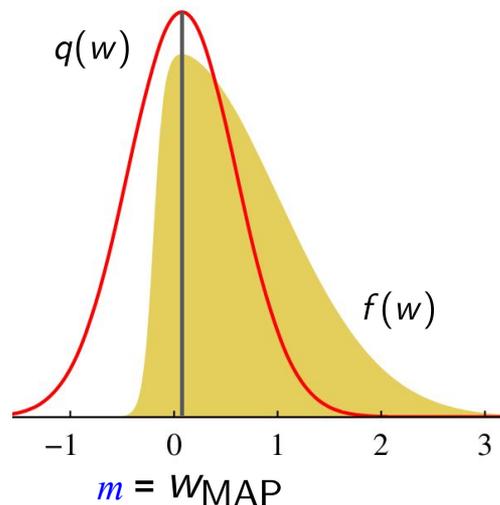
How to adjust  $m$  and  $v$ ?

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

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

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

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  $w_{\text{MAP}}$ :

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

$\nearrow$  MEAN
 $\searrow$  INV VARIANCE

$x = a$

$$f(x) = f(a) + \frac{1}{1!} f'(a)(x-a) + \frac{1}{2!} f''(a)(x-a)^2 + \dots$$

Taking the exponential we obtain:

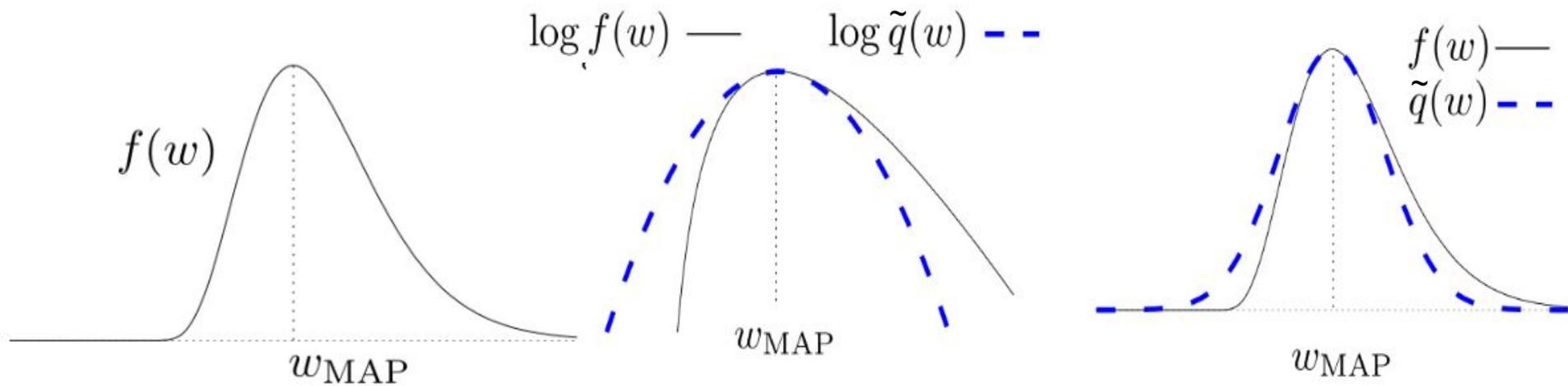
$$f(w) \approx f(w_{\text{MAP}}) \exp \left\{ -\frac{a}{2} (w - w_{\text{MAP}})^2 \right\} = \tilde{q}(w) \quad q(w) = \mathcal{N}(w | w_{\text{MAP}}, a^{-1}),$$

$$\mathcal{N}(x | m, v) = \frac{1}{\sqrt{2\pi v}} \exp \left\{ -\frac{1}{2v} (x - m)^2 \right\}$$

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



# The multivariate case

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

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

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

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

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

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

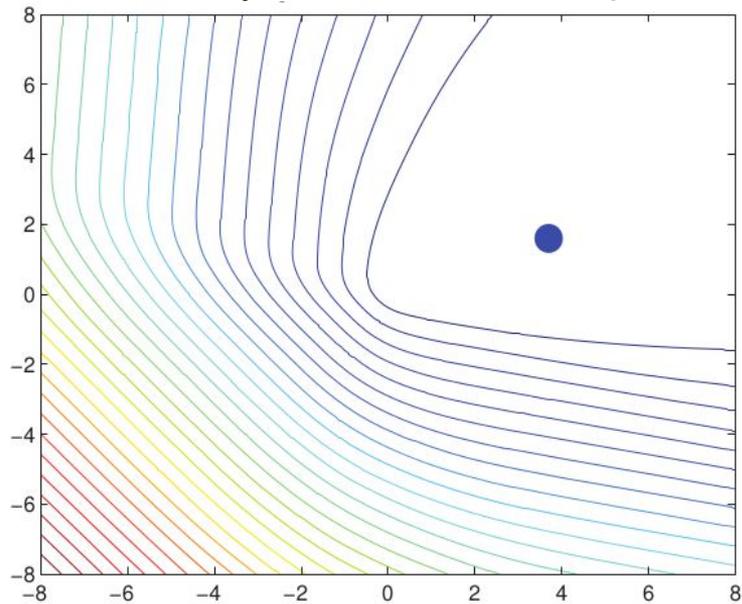
$$q(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{w}_{\text{MAP}}, \mathbf{A}^{-1}),$$

$$\sqrt{(2\pi)^d |\mathbf{V}|}$$

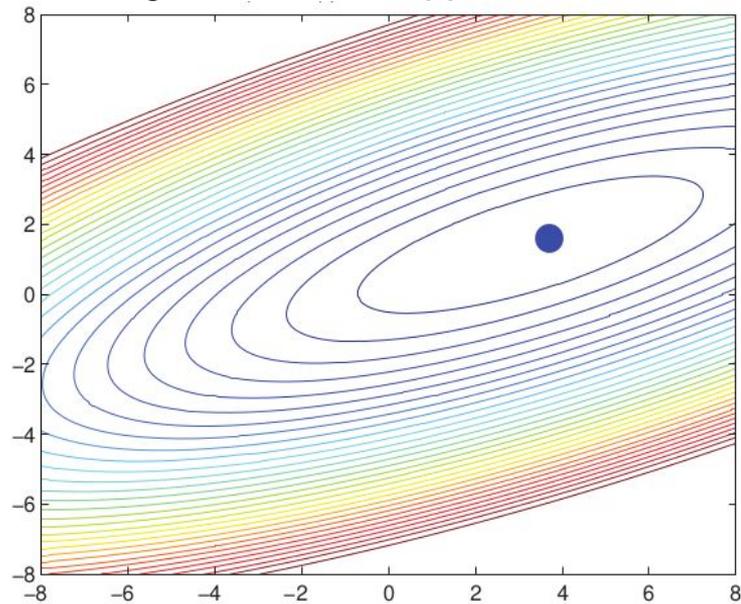
$$Z = \int f(\mathbf{w}) d\mathbf{w} \approx \int \tilde{q}(\mathbf{w}) d\mathbf{w} = f(\mathbf{w}_{\text{MAP}})(2\pi)^{d/2} |\mathbf{A}|^{-1/2}.$$

# 2d example

Log un-normalized posterior  
in binary classification example



Log of Gaussian approximation



# Example: Bayesian logistic regression

The posterior distribution is

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

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

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

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

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

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

# Probit approximation to the predictive distribution

We need to evaluate  $p(y_* | \mathbf{x}_*, \mathbf{y}) \approx \int p(y_* | \mathbf{x}_*, \mathbf{w}) q(\mathbf{w}) d\mathbf{w} = \int \sigma(y_* \mathbf{w}^\top \mathbf{x}_*) \mathcal{N}(\mathbf{w} | \mathbf{w}_{\text{MAP}}, \mathbf{A}^{-1}) d\mathbf{w}$

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  $\sigma(x)$ :

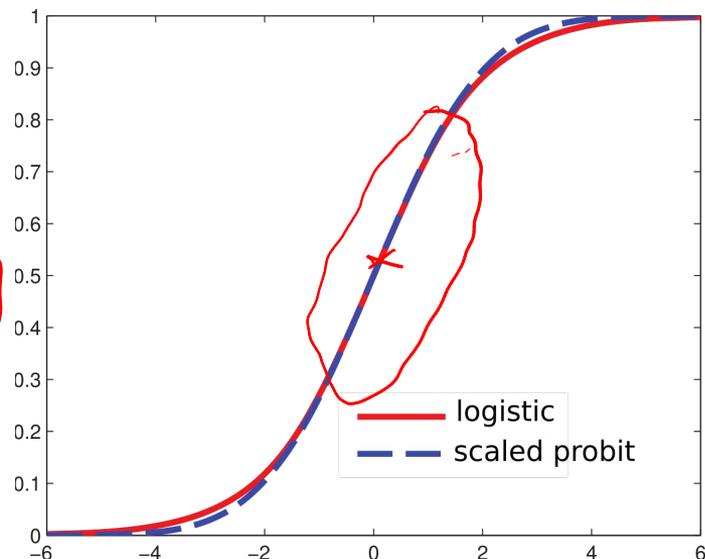
$$\sigma(x) \approx \Phi(\lambda x), \quad \text{where } \lambda^2 = \pi / 8.$$

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

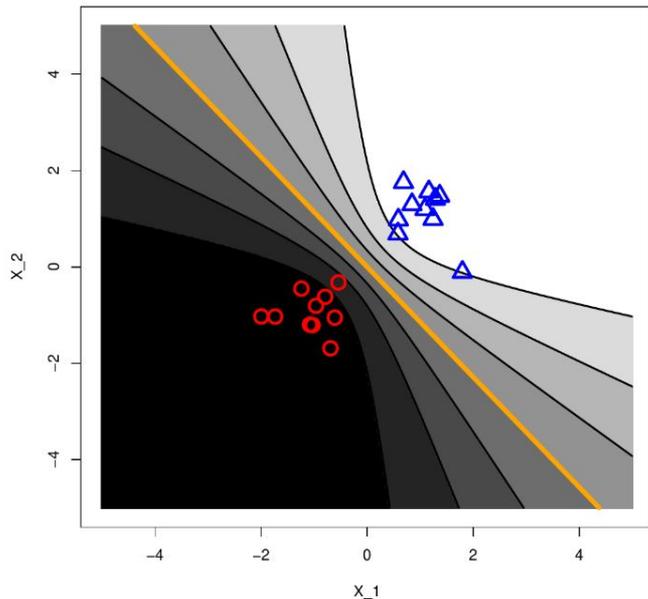
$$\int \sigma(x) \mathcal{N}(x | m, v) dx \approx \sigma(m / \sqrt{1 + \frac{\pi}{8} v})$$



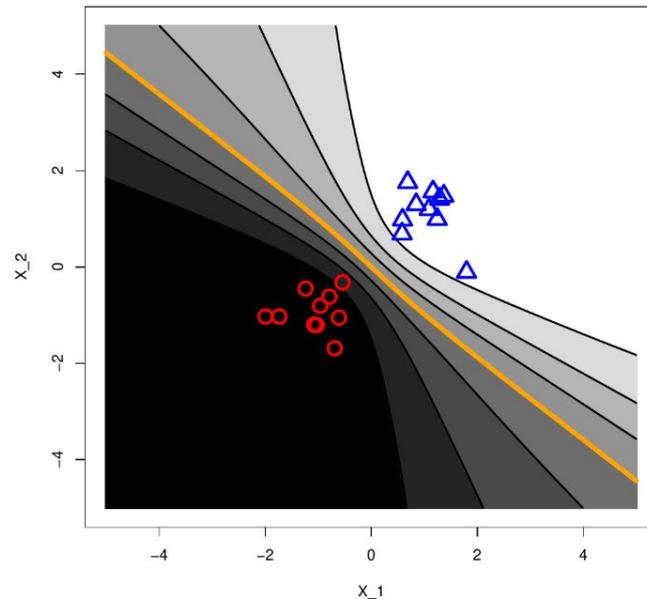
# Results in logistic regression example

$$p(y_* | \mathbf{x}_*, \mathbf{y}) \approx \int \sigma(y_* \mathbf{w}^\top \mathbf{x}_*) \mathcal{N}(\mathbf{w} | \mathbf{w}_{\text{MAP}}, \mathbf{A}^{-1}) d\mathbf{w} \approx \sigma \left( \frac{y_* \mathbf{w}_{\text{MAP}}^\top \mathbf{x}_*}{\sqrt{1 + \frac{\pi}{8} \mathbf{x}_*^\top \mathbf{A}^{-1} \mathbf{x}_*}} \right).$$

