On Weight-Space Issues in Bayesian Neural Networks

Jixiang Qing

LAI, February 11, 2026

BNN: The promise and pitfalls of BNN

- "Bayesian Neural Network **will be solved** in one or two years"

  – Some well-recognized researcher in the field, 2021(ish)

BNN: The promise and pitfalls of BNN

- "Bayesian Neural Network **will be solved** in one or two years"

  – Some well-recognized researcher in the field, 2021(ish)

- "Bayesian Deep Learning is **dead**"

  – Some well-recognized researcher in Lancaster, 2025

BNN: The promise and pitfalls of BNN

- "Bayesian Neural Network **will be solved** in one or two years"

  – Some well-recognized researcher in the field, 2021(ish)

- "Bayesian Deep Learning is **dead**"

  – Some well-recognized researcher in Lancaster, 2025

- "Is Bayesian Neural Network still a thing?"

  – Bayesian Optimisation Workshops, 2025

BNN: The promise and pitfalls of BNN

- "Bayesian Neural Network **will be solved** in one or two years"

  – Some well-recognized researcher in the field, 2021(ish)

- "Bayesian Deep Learning is **dead**"

  – Some well-recognized researcher in Lancaster, 2025

- "Is Bayesian Neural Network still a thing?"

  – Bayesian Optimisation Workshops, 2025

- "Do Bayesian Neural Networks **Actually Behave** Like **Bayesian** Models?"

  – Tom Rainforth and his colleagues, 2025

Outline

Part1: A review of BNN (Bayesian Scheme, Approximation Inference and Parameterization)

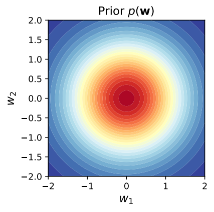
Part2: A Geometric View & Approach Enhance BNN's Performance in Weight Space

Part1: A review of BNN (Bayesian Scheme, Approximation Inference and Parameterization)

The Bayesian Scheme

Model (Generative Assumption):

- Prior: $\boldsymbol{w} \sim p(\boldsymbol{w})$ (e.g., $\boldsymbol{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$)
- Likelihood: $y_i|\boldsymbol{x}_i, \boldsymbol{w} \sim p(y|f_{\boldsymbol{w}}(\boldsymbol{x}_i))$
  (e.g., $f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}, \quad y_i \sim \mathcal{N}(\boldsymbol{w}^\top \boldsymbol{x}_i, \sigma^2)$)

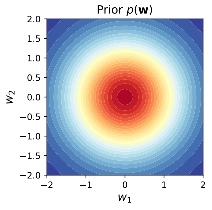


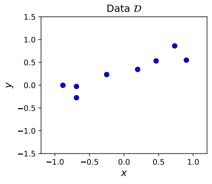
The Bayesian Scheme

Model (Generative Assumption):

- Prior: $\boldsymbol{w} \sim p(\boldsymbol{w})$ (e.g., $\boldsymbol{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$)
- Likelihood: $y_i | \boldsymbol{x}_i, \boldsymbol{w} \sim p(y | f_{\boldsymbol{w}}(\boldsymbol{x}_i))$
  (e.g., $f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}, \quad y_i \sim \mathcal{N}(\boldsymbol{w}^\top \boldsymbol{x}_i, \sigma^2)$)

Setup: Observed data: $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$



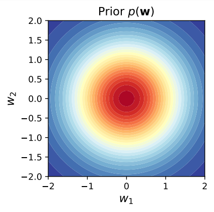


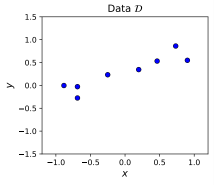
Model (Generative Assumption):

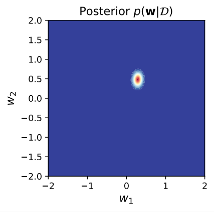
- Prior: $\boldsymbol{w} \sim p(\boldsymbol{w})$ (e.g., $\boldsymbol{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$)
- Likelihood: $y_i | \boldsymbol{x}_i, \boldsymbol{w} \sim p(y | f_{\boldsymbol{w}}(\boldsymbol{x}_i))$
  (e.g., $f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}, \quad y_i \sim \mathcal{N}(\boldsymbol{w}^\top \boldsymbol{x}_i, \sigma^2)$)

Setup: Observed data: $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$

Posterior: $p(\boldsymbol{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w})}{p(\mathcal{D})}$
(e.g., $\boldsymbol{w}|\mathcal{D} \sim \mathcal{N}(\boldsymbol{\mu}_N, \boldsymbol{\Sigma}_N)$, closed-form)







The Bayesian Scheme

Model (Generative Assumption):

- Prior: $\boldsymbol{w} \sim p(\boldsymbol{w})$ (e.g., $\boldsymbol{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$)
- Likelihood: $y_i | \boldsymbol{x}_i, \boldsymbol{w} \sim p(y | f_{\boldsymbol{w}}(\boldsymbol{x}_i))$
  (e.g., $f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}, \quad y_i \sim \mathcal{N}(\boldsymbol{w}^\top \boldsymbol{x}_i, \sigma^2)$)

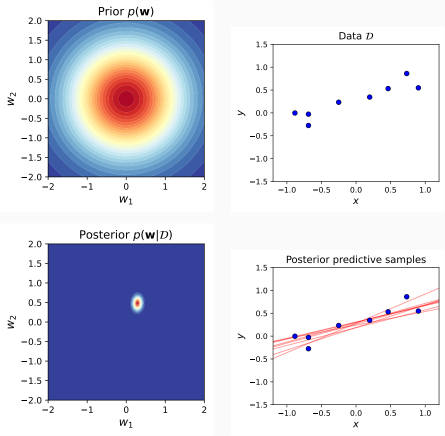
Setup: Observed data: $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$

Posterior: $p(\boldsymbol{w}|\mathcal{D}) = \dfrac{p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w})}{p(\mathcal{D})}$
(e.g., $\boldsymbol{w}|\mathcal{D} \sim \mathcal{N}(\boldsymbol{\mu}_N, \boldsymbol{\Sigma}_N)$, closed-form)

Predictive:

$$p(y^*|\boldsymbol{x}^*, \mathcal{D}) = \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w}) \, p(\boldsymbol{w}|\mathcal{D}) \, d\boldsymbol{w}$$

(e.g., $y^*|\boldsymbol{x}^*, \mathcal{D} \sim \mathcal{N}(\boldsymbol{\mu}_N^\top \boldsymbol{x}^*, \sigma^2 + \boldsymbol{x}^{*\top} \boldsymbol{\Sigma}_N \boldsymbol{x}^*)$)