Laplace Approximation



Exact

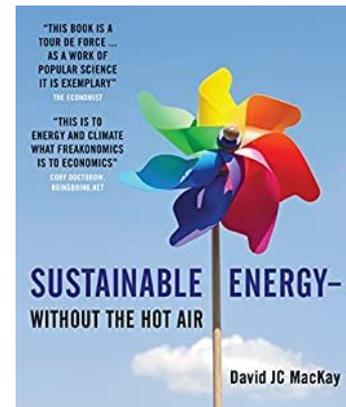
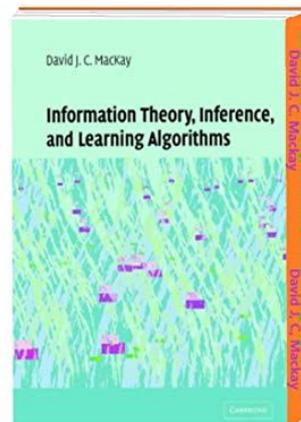
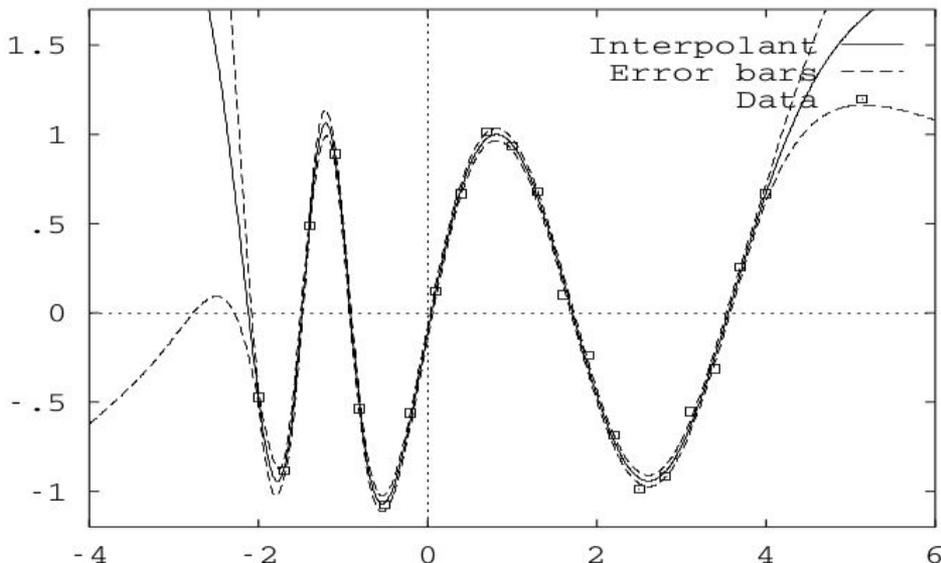


## **Section 3**

# **The Laplace approximation in Bayesian neural networks**

# Bayesian methods for adaptive models

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



Mackay won a 1993 [energy prediction competition](#) using the methods we are about to see.

# Posterior distribution in Bayesian neural networks

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

Let  $\mathbf{w} \equiv (w_1, \dots, w_d)^T$  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 *noise precision*

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

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

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

The **log posterior** is then

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

# The Laplace approximation in Bayesian neural networks

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

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

where

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

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

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

*Handwritten notes:*  
- A red circle around  $\alpha \mathbf{I}$  with an arrow pointing to the word "prior".  
- A red bracket under the entire equation with an arrow pointing to the expression  $q(\mathbf{w}_{\text{MAP}})$ .

# The Laplace approximation in Bayesian neural networks

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

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

where

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

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

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

Favours fit to the data

# The Laplace approximation in Bayesian neural networks

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

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

where

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

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

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

Favours fit to the data

Penalizes model complexity

# Approximating the predictive distribution

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

$$f(\mathbf{x}; \mathbf{w}) \approx f(\mathbf{x}; \mathbf{w}_{\text{MAP}}) + \underbrace{\nabla f(\mathbf{x}; \mathbf{w})}_{\text{linear expansion}} \Big|_{\mathbf{w}_{\text{MAP}}}^{\text{T}} (\mathbf{w} - \mathbf{w}_{\text{MAP}}) \equiv \hat{f}(\mathbf{x}; \mathbf{w})$$

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

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

where

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

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

# Approximating the predictive distribution

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

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

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

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

where

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

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

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

# Approximating the predictive distribution

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

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

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

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

where

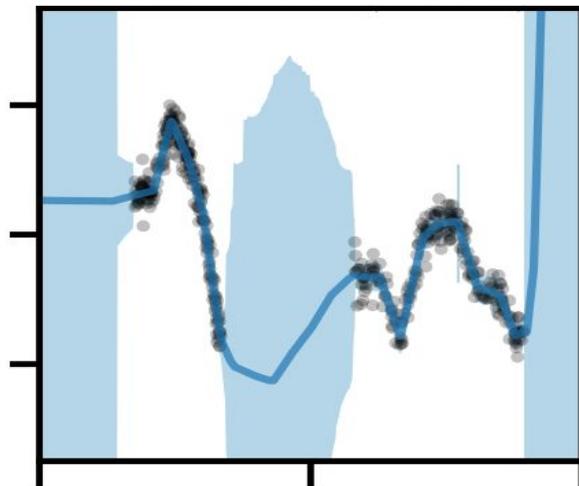
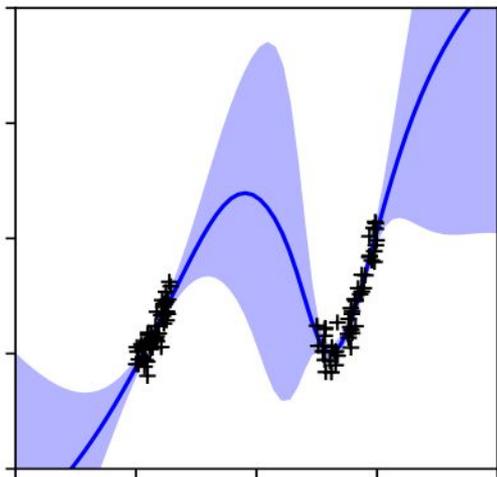
$$m_{\star} = f(\mathbf{x}_{\star}; \mathbf{w}_{\text{MAP}})$$

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

- **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 neurons, 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.

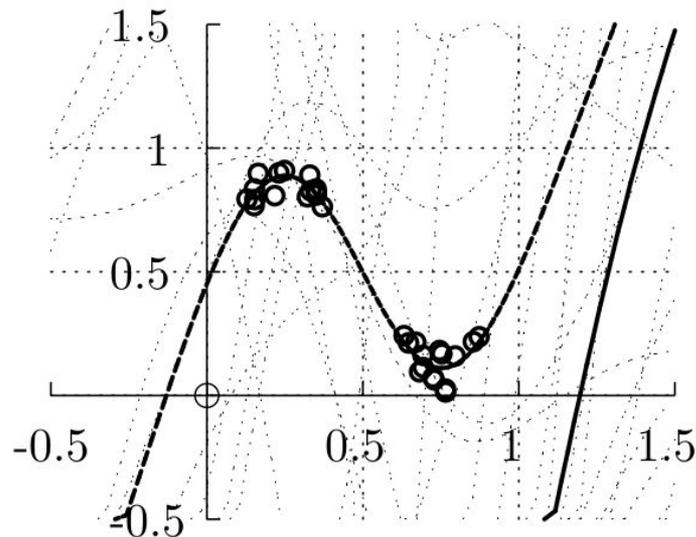
# Why use a linearized model for prediction?

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

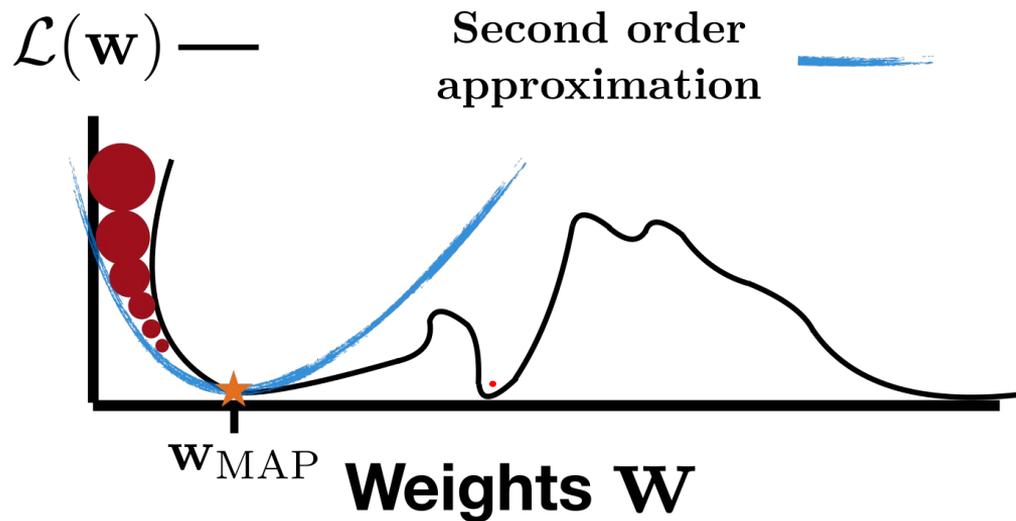
Neil D. Lawrence. Variational Inference in Probabilistic Models. PhD thesis, Cambridge, 2000.

- ..... Samples from Gaussian approximation
- Average network output under generate samples, **most of which is off scale**
- Output from network using  $\mathbf{W}_{\text{MAP}}$ .

**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 $\alpha$

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

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

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

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

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

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

# Tuning the prior precision $\alpha$

## Eigenvalues of

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

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

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

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

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

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

# Tuning the prior precision $\alpha$

## Eigenvalues of

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

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

Effective number of well-determined parameters

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

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

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

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

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

# Tuning the noise precision $\beta$

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

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

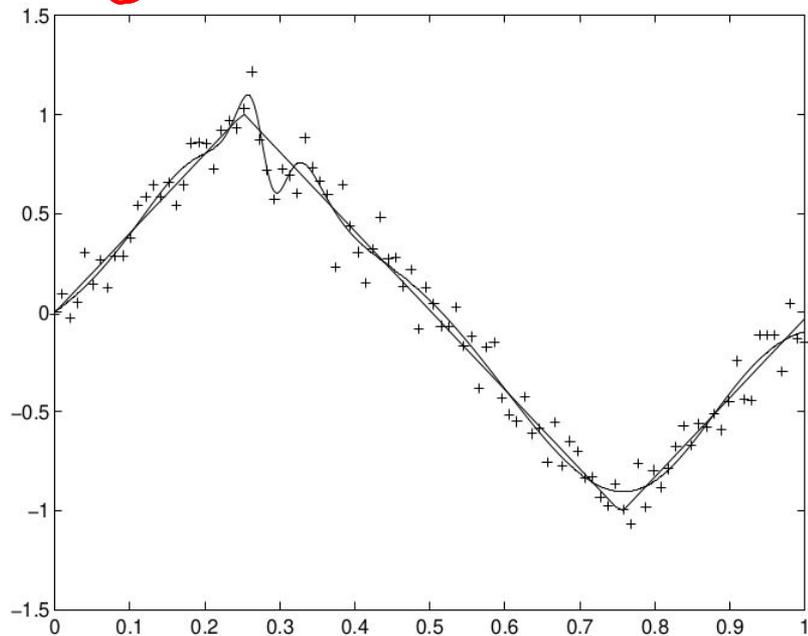
$$\frac{d\lambda_i}{d\beta} = \frac{\lambda_i}{\beta} \quad \text{and thus} \quad \frac{d}{d\beta} \log |\mathbf{A}| = \frac{d}{d\beta} \sum_{i=1}^d \log(\lambda_i + \alpha) = \frac{1}{\beta} \sum_{i=1}^d \frac{\lambda_i}{\lambda_i + \alpha}$$

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

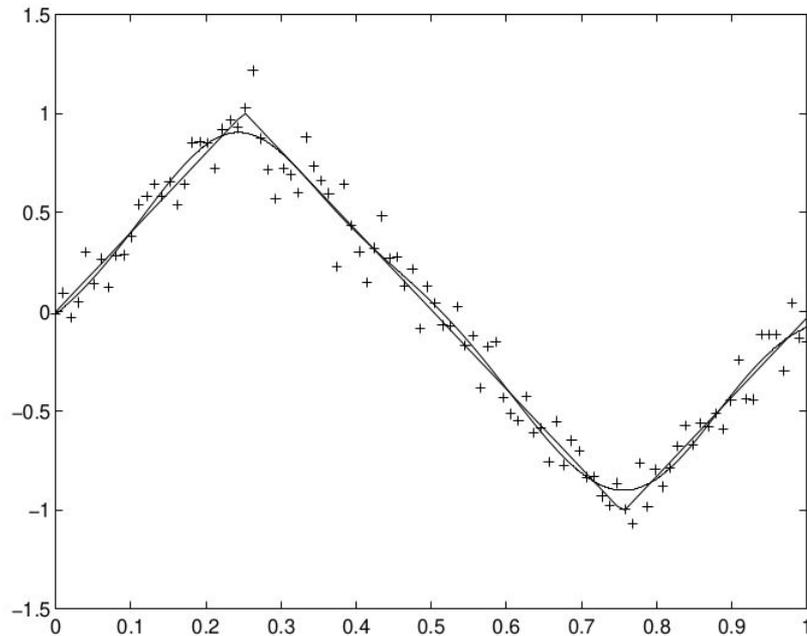
$$\beta = \frac{N - \gamma}{\sum_{n=1}^N \{y_n - f(\mathbf{x}_n; \mathbf{w}_{\text{MAP}})\}^2}$$

# Example of hyperparameter tuning for regression

1-6-1 Network Without Regularization



1-6-1 Network With Bayesian Regularization



# Generalized Gauss-Newton approximation

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

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

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

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

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

where  $\mathbf{J}_n$  is the  $n_{\text{outputs}} \times d$  Jacobian matrix given by  $\mathbf{J}_n = \nabla f(\mathbf{x}_n; \mathbf{w})|_{\mathbf{w}_{\text{MAP}}}$  and

$\mathbf{G}_n$  is the  $n_{\text{outputs}} \times n_{\text{outputs}}$  matrix  $\mathbf{G}_n = \nabla \nabla - \log p(y_n | \hat{f}(\mathbf{x}_n; \mathbf{w}_{\text{MAP}})) \Big|_{\hat{f}(\mathbf{x}_n; \mathbf{w}_{\text{MAP}})} = \beta$

The GGN is **guaranteed to be positive definite** when  $-\log p(y | f(\mathbf{x}; \mathbf{w}))$  is 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  $\alpha$ ,  $\beta$  and network weights  $\mathbf{w}$
2. Train the network using a standard optimization algorithm to obtain  $\mathbf{w}_{\text{MAP}}$
3. Every few cycles of optimization:
  - a. Update  $\mathbf{A} = \mathbf{J}^T \mathbf{G} \mathbf{J} + \alpha \mathbf{I}$  given current estimate value of  $\mathbf{w}$
  - b. Compute  $\gamma$  using eigen-decomposition of  $\mathbf{A}$ , that is,  $\gamma = \sum_{i=1}^d \lambda_i / (\lambda_i + \alpha)$
  - c. Update  $\alpha$  using  $\alpha = \gamma / \mathbf{w}^T \mathbf{w}$
  - d. Update  $\beta$  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  $\gamma$  is close to  $d$ , then network may not be large enough: add more neurons.

If larger network has same final  $\gamma$ , 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

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

where

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

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

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

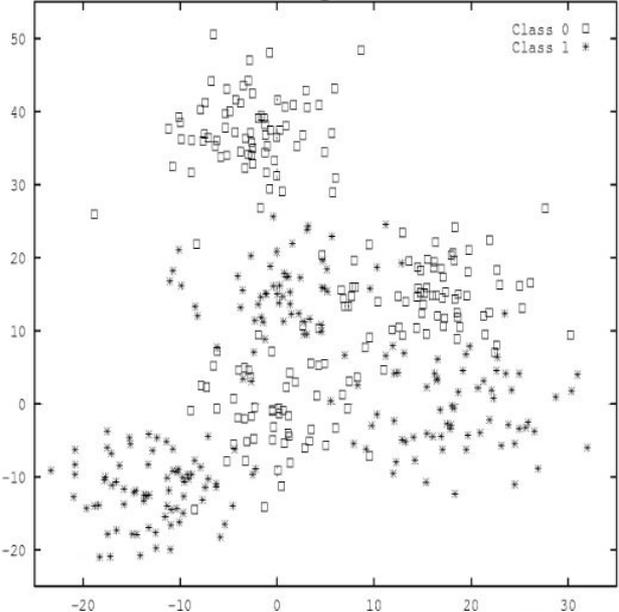
where we use entry-wise vector operations and  $\boldsymbol{\Sigma}(\mathbf{x}_*) = \nabla \mathbf{f}(\mathbf{x}_*; \mathbf{w})|_{\mathbf{w}_{\text{MAP}}}^T \mathbf{A}^{-1} \nabla \mathbf{f}(\mathbf{x}_*; \mathbf{w})|_{\mathbf{w}_{\text{MAP}}}$

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

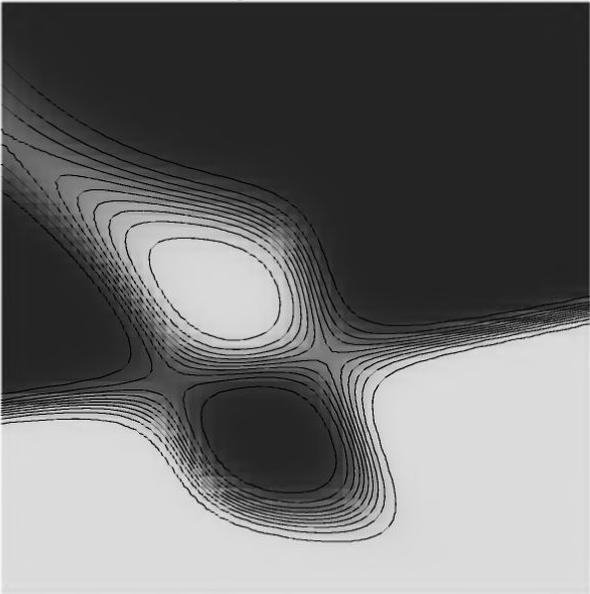
# Classification example

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

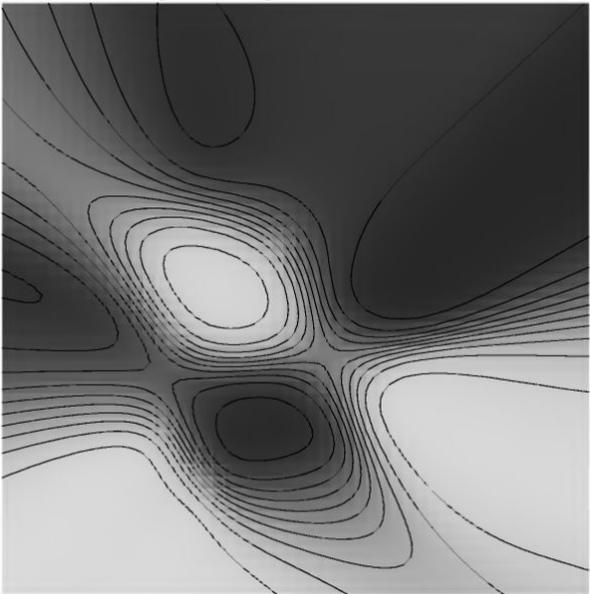
Training Data



MAP predictions



Laplace predictions

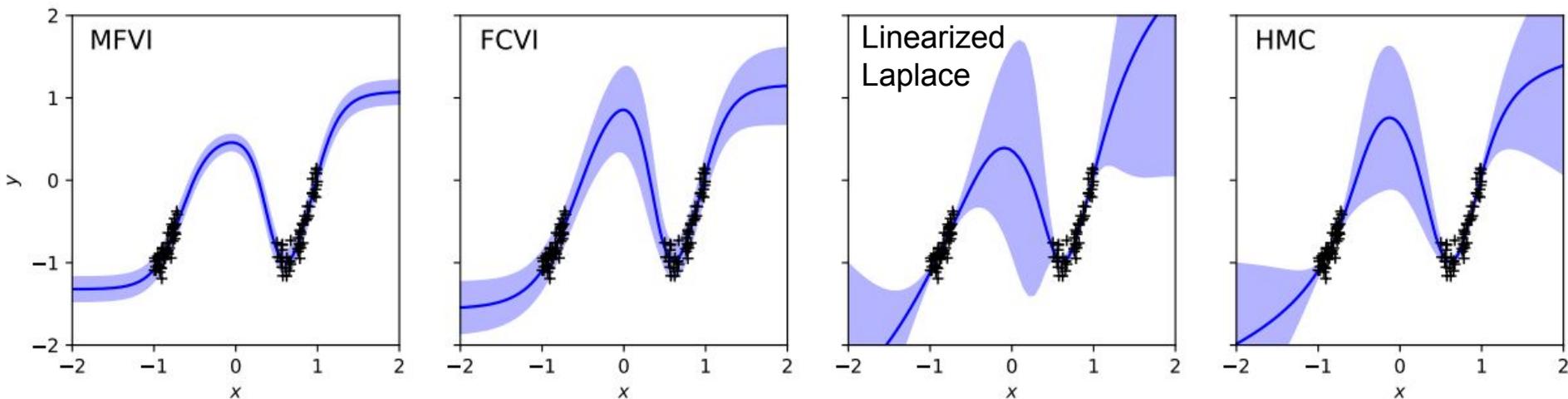


## **Section 4**

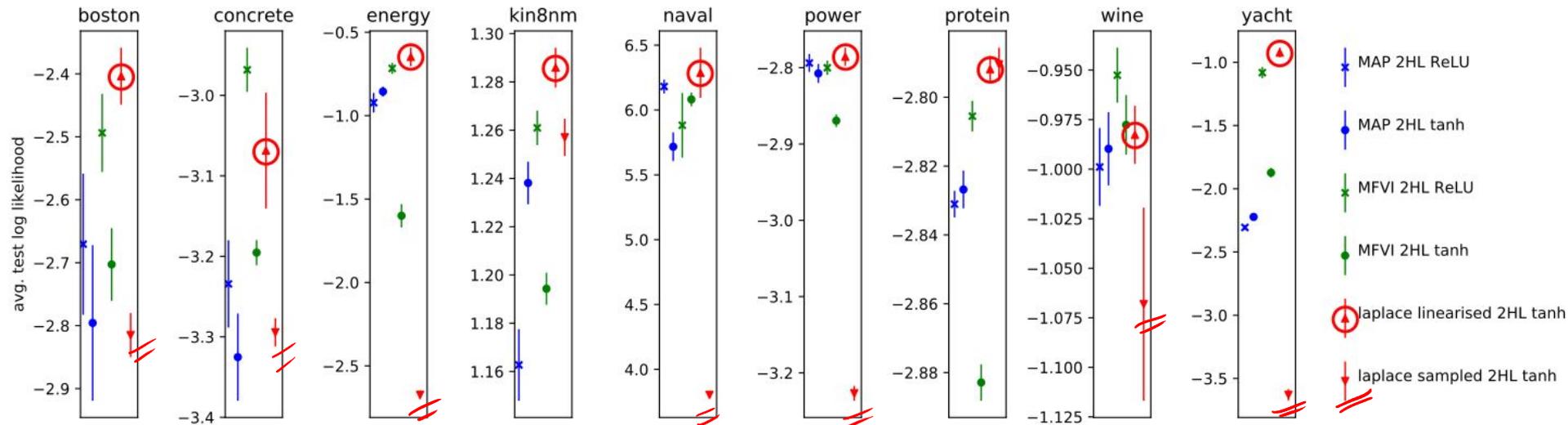
# **More recent works on the Laplace approximation**