Bayesian Neural Networks

- Prior: typically $p(\boldsymbol{w}) = \mathcal{N}(0, \sigma_0^2 I)$
- Likelihood: e.g., $\mathcal{N}(y; f_{\boldsymbol{w}}(\boldsymbol{x}), \sigma^2)$ (regression)
  where $f_{\boldsymbol{w}}(\cdot) : \mathbb{R}^d \to \mathbb{R}$ is a neural network: **nonlinear** w.r.t both parameters $\boldsymbol{w}$ and input $\boldsymbol{x}$

Bayesian Neural Networks

- Prior: typically $p(\boldsymbol{w}) = \mathcal{N}(0, \sigma_0^2 I)$
- Likelihood: e.g., $\mathcal{N}(y; f_{\boldsymbol{w}}(\boldsymbol{x}), \sigma^2)$ (regression)
  where $f_{\boldsymbol{w}}(\cdot) : \mathbb{R}^d \to \mathbb{R}$ is a neural network: **nonlinear** w.r.t both parameters $\boldsymbol{w}$ and input $\boldsymbol{x}$

The Problem:

- Posterior $p(\boldsymbol{w}|\mathcal{D})$ has no closed form
- Predictive integral $\int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})p(\boldsymbol{w}|\mathcal{D})d\boldsymbol{w}$ is intractable

Bayesian Neural Networks

- Prior: typically $p(\boldsymbol{w}) = \mathcal{N}(0, \sigma_0^2 I)$
- Likelihood: e.g., $\mathcal{N}(y; f_{\boldsymbol{w}}(\boldsymbol{x}), \sigma^2)$ (regression)
  where $f_{\boldsymbol{w}}(\cdot) : \mathbb{R}^d \to \mathbb{R}$ is a neural network: **nonlinear** w.r.t both parameters $\boldsymbol{w}$ and input $\boldsymbol{x}$

The Problem:

- Posterior $p(\boldsymbol{w}|\mathcal{D})$ has no closed form
- Predictive integral $\int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})p(\boldsymbol{w}|\mathcal{D})d\boldsymbol{w}$ is intractable

Typically: $\boldsymbol{w}$ is high-dimensional (millions of parameters)

Bayesian Neural Networks

- Prior: typically $p(\boldsymbol{w}) = \mathcal{N}(0, \sigma_0^2 I)$
- Likelihood: e.g., $\mathcal{N}(y; f_{\boldsymbol{w}}(\boldsymbol{x}), \sigma^2)$ (regression)
  where $f_{\boldsymbol{w}}(\cdot) : \mathbb{R}^d \to \mathbb{R}$ is a neural network: **nonlinear** w.r.t both parameters $\boldsymbol{w}$ and input $\boldsymbol{x}$

The Problem:

- Posterior $p(\boldsymbol{w}|\mathcal{D})$ has no closed form
- Predictive integral $\int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})p(\boldsymbol{w}|\mathcal{D})d\boldsymbol{w}$ is intractable

Typically: $\boldsymbol{w}$ is high-dimensional (millions of parameters)

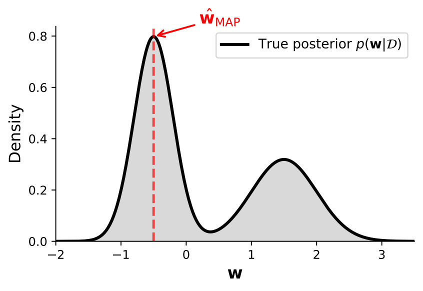
$\Rightarrow$ Need *approximate inference*!
(approximate $p(\boldsymbol{w}|\mathcal{D})$ by sampling / simple distributions $q(\boldsymbol{w})$)

Laplace Approximation (MacKay, 1992)

Idea: Approximate the posterior with a Gaussian at the Maximum A Posteriori (MAP).

Step 1: Find the MAP (= training with weight decay):

$$\hat{\boldsymbol{w}} = \arg\max_{\boldsymbol{w}} \left[\log p(\mathcal{D}|\boldsymbol{w}) + \log p(\boldsymbol{w})\right]$$



Find $\hat{\boldsymbol{w}}_{\text{MAP}}$

Laplace Approximation (MacKay, 1992)

Idea: Approximate the posterior with a Gaussian at the Maximum A Posteriori (MAP).

Step 1: Find the MAP (= training with weight decay):

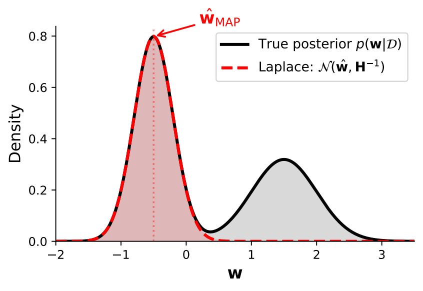
$$\hat{\boldsymbol{w}} = \arg\max_{\boldsymbol{w}} \left[\log p(\mathcal{D}|\boldsymbol{w}) + \log p(\boldsymbol{w})\right]$$

Step 2: Taylor expand log-posterior around $\hat{\boldsymbol{w}}$:

$$\log p(\boldsymbol{w}|\mathcal{D}) \approx \text{const} - \frac{1}{2}(\boldsymbol{w} - \hat{\boldsymbol{w}})^\top \boldsymbol{H}(\boldsymbol{w} - \hat{\boldsymbol{w}})$$

where $\boldsymbol{H} = -\nabla^2_{\boldsymbol{w}} \log p(\boldsymbol{w}|\mathcal{D})\big|_{\hat{\boldsymbol{w}}}$.

Result: $p(\boldsymbol{w}|\mathcal{D}) \approx \mathcal{N}(\hat{\boldsymbol{w}}, \boldsymbol{H}^{-1}) := q(\boldsymbol{w})$

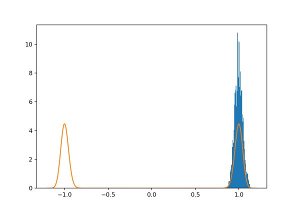


Laplace approximation of a multi-modal posterior.