# 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

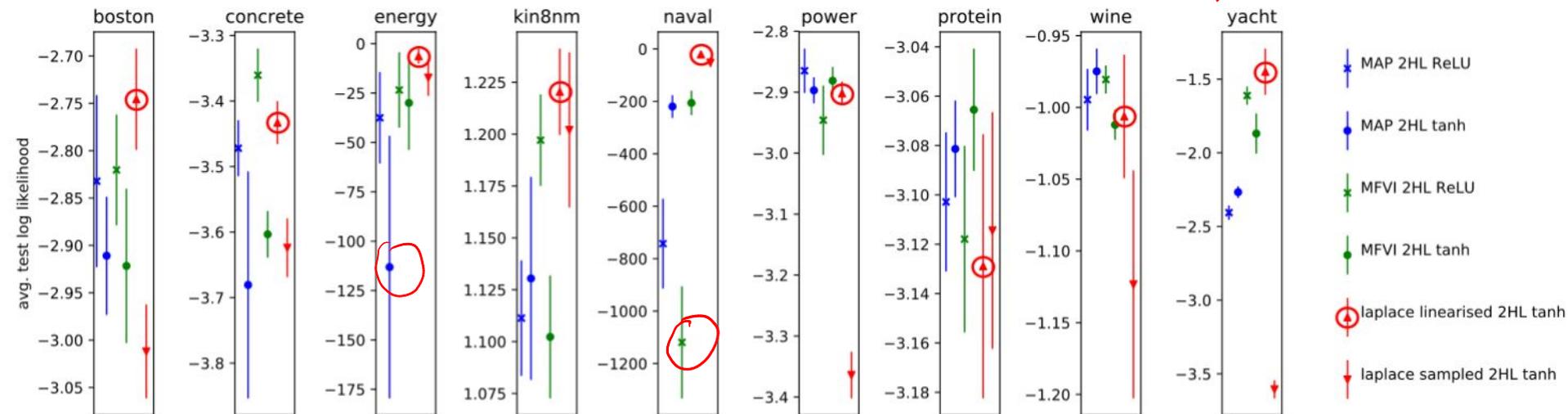
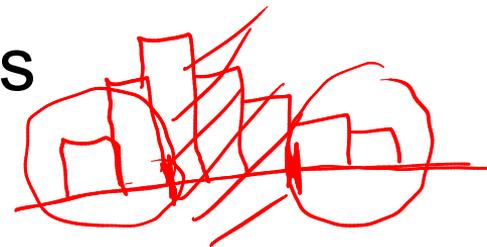


# Results on standard splits of UCI datasets



**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**  $\otimes$ :

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

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

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

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

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

# Scaling up with Kronecker products

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

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix}, \quad (\mathbf{A} \otimes \mathbf{B})^{-1} = \mathbf{A}^{-1} \otimes \mathbf{B}^{-1}, \quad (\mathbf{A} \otimes \mathbf{B})^T = \mathbf{A}^T \otimes \mathbf{B}^T.$$

$$(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = (\mathbf{AC}) \otimes (\mathbf{BD}).$$

$$\text{vec}(\mathbf{A}) = [a_{1,1}, \dots, a_{m,1}, a_{1,2}, \dots, a_{m,2}, \dots, a_{1,n}, \dots, a_{m,n}]^T$$

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

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

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

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

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



# Scaling up with Kronecker products

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

$$\mathbf{G}_\ell^n = \mathbf{B}_\ell \mathbf{W}_{\ell+1}^\top \mathbf{G}_{\ell+1}^n \mathbf{W}_{\ell+1} \mathbf{B}_\ell$$

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

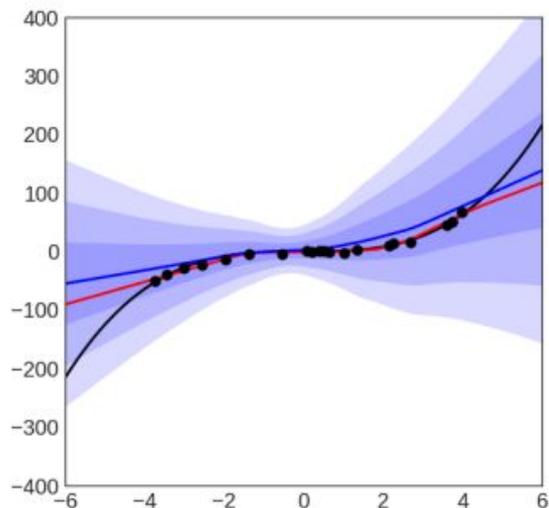
For the whole dataset, the GGN for layer  $\ell$  can be approximated as

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

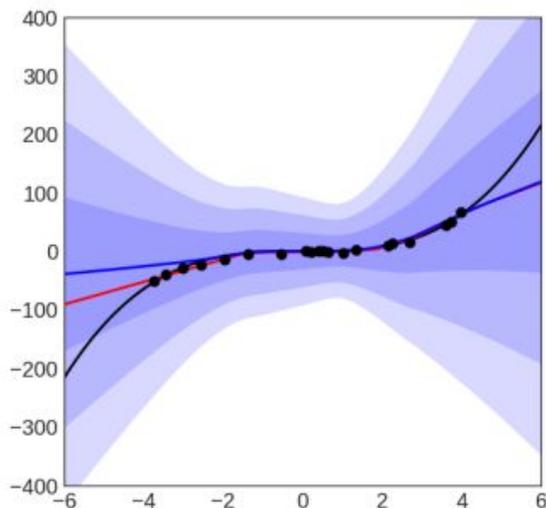

The posterior covariance  $\mathbf{A}$  is then assumed **block diagonal**, one block per layer given by

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

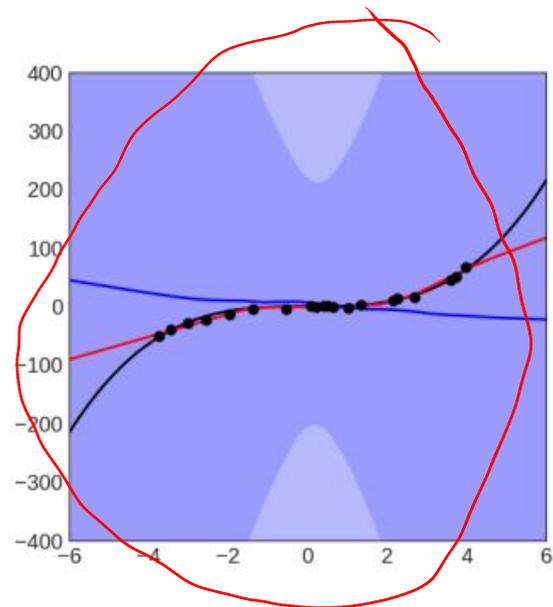
# Results on toy problem



(a) KF Laplace

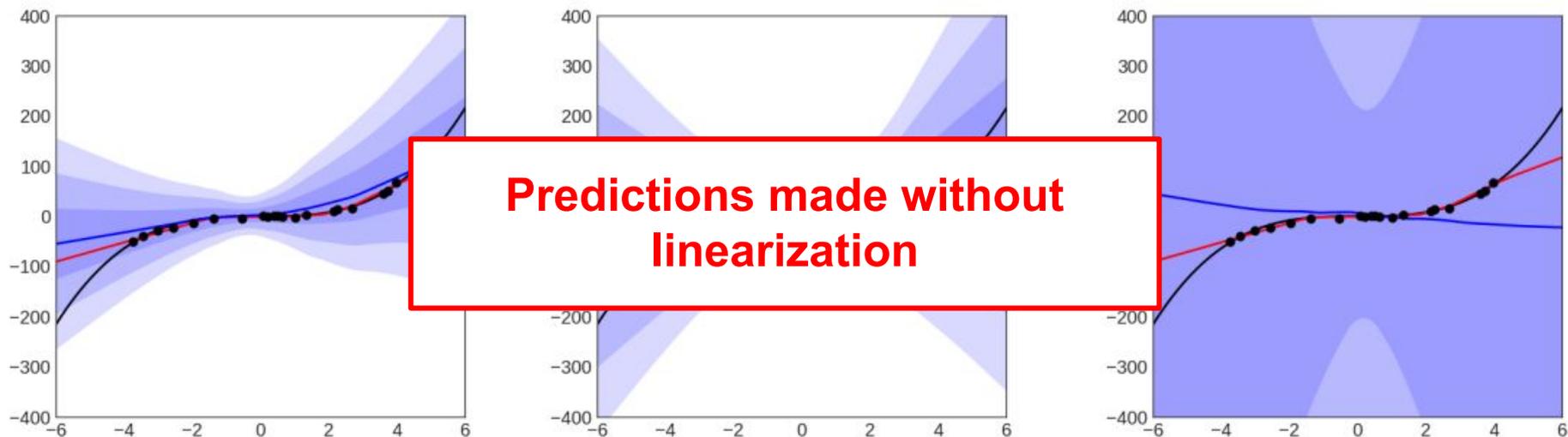


(b) Diagonal Laplace



(c) Full Laplace

# Results on toy problem



(a) KF Laplace

(b) Diagonal Laplace

(c) Full Laplace

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

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 $\uparrow$	NLL $\downarrow$	ECE $\downarrow$	OOD-AUC $\uparrow$
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	<b>92.25<math>\pm</math>0.10</b>	<b>0.244<math>\pm</math>0.003</b>	0.012 $\pm$ 0.003	<b>0.955<math>\pm</math>0.006</b>
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	<b>81.37<math>\pm</math>0.15</b>	0.601 $\pm$ 0.008	0.084 $\pm$ 0.010	<b>0.843<math>\pm</math>0.016</b>

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.

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

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

# 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)

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

**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  $p(\mathbf{W}|\mathbf{y}, \mathbf{X}) \approx \prod_{d=1}^D q(\mathbf{w}_d)$ .

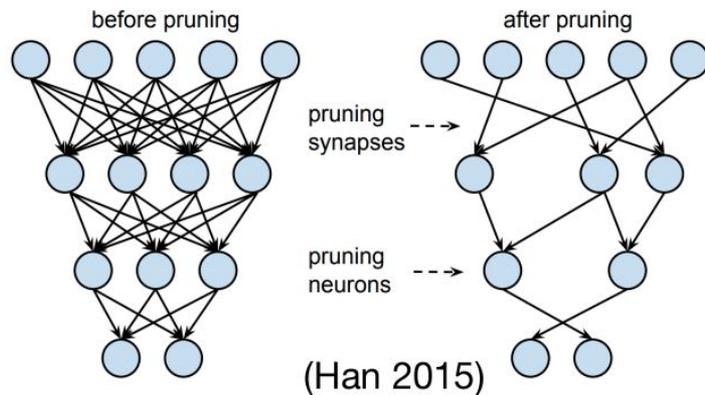
This deteriorates the quality of resulting uncertainty estimates!  
(Ovadia 2019, Fort 2019, Foong 2019, Ashukha 2020)

**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).



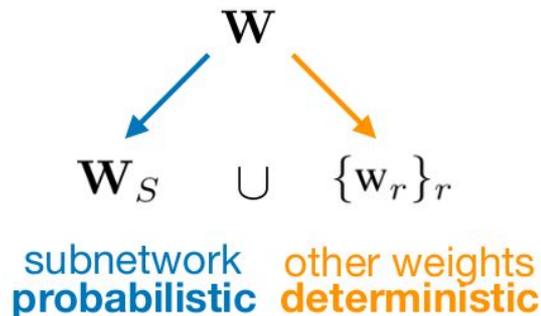
**Can a full DNN's model uncertainty be well-preserved by a small subnetwork's model uncertainty?**

**Answer:** Yes!

# Subnetwork inference

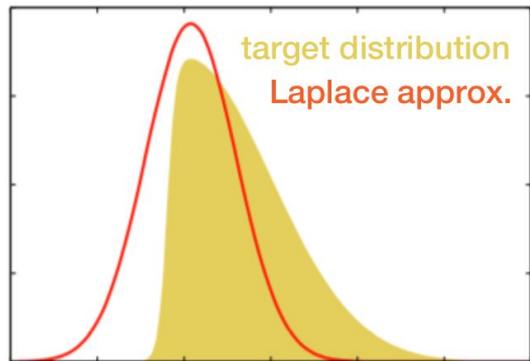
Proposed posterior approximation:

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



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

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

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

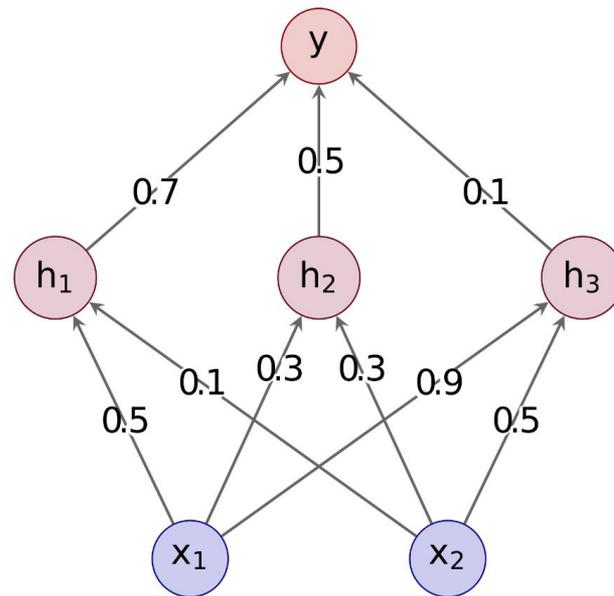
$$p(\mathbf{W}|\mathbf{y}, \mathbf{X}) \approx q(\mathbf{W}) = \mathcal{N}(\mathbf{W}; \mathbf{W}_{MAP}, H^{-1})$$

$H$  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

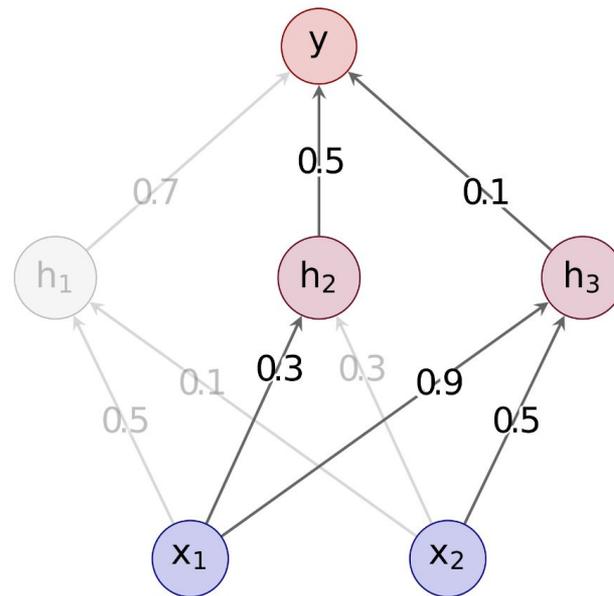
## 1 MAP Estimation



# Subnetwork inference

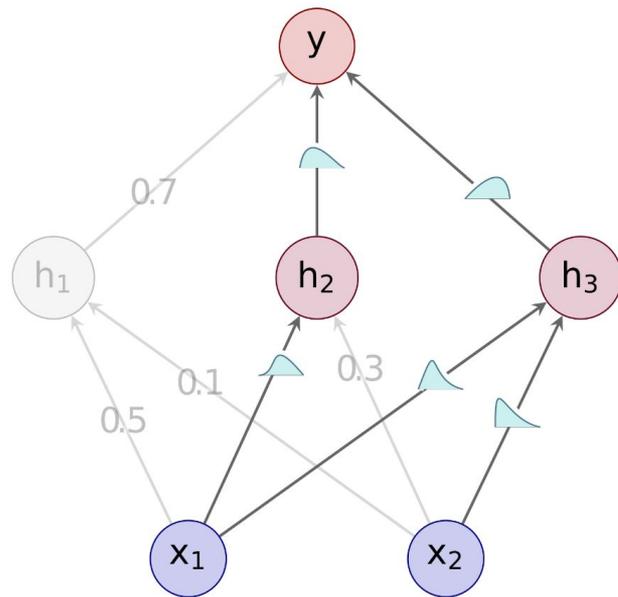
1 MAP Estimation

2 Subnet Selection



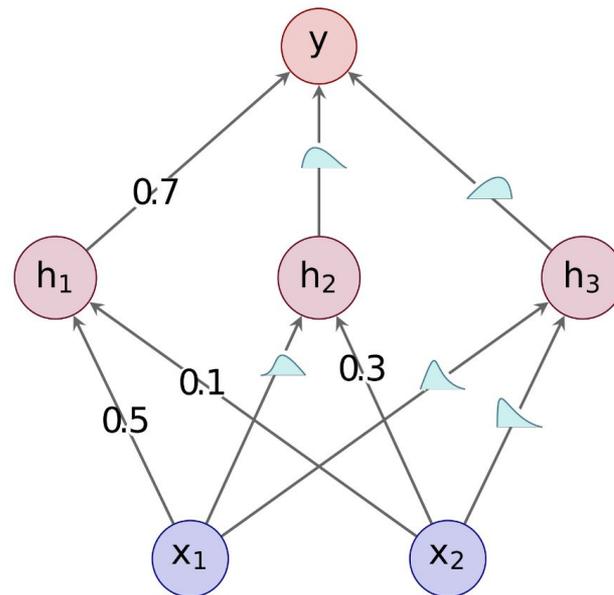
# Subnetwork inference

- 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.

$$\begin{aligned} & \min \text{Wass} [ \text{full posterior} \parallel \text{subnet posterior} ] \\ &= \min \text{Wass} [ p(\mathbf{W} | \mathbf{y}, \mathbf{X}) \parallel q(\mathbf{W}) ] \\ &\approx \min \text{Wass} [ \mathcal{N}(\mathbf{W}; \mathbf{W}_{MAP}, H^{-1}) \parallel \mathcal{N}(\mathbf{W}_S; \mathbf{W}_{MAP}^S, H_S^{-1}) \prod_r \delta(w_r - w_{MAP}^r) ] \end{aligned}$$

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

**Assume that posterior factorizes:  $H$  is diagonal.**

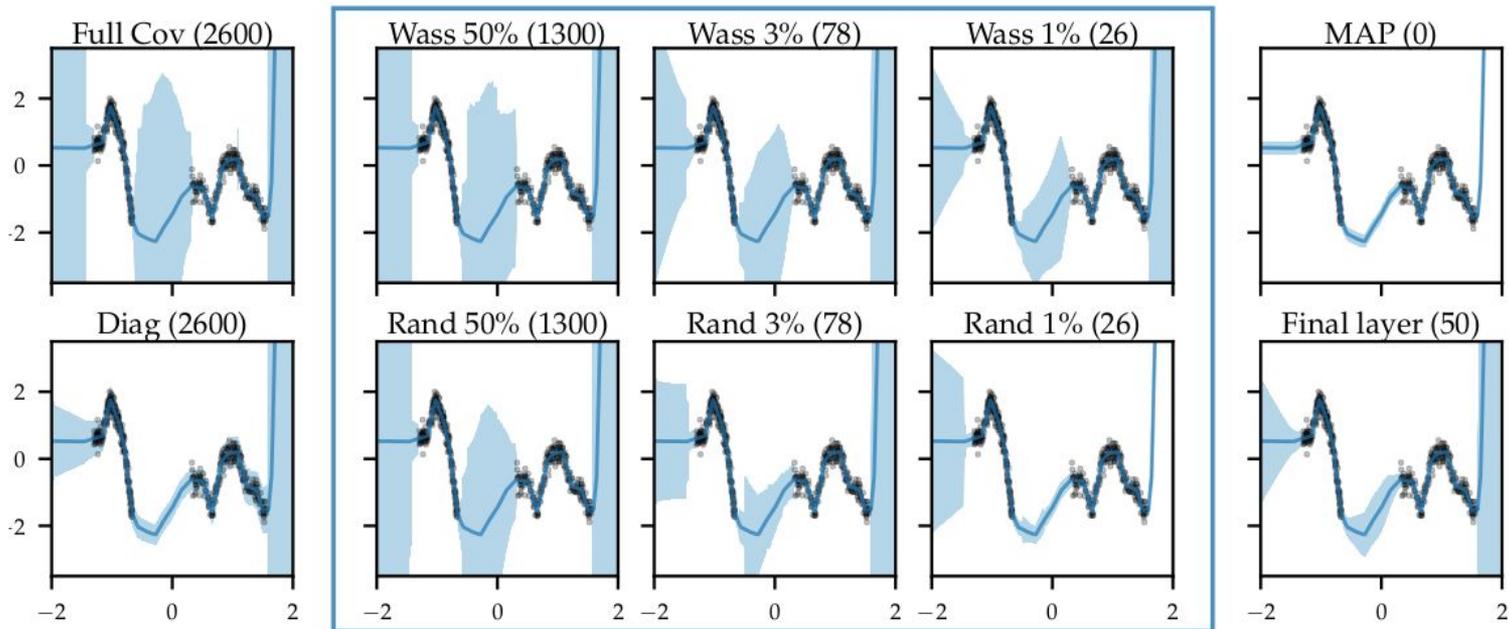
**Diagonal assumption for subnetwork selection  $\gg$  diag. assumption for inference**

**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).



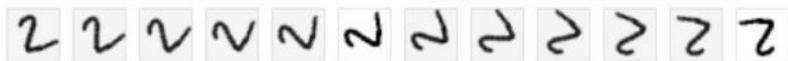
Expressive inference over a small subnetwork preserves **more uncertainty** than crude inference over the full network!

# Image classification under distribution shift

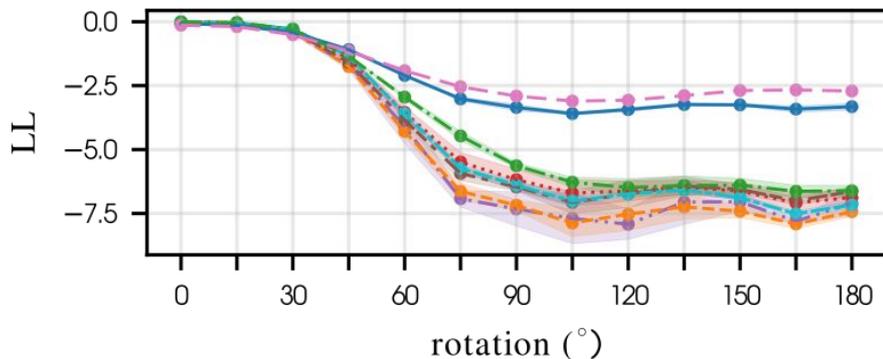
**Model:** ResNet-18 with 11M weights. **Subnetwork:** just 42K (0.38%) weights.