Other Approximate Inference Methods

Markov Chain Monte Carlo (MCMC):

Sample $p(\boldsymbol{w}|\mathcal{D})$ asymptotically

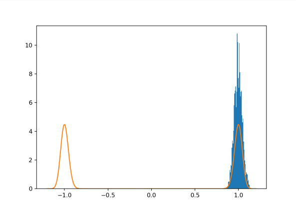


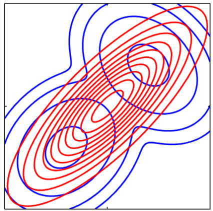
Other Approximate Inference Methods

Markov Chain Monte Carlo (MCMC):
Sample $p(\boldsymbol{w}|\mathcal{D})$ asymptotically

Variational Inference (VI):

- Obtain a lower bound of log marginal likelihood: $\log p(\mathcal{D}) \geq \underbrace{\mathbb{E}_{q_\phi}[\log p(\mathcal{D}|\boldsymbol{w})] - \mathrm{KL}(q_\phi(\boldsymbol{w}) \| p(\boldsymbol{w}))}_{\text{ELBO } \mathcal{L}(\phi)}$
- $q^* = \arg\max_{q_\phi} \mathcal{L}(q_\phi)$; typically $q_\phi(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$





Other Approaches to Workaround the Inference Difficulty

- **MC Dropout** (Gal & Ghahramani, 2016): Use dropout at test time as approximate VI.

$$p(y^*|\boldsymbol{x}^*, \mathcal{D}) \approx \frac{1}{T}\sum_{t=1}^{T} p(y^*|\boldsymbol{x}^*, \hat{\boldsymbol{w}} \odot \boldsymbol{z}_t), \quad z_{t,j} \sim \text{Bernoulli}(p)$$

Other Approaches to Workaround the Inference Difficulty

- **MC Dropout** (Gal & Ghahramani, 2016): Use dropout at test time as approximate VI.

$$p(y^*|\boldsymbol{x}^*, \mathcal{D}) \approx \frac{1}{T}\sum_{t=1}^{T} p(y^*|\boldsymbol{x}^*, \hat{\boldsymbol{w}} \odot \boldsymbol{z}_t), \quad z_{t,j} \sim \text{Bernoulli}(p)$$

- **Deep Ensembles** (Lakshminarayanan et al., 2017): Train M networks independently; $p(y^*|\boldsymbol{x}^*, \mathcal{D}) \approx \frac{1}{M}\sum_{m=1}^{M} p(y^*|\boldsymbol{x}^*, \hat{\boldsymbol{w}}_m)$

Other Approaches to Workaround the Inference Difficulty

- **MC Dropout** (Gal & Ghahramani, 2016): Use dropout at test time as approximate VI.

$$p(y^*|\boldsymbol{x}^*, \mathcal{D}) \approx \frac{1}{T}\sum_{t=1}^{T} p(y^*|\boldsymbol{x}^*, \hat{\boldsymbol{w}} \odot \boldsymbol{z}_t), \quad z_{t,j} \sim \text{Bernoulli}(p)$$

- **Deep Ensembles** (Lakshminarayanan et al., 2017): Train M networks independently; $p(y^*|\boldsymbol{x}^*, \mathcal{D}) \approx \frac{1}{M}\sum_{m=1}^{M} p(y^*|\boldsymbol{x}^*, \hat{\boldsymbol{w}}_m)$
- **SWAG** (Maddox et al., 2019): Continue SGD past convergence, fit a Gaussian to the trajectory.

$$q(\boldsymbol{w}) = \mathcal{N}\left(\bar{\boldsymbol{w}},\ \boldsymbol{\Sigma}_{\text{low-rank+diag}}\right), \quad \bar{\boldsymbol{w}} = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{w}_t$$

Other Approaches to Workaround the Inference Difficulty

- **MC Dropout** (Gal & Ghahramani, 2016): Use dropout at test time as approximate VI.

$$p(y^*|\boldsymbol{x}^*, \mathcal{D}) \approx \frac{1}{T}\sum_{t=1}^{T} p(y^*|\boldsymbol{x}^*, \hat{\boldsymbol{w}} \odot \boldsymbol{z}_t), \quad z_{t,j} \sim \text{Bernoulli}(p)$$

- **Deep Ensembles** (Lakshminarayanan et al., 2017): Train M networks independently; $p(y^*|\boldsymbol{x}^*, \mathcal{D}) \approx \frac{1}{M}\sum_{m=1}^{M} p(y^*|\boldsymbol{x}^*, \hat{\boldsymbol{w}}_m)$
- **SWAG** (Maddox et al., 2019): Continue SGD past convergence, fit a Gaussian to the trajectory.

$$q(\boldsymbol{w}) = \mathcal{N}(\bar{\boldsymbol{w}},\ \boldsymbol{\Sigma}_{\text{low-rank+diag}}), \quad \bar{\boldsymbol{w}} = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{w}_t$$

- **Bayesian Last Layer** (Snoek et al., 2015): Stochasticity only on last layer

Other Approaches to Workaround the Inference Difficulty

- **MC Dropout** (Gal & Ghahramani, 2016): Use dropout at test time as approximate VI.

$$p(y^*|\boldsymbol{x}^*, \mathcal{D}) \approx \frac{1}{T}\sum_{t=1}^{T} p(y^*|\boldsymbol{x}^*, \hat{\boldsymbol{w}} \odot \boldsymbol{z}_t), \quad z_{t,j} \sim \text{Bernoulli}(p)$$

- **Deep Ensembles** (Lakshminarayanan et al., 2017): Train M networks independently; $p(y^*|\boldsymbol{x}^*, \mathcal{D}) \approx \frac{1}{M}\sum_{m=1}^{M} p(y^*|\boldsymbol{x}^*, \hat{\boldsymbol{w}}_m)$
- **SWAG** (Maddox et al., 2019): Continue SGD past convergence, fit a Gaussian to the trajectory.

$$q(\boldsymbol{w}) = \mathcal{N}(\bar{\boldsymbol{w}},\ \boldsymbol{\Sigma}_{\text{low-rank+diag}}), \quad \bar{\boldsymbol{w}} = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{w}_t$$

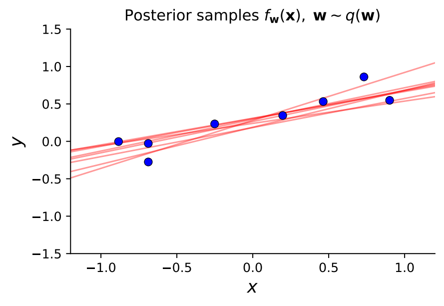
- **Bayesian Last Layer** (Snoek et al., 2015): Stochasticity only on last layer
- **Linearized Laplace** (Khan et al., 2019; Immer et al., 2021): Linearize $f_{\boldsymbol{w}}$ around $\hat{\boldsymbol{w}}$ and perform exact (Gaussian likelihood) or approximate (non-Gaussian likelihood) inference on the resulting linear model.

Making the Predictive Integral Tractable by Linearizing over Weights

Given $q(\boldsymbol{w}) \approx p(\boldsymbol{w}|\mathcal{D})$ (e.g., Laplace, SWAG, VI), the predictive integral remains intractable:

$$p(y^*|\boldsymbol{x}^*, \mathcal{D}) = \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})\, q(\boldsymbol{w})\, d\boldsymbol{w}$$

because $f_{\boldsymbol{w}}$ is **nonlinear** in $\boldsymbol{w}$.