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

**Rotated MNIST** (Ovadia 2019)



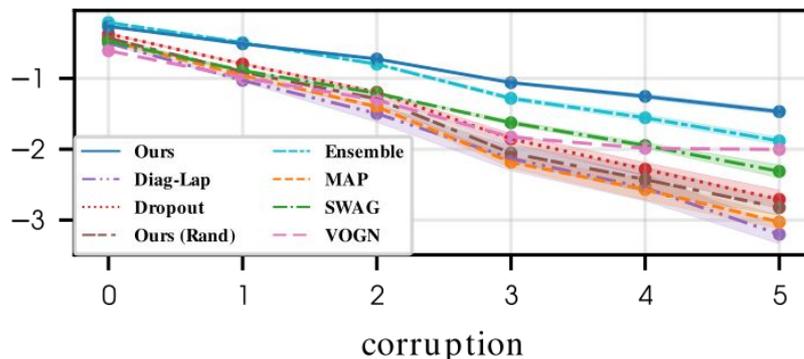
**Rotated MNIST**



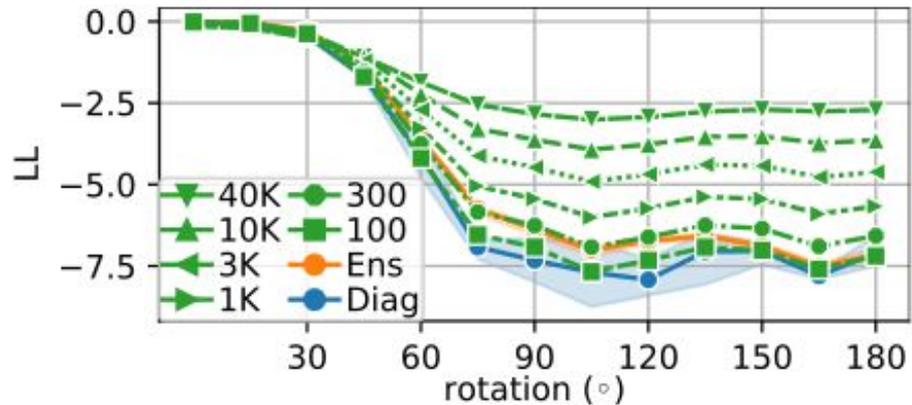
**Corrupted CIFAR10** (Ovadia 2019)



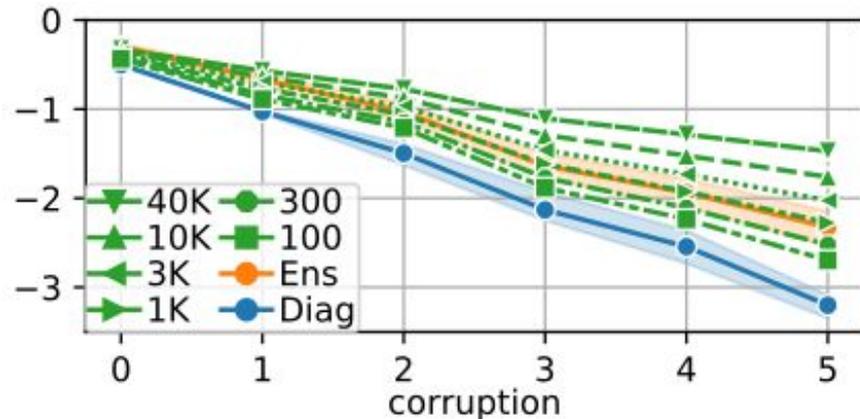
**Corrupted CIFAR10**



# 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

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



Javier Antoran



David Janz



James Allingham



Erik Daxberger



Riccardo Barbano



Eric Nalisnick

# Review of what we have seen so far

- A Gaussian can be a **poor approximation** to the neural network posterior,
- but **a very good one** for the posterior of the **linearized** neural network model

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

- since the corresponding log posterior

$$\sum_{n=1}^N \log p(y_n | \hat{f}(\mathbf{x}_n; \mathbf{w})) - \frac{\alpha}{2} \mathbf{w}^{\text{T}} \mathbf{w} + \frac{d}{2} \log \alpha$$

is **concave** in most cases and even Gaussian when the likelihood is Gaussian.

# Hyper-parameter tuning in subnetwork inference

After finding  $\mathbf{w}_{\text{MAP}}$ , tuning the prior precision  $\alpha$  is **key for accurate error bars** with the linearized Laplace approximation.

- How did you **tune such hyper-parameters** in subnetwork inference?

We used cross validation!

- Why not optimizing the **model evidence** instead?

It did not work. **The chosen prior precision over-estimated uncertainty.**

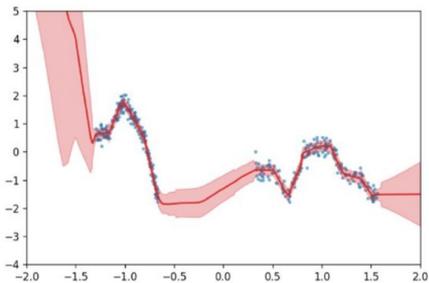
# Example

Neural network with 2 hidden layers, 2600 parameters and batchnorm.

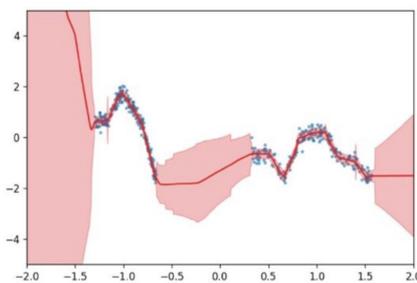
More  
reasonable  
solution



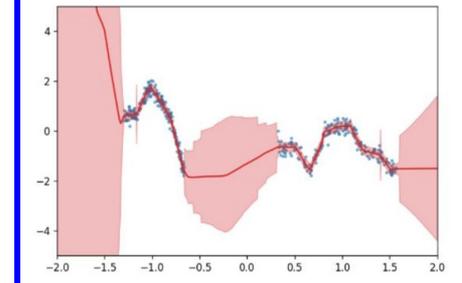
$\alpha = 100$



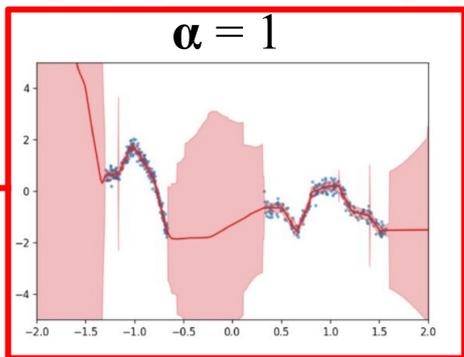
$\alpha = 10$



$\alpha = 5$



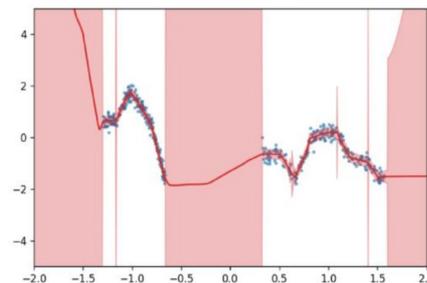
$\alpha = 1$



Largest  
model  
evidence



$\alpha = 0.1$



**Why is optimizing the approximation to the model evidence not working well?**

# Why is that David Mackay did not have this problem?

Modern neural networks are often trained using **stochastic optimization, batchnorm, early stopping, etc.**

**They prevent finding a posterior mode** and were not used by Mackay. He used 2nd order batch optimizers.

If the output  $\mathbf{w}_{\text{MAP}}$  of the optimizer is a posterior mode, that is,  $\nabla \log p(\mathbf{w}|\mathbf{y}, \mathbf{X}, f(\mathbf{x}; \mathbf{w}))|_{\mathbf{w}_{\text{MAP}}} = 0$  then

$$\nabla \log p(\mathbf{w}|\mathbf{y}, \mathbf{X}, \hat{f}(\mathbf{x}; \mathbf{w})) \Big|_{\mathbf{w}_{\text{MAP}}} = 0$$

meaning that  $\mathbf{w}_{\text{MAP}}$  is **also a posterior mode of the linearized model!**

Furthermore, since the Laplace approximation of the model evidence in the linearized model is given by

$$\log p(\mathbf{y}|\mathbf{X}) \approx \sum_{n=1}^n \log p(y_n | \hat{f}(\mathbf{x}_n; \mathbf{w}_{\text{MAP}})) - \log p(\mathbf{w}_{\text{MAP}}|\alpha) - \frac{1}{2} \log |\mathbf{A}|$$

Both the linearized model and the original neural network have **the same evidence approximation!**

But this no longer the case when the linearization point  $\mathbf{w}_{\text{MAP}}$  is not a posterior mode

# When the linearization point $\mathbf{w}_{\text{MAP}}$ is not a mode

How to tune hyper-parameters in this case?

## Proposed solution:

Optimize the Laplace approximation of the model evidence **for the linear model**, that is,

$$\log p(\mathbf{y}|\mathbf{X}) \approx \sum_{n=1}^n \log p(y_n | \hat{f}(\mathbf{x}_n; \mathbf{w}')) - \log p(\mathbf{w}'|\alpha) - \frac{1}{2} \log |\mathbf{A}| \quad \text{where } \mathbf{w}' = \underset{\mathbf{w}}{\text{argmax}} \log p(\mathbf{w}|\mathbf{y}, \mathbf{X}, \hat{f}(\mathbf{x}; \mathbf{w}))$$

and  $\mathbf{A}$  is the GGN matrix given by

$$\mathbf{A} = \sum_{n=1}^N \mathbf{J}_n^T \mathbf{G}_n \mathbf{J}_n + \alpha \mathbf{I} \quad \text{where } \mathbf{J}_n = \nabla f(\mathbf{x}_n; \mathbf{w})|_{\mathbf{w}_{\text{MAP}}} \quad \text{and} \quad \mathbf{G}_n = \nabla \nabla - \log p(y_n | \hat{f}(\mathbf{x}_n; \mathbf{w}')) \Big|_{\hat{f}(\mathbf{x}_n; \mathbf{w}')}$$

To avoid recomputing  $\mathbf{A}$  when  $\mathbf{w}'$  changes, we assume  $\mathbf{w}'$  is equal to  $\mathbf{w}_{\text{MAP}}$  when evaluating  $\mathbf{G}_n$ .

The latter approximation is **exact** when the likelihood is Gaussian.

# How to obtain the mode $\mathbf{w}'$ of the linearized model?

Efficient Gradient based optimization of log posterior possible **without explicit Jacobian computations**.

E.g., using **finite differences** or **forward differentiation** to approximate Jacobian vector products.

---

**Algorithm 1:** Efficient evaluation of the likelihood gradient for the linearised model

---

**Inputs:** Neural network  $f$ , Observation  $\mathbf{x}$ , Linearisation point  $\mathbf{w}_{\text{MAP}}$ , Weights to optimise  $\mathbf{w}$ , Log-likelihood function  $\ell(\cdot, y)$ , Machine precision  $\epsilon$

```
1  $\delta = \sqrt{\epsilon}(1 + \|\mathbf{w}_{\text{MAP}}\|_{\infty})/\|\mathbf{w}\|_{\infty}$  // Set finite diff. stepsize (Andrei, 2009)
2  $\hat{y} = J(x) \cdot \mathbf{w} \approx \frac{f(x, \mathbf{w}_{\text{MAP}} + \delta \mathbf{w}) - f(x, \mathbf{w}_{\text{MAP}} - \delta \mathbf{w})}{2\delta}$  // Two sided finite diff. approx. to Jvp
3  $g = \ell'(J(x) \cdot \mathbf{w}, y)$  // Evaluate gradient of loss at  $J(x) \cdot \mathbf{w}$ 
4  $g^T \cdot J(x) = \partial_{\mathbf{w}}[g^T \cdot f(\mathbf{w}, x)](\mathbf{w}_{\text{MAP}})$  // Project gradient with backward mode AD
```

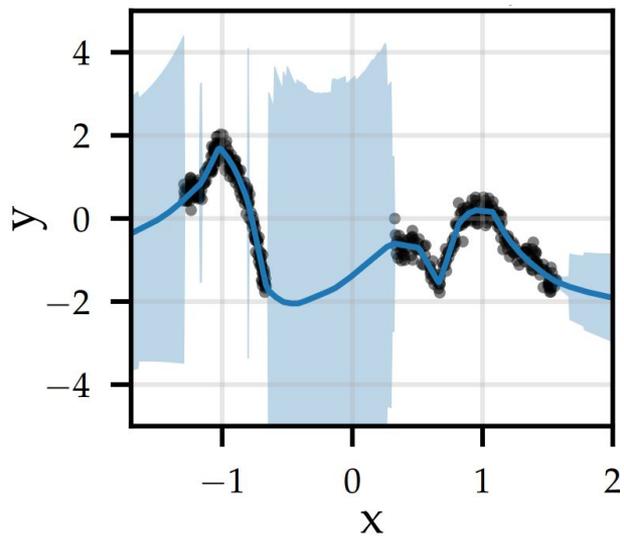
**Output:**  $g^T \cdot J(x)$

---

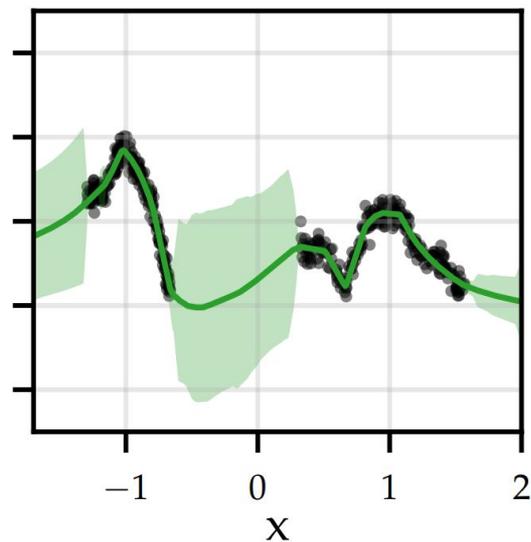
**Computational cost:** same as two forward passes and one backward pass in the original network.

# Results on toy problem

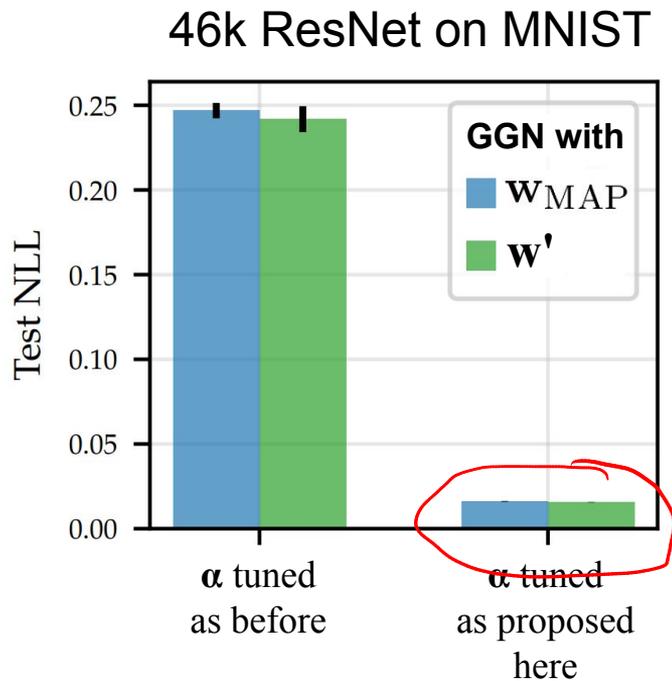
$\alpha$  tuned as before



$\alpha$  tuned with the proposed method



# What about approximating $w'$ with $w_{MAP}$ in the GGN?



# Networks with normalization layers

Let  $\mathbf{w} = \mathbf{w}_a + \mathbf{w}_b$  with  $\mathbf{w}_a$  having zero entries in those dimensions subject to normalization, while  $\mathbf{w}_b$  has zero entries for the other dimensions.

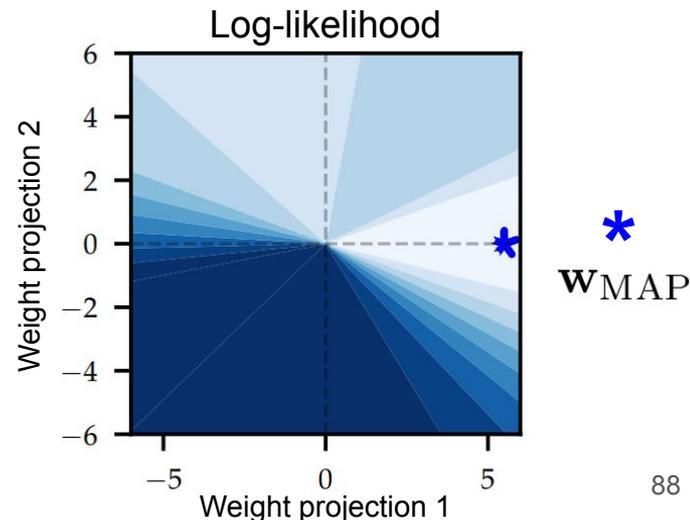
**Examples:** networks with batch norm, layer norm, group norm, etc.

**Normalization introduces scale invariance:**

$$f(\mathbf{x}; \mathbf{w}_a + \mathbf{w}_b) = f(\mathbf{x}; \mathbf{w}_a + k\mathbf{w}_b) \quad \text{for } k > 0$$

Linearization point  $\mathbf{w}_{\text{MAP}}$  can never be a mode.

**There are no posterior modes!**

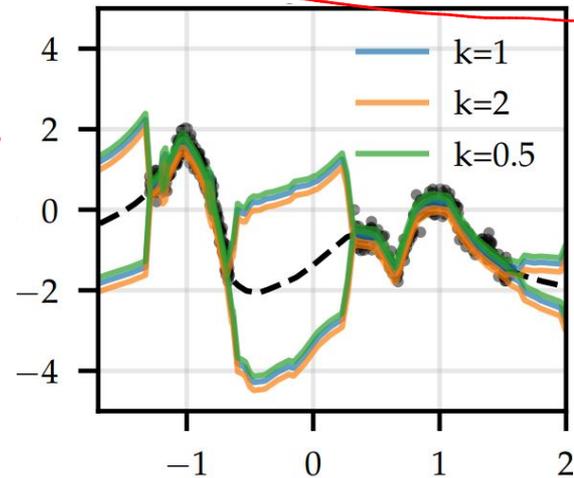
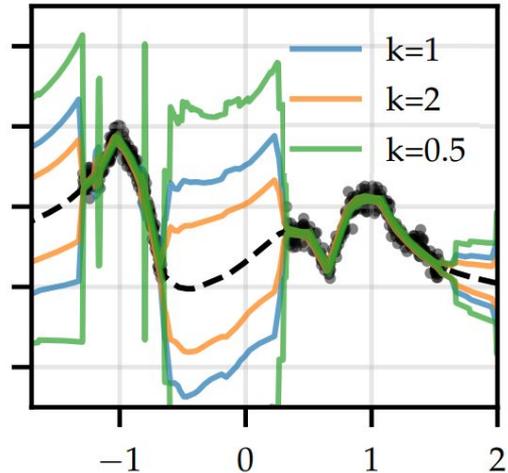


# Effect of normalized layers on predictive uncertainty

$k$  does not affect predictions and it should not affect the predictive uncertainty.

**But it does** since  $k$  scales the Jacobian features used by the linearized model.

We can solve the problem by using different  $\alpha$  for each group of normalized weights.



# Results 46k parameter models

Negative test log-likelihood across different problems and models.

		TRANSFORMER	CNN	RESNET	PRE-RESNET	FIXUP	U-NET
Now	single $\alpha$	<b>0.162</b> $\pm$ 0.042	<b>0.025</b> $\pm$ 0.000	<b>0.017</b> $\pm$ 0.000	0.017 $\pm$ 0.000	<b>0.055</b> $\pm$ 0.006	-1.793 $\pm$ 0.050
	layerwise $\alpha$	<b>0.162</b> $\pm$ 0.042	<b>0.025</b> $\pm$ 0.000	<b>0.016</b> $\pm$ 0.001	<u>0.016</u> $\pm$ 0.000	0.061 $\pm$ 0.005	<u>-2.240</u> $\pm$ 0.027
Before	single $\alpha$	0.310 $\pm$ 0.060	0.253 $\pm$ 0.001	0.252 $\pm$ 0.006	<u>0.220</u> $\pm$ 0.004	<u>0.153</u> $\pm$ 0.021	-1.164 $\pm$ 0.052
	layerwise $\alpha$	<u>0.162</u> $\pm$ 0.042	<u>0.205</u> $\pm$ 0.002	<u>0.236</u> $\pm$ 0.005	0.239 $\pm$ 0.004	0.200 $\pm$ 0.018	<u>-1.703</u> $\pm$ 0.023

Using  $w'$  to optimize model evidence approximation and multiple  $\alpha$ 's works best.

**FixUp** is a non-scale invariant alternative to normalization layers.

# A simplified linear model

So far we have used a linearized model of the following form:

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

If the last layer is linear, the linear span of  $\nabla f(\mathbf{x}; \mathbf{w})|_{\mathbf{w}_{\text{MAP}}}$  contains  $f(\mathbf{x}; \mathbf{w}_{\text{MAP}})$ .

This justifies using the following **simplified linear model** for uncertainty estimation:

$$f(\mathbf{x}; \mathbf{w}) \approx \mathbf{w}^T \nabla f(\mathbf{x}; \mathbf{w})|_{\mathbf{w}_{\text{MAP}}}$$

The two linear approximations above **are equivalent in normalized networks!**

# Results for ResNet 50 on CIFAR 10 (23M weights)

Negative test log-likelihood values

		BATCH NORM	FIXUP
Now, using simplified linear model	single $\alpha$	<b>0.112</b> $\pm 0.004$	0.128 $\pm 0.000$
	layerwise $\alpha$	<b>0.109</b> $\pm 0.003$	<b>0.096</b> $\pm 0.000$
Now	single $\alpha$	0.190 $\pm 0.005$	0.249 $\pm 0.002$
	layerwise $\alpha$	0.194 $\pm 0.009$	<u>0.193</u> $\pm 0.001$
Before	single $\alpha$	0.570 $\pm 0.004$	0.412 $\pm 0.000$
	layerwise $\alpha$	0.567 $\pm 0.004$	<u>0.360</u> $\pm 0.000$

We use kroncker product approximations for scalable computation.

The simplified linear model works best.

Using  $w'$  to optimize model evidence approximation and multiple  $\alpha$ 's works best.

# Take-home messages

There are pitfalls in the naïve application of linearised Laplace to modern NNs

- **Training often does not find a posterior mode**, but we can still linearize the network around the resulting weight values.
- For accurate uncertainties, **prior variances should be tuned using the linear model evidence**. We have proposed an **efficient algorithm** for this.
- Normalisation layers introduce invariance to weight scale. To obtain uncertainties that are also invariant to this scaling, **we should use different prior variances for each normalised group of weights**, e.g. different layers.
- A **simplified linear model** on the Jacobian features produces better uncertainties.

# Section 7

## Case study: X-ray image reconstruction

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



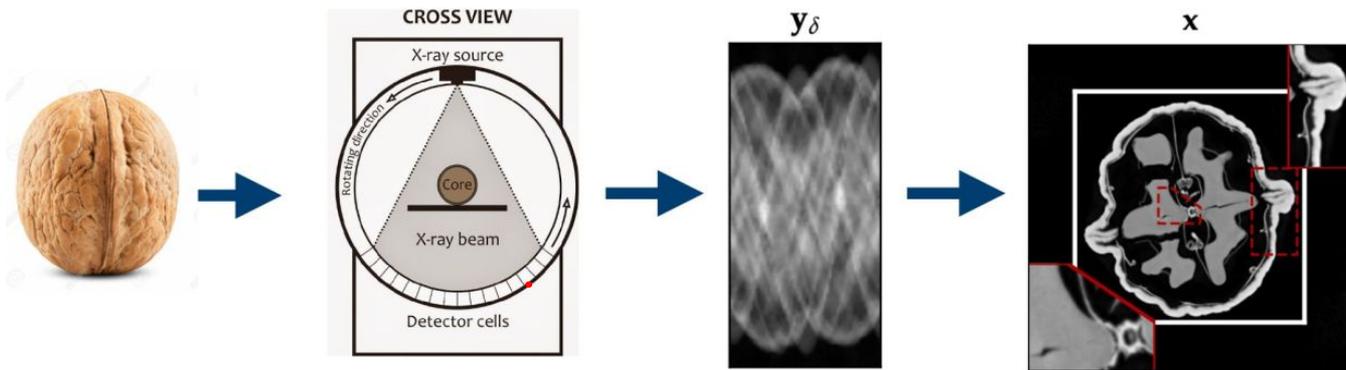
Bangti Jin

# Inverse imaging problem

Aims at recovering an unknown image  $\mathbf{x} \in \mathbb{R}^{d_x}$  from noisy measurements  $\mathbf{y}_\delta \in \mathbb{R}^{d_y}$ :

$$\mathbf{y}_\delta = \mathbf{A}\mathbf{x} + \boldsymbol{\eta},$$

where  $\mathbf{A} \in \mathbb{R}^{d_y \times d_x}$  is a linear forward operator and  $\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, \sigma_y^2 \mathbf{I})$ .



- Ill posed problem  $d_y \ll d_x$  solved typically with **regularized reconstruction**.
- Can we solve this task obtaining **accurate uncertainties** for  $\mathbf{x}$ ?