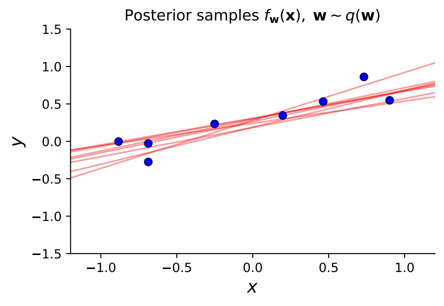
Samples from $q(\boldsymbol{w})$: no marginalisation.

Making the Predictive Integral Tractable by Linearizing over Weights

Given $q(\boldsymbol{w}) \approx p(\boldsymbol{w}|\mathcal{D})$ (e.g., Laplace, SWAG, VI), the predictive integral remains intractable:

$$p(y^*|\boldsymbol{x}^*, \mathcal{D}) = \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})\, q(\boldsymbol{w})\, d\boldsymbol{w}$$

because $f_{\boldsymbol{w}}$ is **nonlinear** in $\boldsymbol{w}$.



Samples from $q(\boldsymbol{w})$: no marginalisation.

Making the Predictive Integral Tractable by Linearizing over Weights

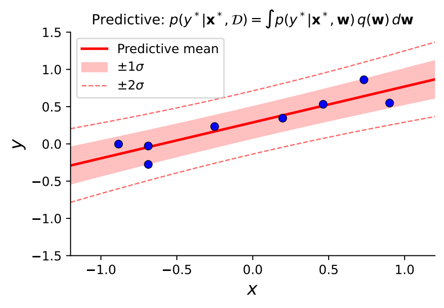
Given $q(\boldsymbol{w}) \approx p(\boldsymbol{w}|\mathcal{D})$ (e.g., Laplace, SWAG, VI), the predictive integral remains intractable:

$$p(y^*|\boldsymbol{x}^*, \mathcal{D}) = \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})\, q(\boldsymbol{w})\, d\boldsymbol{w}$$

because $f_{\boldsymbol{w}}$ is **nonlinear** in $\boldsymbol{w}$.

Practical Approach: Linearise $f_{\boldsymbol{w}}$ around $\hat{\boldsymbol{w}}$ Khan et al. (2019); Immer et al. (2021):

$$f_{\boldsymbol{w}}(\boldsymbol{x}) \approx f_{\hat{\boldsymbol{w}}}(\boldsymbol{x}) + \boldsymbol{J}_{\hat{\boldsymbol{w}}}(\boldsymbol{x})(\boldsymbol{w} - \hat{\boldsymbol{w}})$$



After marginalisation: mean $\pm$ uncertainty.

Issues in Weight-Space Inference

Another hidden problem from f_w's parameterization

Consider a single hidden layer network with H hidden units:

$$f_{\boldsymbol{w}}(\boldsymbol{x}) = \sum_{j=1}^{H} v_j \, \sigma\Big(\boldsymbol{w}_j^\top \boldsymbol{x} + b_j\Big)$$

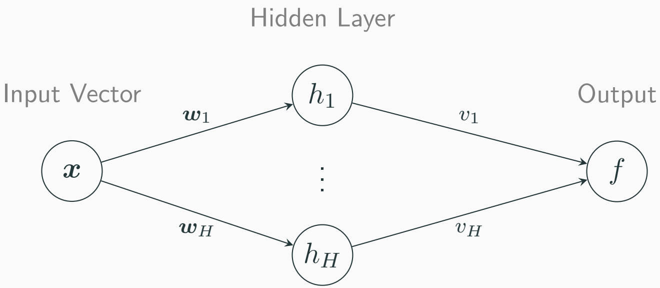
where $\boldsymbol{w} = (\boldsymbol{w}_1, b_1, v_1, \ldots, \boldsymbol{w}_H, b_H, v_H)$ collects all parameters.

Another hidden problem from f_w's parameterization

Consider a single hidden layer network with H hidden units:

$$f_{\boldsymbol{w}}(\boldsymbol{x}) = \sum_{j=1}^{H} v_j \, \sigma\left(\boldsymbol{w}_j^\top \boldsymbol{x} + b_j\right)$$

where $\boldsymbol{w} = (\boldsymbol{w}_1, b_1, v_1, \ldots, \boldsymbol{w}_H, b_H, v_H)$ collects all parameters.

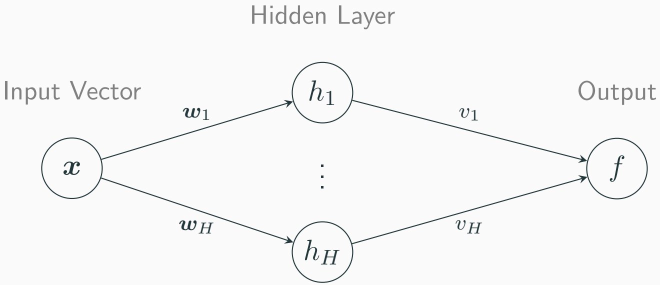


Another hidden problem from f_w's parameterization

Consider a single hidden layer network with H hidden units:

$$f_{\boldsymbol{w}}(\boldsymbol{x}) = \sum_{j=1}^{H} v_j \, \sigma\Big(\boldsymbol{w}_j^\top \boldsymbol{x} + b_j\Big)$$

where $\boldsymbol{w} = (\boldsymbol{w}_1, b_1, v_1, \ldots, \boldsymbol{w}_H, b_H, v_H)$ collects all parameters.



Key question: Can different weights produce the same function?

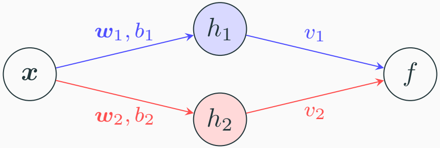
The Map $w \mapsto f_w$ Is Not Injective: Permutation Symmetry

Take $H = 2$. The two configurations below define **the same function**:

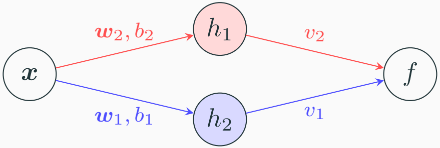
The Map $w \mapsto f_w$ Is Not Injective: Permutation Symmetry

Take $H = 2$. The two configurations below define **the same function**:

Configuration A



Configuration B



The Map $w \mapsto f_w$ Is Not Injective: Permutation Symmetry

Take $H = 2$. The two configurations below define **the same function**:

Swapping $(\boldsymbol{w}_1, b_1, v_1) \leftrightarrow (\boldsymbol{w}_2, b_2, v_2)$:

$$f_{\boldsymbol{w}}(\boldsymbol{x}) = v_1\sigma(\boldsymbol{w}_1^\top \boldsymbol{x} + b_1) + v_2\sigma(\boldsymbol{w}_2^\top \boldsymbol{x} + b_2) = f_{\boldsymbol{w}'}(\boldsymbol{x})$$

Different weights $\boldsymbol{w} \neq \boldsymbol{w}'$, but $f_{\boldsymbol{w}} = f_{\boldsymbol{w}'}$.