# Regularized reconstruction with the **deep image prior**

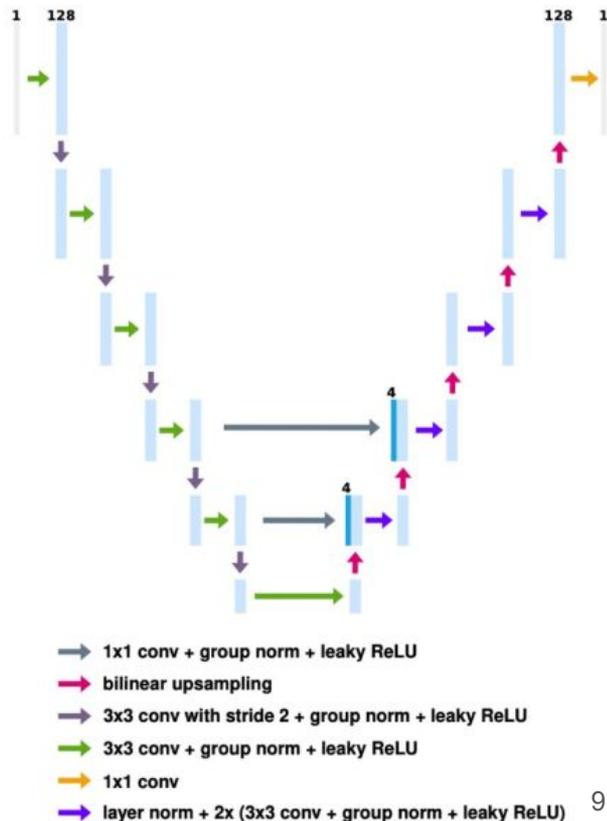
- Assumes **the image is the output of a neural network (U-Net)** with fixed input (omitted below).
- Reparameterizes the image in terms of the network weights  $\mathbf{w}$ , which need to be estimated.
- Weight values obtained by solving

$$\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w} \in \mathbb{R}^{d_{\mathbf{w}}}} \|\mathbf{A}f(\mathbf{w}) - \mathbf{y}_\delta\|^2 + \lambda \operatorname{TV}(f(\mathbf{w}))$$

where the **total variation** regularizer is given by

$$\operatorname{TV}(f(\mathbf{w})) = \sum_{i,j} |f_{i,j} - f_{i+1,j}| + \sum_{i,j} |f_{i,j} - f_{i,j+1}|$$

- Can we obtain uncertainties using linearized Laplace?



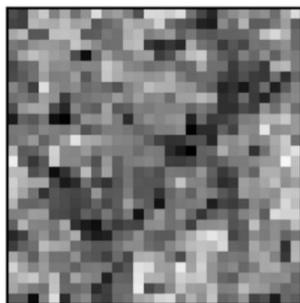
# Approximating the TV with a Gaussian prior

- We cannot apply linearized Laplace with a prior like  $p(f(\mathbf{w})) \propto \exp\{-\lambda \text{TV}(f(\mathbf{w}))\}$ .
- Can we use a similar Gaussian prior operating on  $\mathbf{w}$ ?
- Our Gaussian prior should have a **covariance** structure that enforces **low TV values**:

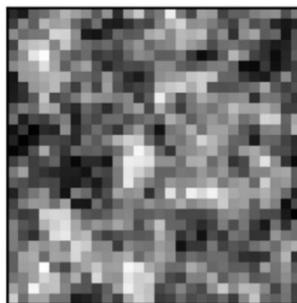
$$p(\mathbf{w}|\ell, \sigma^2) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \Sigma(\ell, \sigma^2)) \quad \Sigma(\ell, \sigma^2)_{kij, k' i' j'} = \sigma_d^2 \exp\left\{\frac{-d(i-i', j-j')}{\ell_d}\right\} \delta_{kk'}$$

$$\kappa_d = \mathbf{E}_{p(\mathbf{w}_d|\ell, \sigma^2)} \prod_{i=1, i \neq d}^D \delta(\mathbf{w}_i) [\text{TV}(f(\mathbf{w}))] \quad p(\ell) = \prod_{d=1}^D \exp\{\lambda \kappa_d\} \left| \frac{\partial \kappa_d}{\partial \ell_d} \right|$$

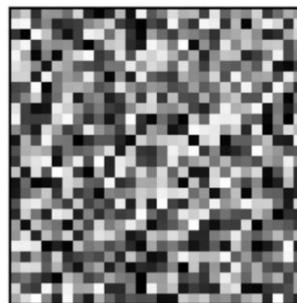
TV



Our prior



Fact. Gauss.

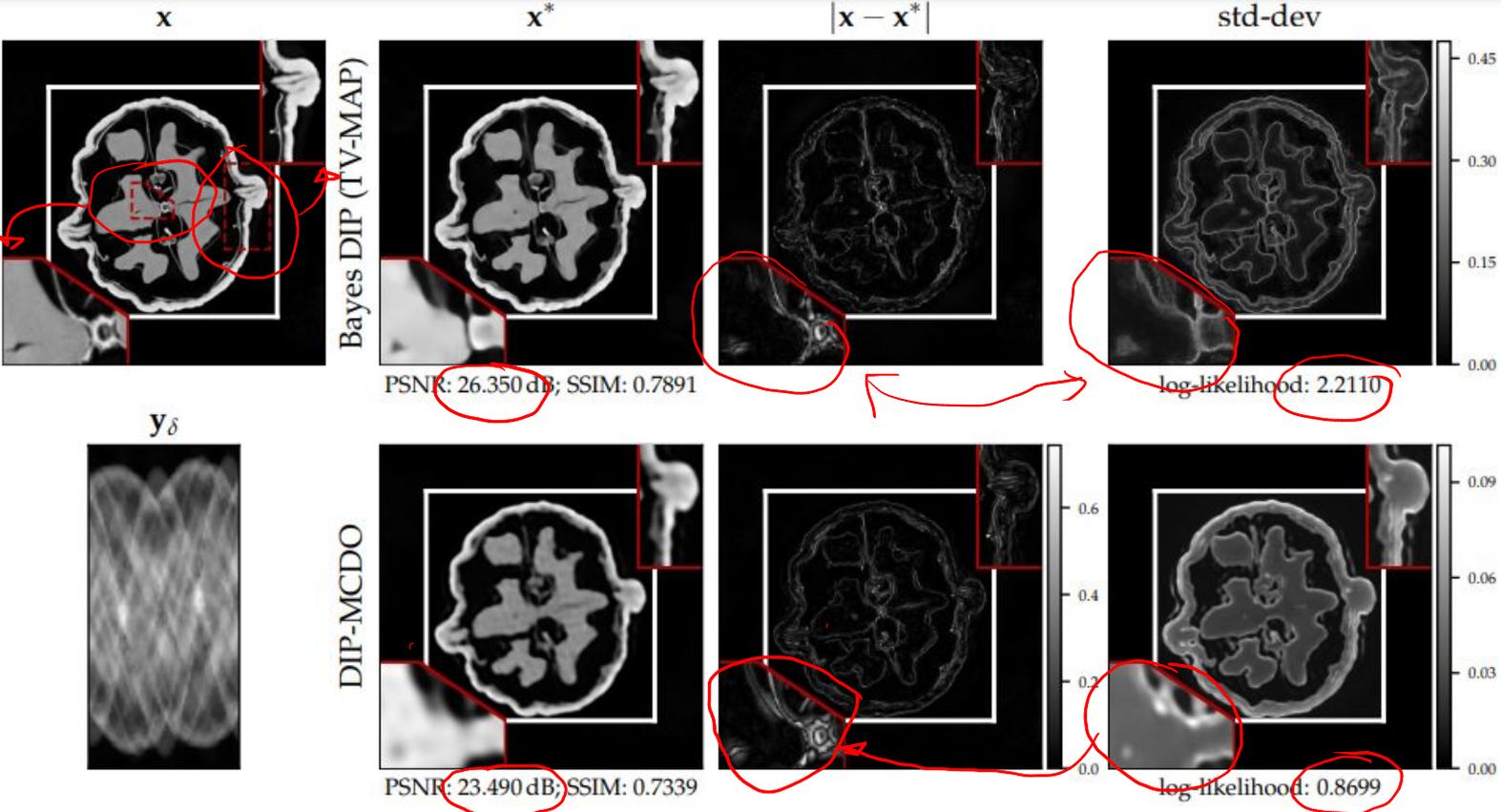


# Procedure for a probabilistic Deep Image prior

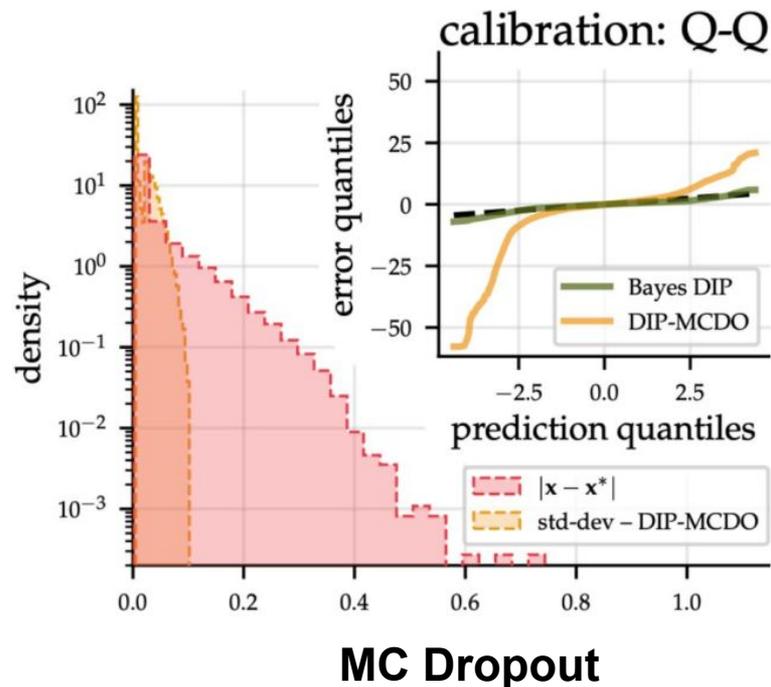
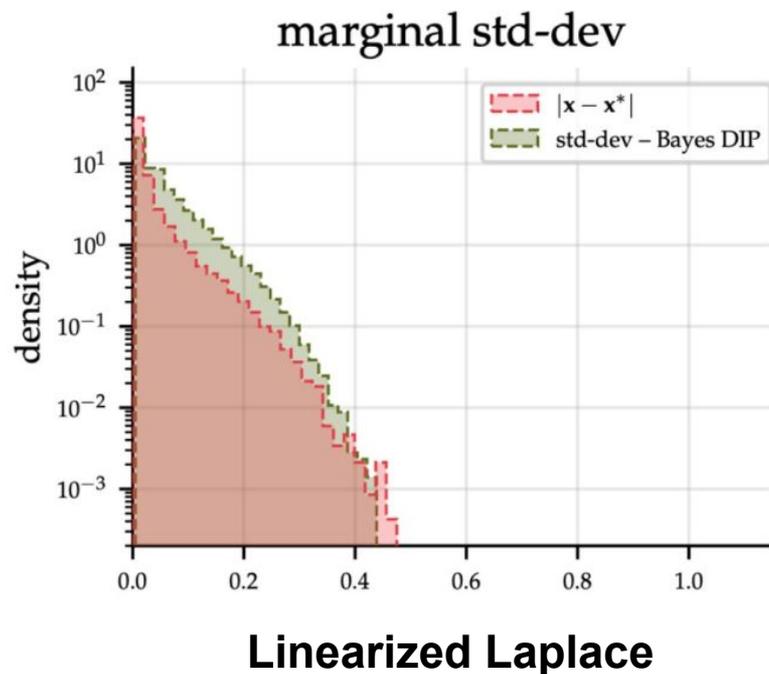
1. Train the U-Net to minimize reconstruction error with TV regularization.
2. Linearize the U-Net around the selected weight values.
3. Build a Bayesian model combining the linearized U-Net with our Gaussian prior.
4. Optimize prior hyper-parameters by maximizing the model evidence plus  $\log p(\ell)$ .
5. Make predictions with original U-Net and obtain uncertainties with Bayesian model.

The arxiv paper describes several technical details on how to do efficient computations.

# Results



# Calibration comparison



# Take-home messages

We have proposed a method that

- Uses linearized Laplace to obtain a probabilistic formulation of the DIP.
- Is based on Gaussian priors that induce a similar effect as the TV regularizer.
- Yields well-calibrated uncertainties in tomographic reconstruction tasks.
- Outperforms existing methods for uncertainty estimation in DIP (MC-dropout).

**Thanks!**