The Map $w \mapsto f_w$ Is Not Injective: Permutation Symmetry

Take $H = 2$. The two configurations below define **the same function**:

Swapping $(\boldsymbol{w}_1, b_1, v_1) \leftrightarrow (\boldsymbol{w}_2, b_2, v_2)$:

$$f_{\boldsymbol{w}}(\boldsymbol{x}) = v_1\sigma(\boldsymbol{w}_1^\top \boldsymbol{x} + b_1) + v_2\sigma(\boldsymbol{w}_2^\top \boldsymbol{x} + b_2) = f_{\boldsymbol{w}'}(\boldsymbol{x})$$

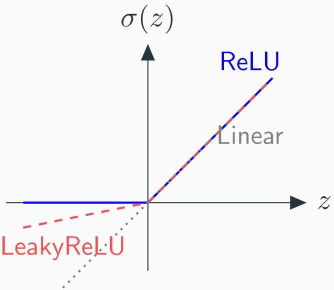
Different weights $\boldsymbol{w} \neq \boldsymbol{w}'$, but $f_{\boldsymbol{w}} = f_{\boldsymbol{w}'}$.

For **only one** hidden layer with H units, at least $H \times (H-1) \times \cdots \times 1 = H!$ equivalent weight configurations.

Scaling Property of Certain Activations $\sigma(\cdot)$

Some activations **scale linearly**:

$$\sigma(\alpha z) = \alpha\,\sigma(z) \quad \forall\,\alpha > 0$$



All piecewise-linear through the origin.

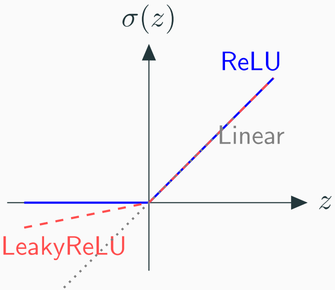
Scaling Property of Certain Activations $\sigma(\cdot)$

Some activations **scale linearly**:

$$\sigma(\alpha z) = \alpha\, \sigma(z) \quad \forall\, \alpha > 0$$

This family includes:

- ReLU: $\max(0, z)$
- LeakyReLU
- PReLU
- Linear



All piecewise-linear through the origin.

Scaling Symmetry

For any positively homogeneous σ, any $\alpha > 0$, and hidden unit j:

$$v_j \, \sigma(\boldsymbol{w}_j^\top \boldsymbol{x} + b_j) = \underbrace{(v_j/\alpha)}_{\text{new } v_j} \, \sigma\Big(\underbrace{(\alpha \boldsymbol{w}_j)^\top}_{\text{new } \boldsymbol{w}_j} \boldsymbol{x} + \underbrace{\alpha b_j}_{\text{new } b_j} \Big)$$

A **continuous family** of equivalent weight configurations, parameterized by α.

Scaling Symmetry

For any positively homogeneous σ, any $\alpha > 0$, and hidden unit j:

$$v_j\, \sigma(\boldsymbol{w}_j^\top \boldsymbol{x} + b_j) = \underbrace{(v_j/\alpha)}_{\text{new } v_j}\, \sigma\Big(\underbrace{(\alpha \boldsymbol{w}_j)}_{\text{new } \boldsymbol{w}_j}{}^\top \boldsymbol{x} + \underbrace{\alpha b_j}_{\text{new } b_j} \Big)$$

A **continuous family** of equivalent weight configurations, parameterized by α.

Combined consequences for the posterior:

- **Permutation** (any activation): $H!$ discrete copies of every mode per layer
- **Scaling** (homogeneous activations): continuous manifolds of equivalent weights

Scaling Symmetry

For any positively homogeneous σ, any $\alpha > 0$, and hidden unit j:

$$v_j \, \sigma(\boldsymbol{w}_j^\top \boldsymbol{x} + b_j) = \underbrace{(v_j/\alpha)}_{\text{new } v_j} \, \sigma\Big(\underbrace{(\alpha \boldsymbol{w}_j)^\top}_{\text{new } \boldsymbol{w}_j} \boldsymbol{x} + \underbrace{\alpha b_j}_{\text{new } b_j} \Big)$$

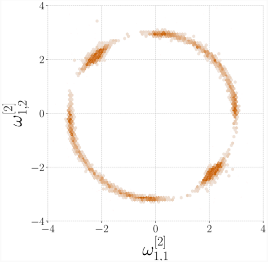
A **continuous family** of equivalent weight configurations, parameterized by α.

Combined consequences for the posterior:

- **Permutation** (any activation): $H!$ discrete copies of every mode per layer
- **Scaling** (homogeneous activations): continuous manifolds of equivalent weights

This is a property of the **parameterization**, not of the model representation capability.

What does this mean for Approximation Inference of $p(w|\mathcal{D})$

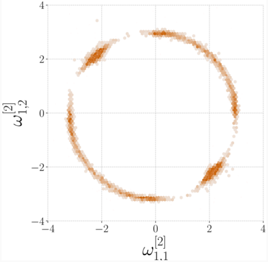


Original Posterior

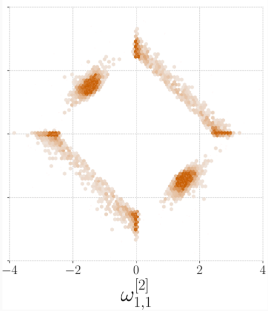
Estimated posterior over output weights v_1 vs v_2 of a 2-hidden-neuron network (Laurent et al., 2023).

1) The scaling orbit $\{(\alpha \boldsymbol{w}_j,\, \alpha b_j,\, v_j/\alpha) : \alpha > 0\}$ connects equivalent weights into the ring structure; the mirror symmetry across $v_1 = v_2$ reflects the 2! permutation copies.

What does this mean for Approximation Inference of $p(w|\mathcal{D})$



Original Posterior

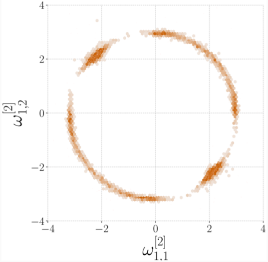


After Scaling Removal

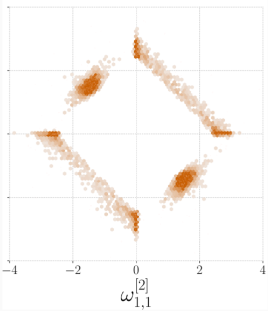
Estimated posterior over output weights v_1 vs v_2 of a 2-hidden-neuron network (Laurent et al., 2023).

1) The scaling orbit $\{(\alpha \boldsymbol{w}_j,\, \alpha b_j,\, v_j/\alpha) : \alpha > 0\}$ connects equivalent weights into the ring structure; the mirror symmetry across $v_1 = v_2$ reflects the $2!$ permutation copies. 2) Fixing $\|[\boldsymbol{w}_j;\, b_j]\| = 1$ determines a unique v_j,but the $2!$ copies remain.

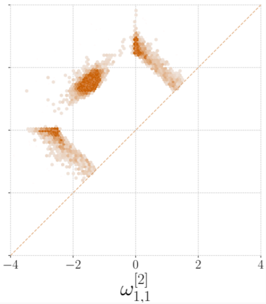
What does this mean for Approximation Inference of $p(w|\mathcal{D})$



Original Posterior



After Scaling Removal



After Scaling &

Permutation Removal

Estimated posterior over output weights v_1 vs v_2 of a 2-hidden-neuron network (Laurent et al., 2023).

1) The scaling orbit $\{(\alpha \boldsymbol{w}_j,\, \alpha b_j,\, v_j/\alpha) : \alpha > 0\}$ connects equivalent weights into the ring structure; the mirror symmetry across $v_1 = v_2$ reflects the $2!$ permutation copies. 2) Fixing $\|[\boldsymbol{w}_j;\, b_j]\| = 1$ determines a unique v_j,but the $2!$ copies remain. 3) Imposing $v_1 \geq v_2$ breaks the permutation symmetry.

Part2: A Geometric View & Approach Enhance BNN's Performance in Weight Space

A practical question

- People tried to use Laplace approximation to approximate $p(\boldsymbol{w}|\mathcal{D})$, but under the sampling scheme, the predictive uncertainty has **severe under-fit**.

A practical question

- People tried to use Laplace approximation to approximate $p(\boldsymbol{w}|\mathcal{D})$, but under the sampling scheme, the predictive uncertainty has **severe under-fit**.
- Surprisingly, when having an additional approximation over linearization of weights, under-fit has been suppressed.

A practical question

- People tried to use Laplace approximation to approximate $p(\boldsymbol{w}|\mathcal{D})$, but under the sampling scheme, the predictive uncertainty has **severe under-fit**.
- Surprisingly, when having an additional approximation over linearization of weights, under-fit has been suppressed.
- *Open Question* (Papamarkou et al., 2024): why the even crude linearization is beneficial?

Revisiting the Linearized Model (Khan et al., 2019; Immer et al., 2021)

Recall from Part 1: given a trained network with MAP estimate $\hat{\boldsymbol{w}}$, we **linearize** $f_{\boldsymbol{w}}$ w.r.t. $\boldsymbol{w}$ around $\hat{\boldsymbol{w}}$:

$$f^{\mathsf{lin}}_{\hat{\boldsymbol{w}}}(\boldsymbol{w}, \boldsymbol{x}) = f_{\hat{\boldsymbol{w}}}(\boldsymbol{x}) + \boldsymbol{J}_{\hat{\boldsymbol{w}}}(\boldsymbol{x})\,(\boldsymbol{w} - \hat{\boldsymbol{w}})$$

where $\boldsymbol{J}_{\hat{\boldsymbol{w}}}(\boldsymbol{x}) \in \mathbb{R}^{O \times D}$ is the Jacobian (O: output dim, D: number of parameters).

Revisiting the Linearized Model (Khan et al., 2019; Immer et al., 2021)

Recall from Part 1: given a trained network with MAP estimate $\hat{\boldsymbol{w}}$, we **linearize** $f_{\boldsymbol{w}}$ w.r.t. $\boldsymbol{w}$ around $\hat{\boldsymbol{w}}$:

$$f^{\text{lin}}_{\hat{\boldsymbol{w}}}(\boldsymbol{w}, \boldsymbol{x}) = f_{\hat{\boldsymbol{w}}}(\boldsymbol{x}) + \boldsymbol{J}_{\hat{\boldsymbol{w}}}(\boldsymbol{x})\,(\boldsymbol{w} - \hat{\boldsymbol{w}})$$

where $\boldsymbol{J}_{\hat{\boldsymbol{w}}}(\boldsymbol{x}) \in \mathbb{R}^{O \times D}$ is the Jacobian (O: output dim, D: number of parameters).

In Part 1 we linearized $f_{\boldsymbol{w}}$ only in the predictive integral. Here we treat $f^{\text{lin}}_{\hat{\boldsymbol{w}}}$ as the model itself and study its structure.

Revisiting the Linearized Model (Khan et al., 2019; Immer et al., 2021)

Recall from Part 1: given a trained network with MAP estimate $\hat{\boldsymbol{w}}$, we **linearize** $f_{\boldsymbol{w}}$ w.r.t. $\boldsymbol{w}$ around $\hat{\boldsymbol{w}}$:

$$f_{\hat{\boldsymbol{w}}}^{\text{lin}}(\boldsymbol{w}, \boldsymbol{x}) = f_{\hat{\boldsymbol{w}}}(\boldsymbol{x}) + \boldsymbol{J}_{\hat{\boldsymbol{w}}}(\boldsymbol{x})\,(\boldsymbol{w} - \hat{\boldsymbol{w}})$$

where $\boldsymbol{J}_{\hat{\boldsymbol{w}}}(\boldsymbol{x}) \in \mathbb{R}^{O \times D}$ is the Jacobian (O: output dim, D: number of parameters).

In Part 1 we linearized $f_{\boldsymbol{w}}$ only in the predictive integral. Here we treat $f_{\hat{\boldsymbol{w}}}^{\text{lin}}$ as the model itself and study its structure.

Key observation: For fixed $\boldsymbol{x}$, $f_{\hat{\boldsymbol{w}}}^{\text{lin}}$ is *affine in* $\boldsymbol{w}$. A weight perturbation $\boldsymbol{\delta} = \boldsymbol{w} - \hat{\boldsymbol{w}}$ produces a function change of $\boldsymbol{J}_{\hat{\boldsymbol{w}}}\boldsymbol{\delta}$.

The Jacobian $\boldsymbol{J}_{\hat{\boldsymbol{w}}}$ is the linear map from weight perturbations to function changes.

$\Rightarrow$ *What does this tell us about parameterization redundancy?*

Reparameterizations of Linear Functions

Consider a linear model $f(\boldsymbol{w}) = \boldsymbol{A}\boldsymbol{w} + \boldsymbol{b}$. Can two different weights produce the **same** function?

That is, does there exist $\boldsymbol{w}' \neq \boldsymbol{w}$ such that $f(\boldsymbol{w}') = f(\boldsymbol{w})$?

Reparameterizations of Linear Functions

Consider a linear model $f(\boldsymbol{w}) = \boldsymbol{A}\boldsymbol{w} + \boldsymbol{b}$. Can two different weights produce the **same** function?

That is, does there exist $\boldsymbol{w}' \neq \boldsymbol{w}$ such that $f(\boldsymbol{w}') = f(\boldsymbol{w})$? Expanding both sides:

$$\boldsymbol{A}\,\boldsymbol{w}' + \boldsymbol{b} = \boldsymbol{A}\,\boldsymbol{w} + \boldsymbol{b} \quad \Longrightarrow \quad \boldsymbol{A}\,\underbrace{(\boldsymbol{w}' - \boldsymbol{w})}_{\boldsymbol{\delta}} = \boldsymbol{0}$$

Reparameterizations of Linear Functions

Consider a linear model $f(\boldsymbol{w}) = \boldsymbol{A}\boldsymbol{w} + \boldsymbol{b}$. Can two different weights produce the **same** function?

That is, does there exist $\boldsymbol{w}' \neq \boldsymbol{w}$ such that $f(\boldsymbol{w}') = f(\boldsymbol{w})$? Expanding both sides:

$$\boldsymbol{A}\,\boldsymbol{w}' + \boldsymbol{b} = \boldsymbol{A}\,\boldsymbol{w} + \boldsymbol{b} \quad \Longrightarrow \quad \boldsymbol{A}\,\underbrace{(\boldsymbol{w}' - \boldsymbol{w})}_{\boldsymbol{\delta}} = \boldsymbol{0}$$

The set of all such $\boldsymbol{\delta}$ is known as the *null space* (kernel) of $\boldsymbol{A}$:

$$\ker(\boldsymbol{A}) = \{\boldsymbol{\delta} \in \mathbb{R}^D : \boldsymbol{A}\boldsymbol{\delta} = \boldsymbol{0}\}$$

These are **all and only** the directions along which you can move weights without changing the function — i.e., **reparameterizations**.

Reparameterizations of Linear Functions

Consider a linear model $f(\boldsymbol{w}) = \boldsymbol{A}\boldsymbol{w} + \boldsymbol{b}$. Can two different weights produce the **same** function?

That is, does there exist $\boldsymbol{w}' \neq \boldsymbol{w}$ such that $f(\boldsymbol{w}') = f(\boldsymbol{w})$? Expanding both sides:

$$\boldsymbol{A}\,\boldsymbol{w}' + \boldsymbol{b} = \boldsymbol{A}\,\boldsymbol{w} + \boldsymbol{b} \quad \Longrightarrow \quad \boldsymbol{A}\,\underbrace{(\boldsymbol{w}' - \boldsymbol{w})}_{\boldsymbol{\delta}} = \boldsymbol{0}$$

The set of all such $\boldsymbol{\delta}$ is known as the *null space* (kernel) of $\boldsymbol{A}$:

$$\ker(\boldsymbol{A}) = \{\boldsymbol{\delta} \in \mathbb{R}^D : \boldsymbol{A}\boldsymbol{\delta} = \boldsymbol{0}\}$$

These are **all and only** the directions along which you can move weights without changing the function — i.e., **reparameterizations**.

For the linearized neural network, $\boldsymbol{A} = \boldsymbol{J}_{\hat{\boldsymbol{w}}}$, so:

$$\ker(\boldsymbol{J}_{\hat{\boldsymbol{w}}}) = \{\text{reparameterization directions of } f^{\text{lin}}_{\hat{\boldsymbol{w}}}\}$$

Two Kinds of Directions in Weight Space

We just saw: directions in $\ker(\boldsymbol{J}_{\hat{\boldsymbol{w}}})$ are reparameterizations — they don't change the function.

Applying Laplace to the linearized model, the posterior is:

$$q(\boldsymbol{w}|\mathcal{D}) = \mathcal{N}\left(\boldsymbol{w} \mid \hat{\boldsymbol{w}},\ \left(\boldsymbol{J}_{\hat{\boldsymbol{w}}}^{\top}\boldsymbol{H}\,\boldsymbol{J}_{\hat{\boldsymbol{w}}} + \alpha\boldsymbol{I}\right)^{-1}\right),$$

where $\alpha = \sigma_0^{-2}$ is the prior precision.

The precision matrix $\boldsymbol{J}_{\hat{\boldsymbol{w}}}^{\top}\boldsymbol{H}\,\boldsymbol{J}_{\hat{\boldsymbol{w}}}$ is known as the Generalized Gauss-Newton matrix (*GGN*)

Two Kinds of Directions in Weight Space

We just saw: directions in $\ker(\boldsymbol{J}_{\hat{\boldsymbol{w}}})$ are reparameterizations — they don't change the function.

Applying Laplace to the linearized model, the posterior is:

$$q(\boldsymbol{w}|\mathcal{D}) = \mathcal{N}\left(\boldsymbol{w} \mid \hat{\boldsymbol{w}},\ \left(\boldsymbol{J}_{\hat{\boldsymbol{w}}}^\top \boldsymbol{H}\, \boldsymbol{J}_{\hat{\boldsymbol{w}}} + \alpha \boldsymbol{I}\right)^{-1}\right),$$

where $\alpha = \sigma_0^{-2}$ is the prior precision.

The precision matrix $\boldsymbol{J}_{\hat{\boldsymbol{w}}}^\top \boldsymbol{H}\, \boldsymbol{J}_{\hat{\boldsymbol{w}}}$ is known as the Generalized Gauss-Newton matrix (*GGN*)

(Since symmetric), this gives a clean split of the entire weight space:

$$\mathbb{R}^D = \operatorname{im}(\mathsf{GGN}_{\hat{\boldsymbol{w}}}) \oplus \ker(\mathsf{GGN}_{\hat{\boldsymbol{w}}})$$

Every perturbation $\boldsymbol{\delta}$ decomposes uniquely into these two orthogonal components.

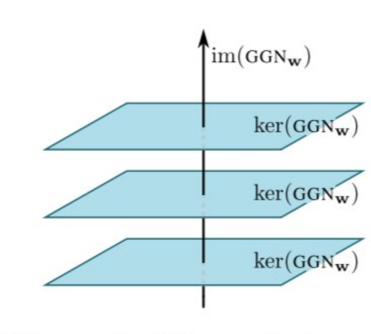
The Overparameterization Problem

Each per-input Jacobian $\boldsymbol{J}_{\hat{\boldsymbol{w}}}(\boldsymbol{x}_i) \in \mathbb{R}^{O \times D}$. Stacking all N data points gives an $NO \times D$ matrix, so:

$$\mathrm{rank}(\mathsf{GGN}_{\hat{\boldsymbol{w}}}) \leq NO$$

By rank-nullity, $\dim\big(\ker(\mathsf{GGN}_{\hat{\boldsymbol{w}}})\big) \geq D - NO$.

Meaning: In overparameterized networks ($D \gg NO$), the **vast majority** of weight space is reparameterizations!



$\ker(\mathsf{GGN})$: horizontal planes (vast).

$\mathrm{im}(\mathsf{GGN})$: vertical axis (thin).

What Does This Mean for the Posterior?

The $\mathsf{GGN}_{\hat{\boldsymbol{w}}}$ is symmetric, so by spectral decomposition:

$$\mathsf{GGN}_{\hat{\boldsymbol{w}}} = \boldsymbol{U}_1^\top \tilde{\boldsymbol{\Lambda}} \boldsymbol{U}_1 + \boldsymbol{U}_2^\top \cdot \boldsymbol{0} \cdot \boldsymbol{U}_2$$

where $\boldsymbol{U}_1$: eigenvectors with $\tilde{\lambda}_i > 0$ (span im), $\boldsymbol{U}_2$: eigenvectors with $\tilde{\lambda}_i = 0$ (span ker).

What Does This Mean for the Posterior?

The $\mathsf{GGN}_{\hat{\boldsymbol{w}}}$ is symmetric, so by spectral decomposition:

$$\mathsf{GGN}_{\hat{\boldsymbol{w}}} = \boldsymbol{U}_1^\top \tilde{\boldsymbol{\Lambda}} \boldsymbol{U}_1 + \boldsymbol{U}_2^\top \cdot \boldsymbol{0} \cdot \boldsymbol{U}_2$$

where $\boldsymbol{U}_1$: eigenvectors with $\tilde{\lambda}_i > 0$ (span im), $\boldsymbol{U}_2$: eigenvectors with $\tilde{\lambda}_i = 0$ (span ker).

Adding $\alpha \boldsymbol{I}$ and inverting:

$$\boldsymbol{\Sigma} = (\mathsf{GGN}_{\hat{\boldsymbol{w}}} + \alpha \boldsymbol{I})^{-1} = \underbrace{\boldsymbol{U}_1^\top (\tilde{\boldsymbol{\Lambda}} + \alpha \boldsymbol{I})^{-1} \boldsymbol{U}_1}_{\text{image: constrained by data}} + \underbrace{\alpha^{-1} \boldsymbol{U}_2^\top \boldsymbol{U}_2}_{\text{kernel: prior only}}$$

What Does This Mean for the Posterior?

The $\mathsf{GGN}_{\hat{\boldsymbol{w}}}$ is symmetric, so by spectral decomposition:

$$\mathsf{GGN}_{\hat{\boldsymbol{w}}} = \boldsymbol{U}_1^\top \tilde{\boldsymbol{\Lambda}} \boldsymbol{U}_1 + \boldsymbol{U}_2^\top \cdot \boldsymbol{0} \cdot \boldsymbol{U}_2$$

where $\boldsymbol{U}_1$: eigenvectors with $\tilde{\lambda}_i > 0$ (span im), $\boldsymbol{U}_2$: eigenvectors with $\tilde{\lambda}_i = 0$ (span $\ker$).

Adding $\alpha \boldsymbol{I}$ and inverting:

$$\boldsymbol{\Sigma} = (\mathsf{GGN}_{\hat{\boldsymbol{w}}} + \alpha \boldsymbol{I})^{-1} = \underbrace{\boldsymbol{U}_1^\top (\tilde{\boldsymbol{\Lambda}} + \alpha \boldsymbol{I})^{-1} \boldsymbol{U}_1}_{\text{image: constrained by data}} + \underbrace{\alpha^{-1} \boldsymbol{U}_2^\top \boldsymbol{U}_2}_{\text{kernel: prior only}}$$

With a weak prior (large $\sigma_0 \to$ small α), the kernel variance α^{-1} dominates. The posterior assigns large variance to directions that **do not change the function** (i.e., $\ker(\mathsf{GGN}_{\hat{\boldsymbol{w}}})$).

Two Questions

The posterior assigns large variance along reparameterization directions that locally do not change the function.

Q1: Why does the linearized Laplace approximation not suffer from this?

Q2: Why does this cause underfit when sampling from the nonlinear network?

Why Linearized Laplace Does Not Underfit

Key fact: $\ker(\mathsf{GGN}_{\hat{\boldsymbol{w}}}) = \ker(\boldsymbol{J}_{\hat{\boldsymbol{w}}})$: the directions with prior-only variance are exactly the directions where $\boldsymbol{J}_{\hat{\boldsymbol{w}}}\boldsymbol{w}_{\mathsf{ker}} = \mathbf{0}$.

Why Linearized Laplace Does Not Underfit

Key fact: $\ker(\mathsf{GGN}_{\hat{\boldsymbol{w}}}) = \ker(\boldsymbol{J}_{\hat{\boldsymbol{w}}})$: the directions with prior-only variance are exactly the directions where $\boldsymbol{J}_{\hat{\boldsymbol{w}}} \boldsymbol{w}_{\mathsf{ker}} = \mathbf{0}$.

Any sample decomposes as $\boldsymbol{w} = \hat{\boldsymbol{w}} + \boldsymbol{w}_{\mathsf{ker}} + \boldsymbol{w}_{\mathsf{im}}$.

Linearized Laplace (LLA):

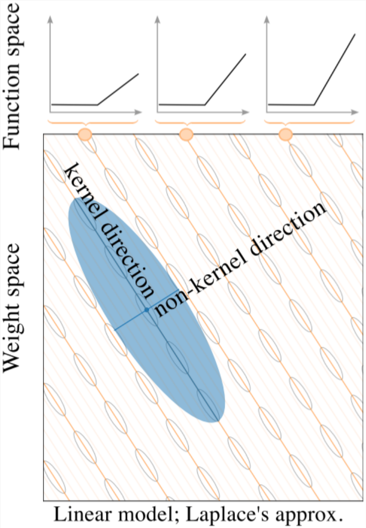
$$f^{\mathsf{lin}}(\boldsymbol{w}, \boldsymbol{x}) = f_{\hat{\boldsymbol{w}}}(\boldsymbol{x}) + \boldsymbol{J}_{\hat{\boldsymbol{w}}} \underbrace{\boldsymbol{w}_{\mathsf{ker}}}_{=\mathbf{0}} + \boldsymbol{J}_{\hat{\boldsymbol{w}}} \boldsymbol{w}_{\mathsf{im}}$$

Kernel projected out by $\boldsymbol{J}_{\hat{\boldsymbol{w}}}$. The predictive only reflects variance in $\mathrm{im}(\mathsf{GGN})$.

Why Sampled Laplace Underfits

For the nonlinear network, $\boldsymbol{w}_{\text{ker}}$ is *not* filtered out:

$$f(\hat{\boldsymbol{w}} + \boldsymbol{w}_{\text{ker}} + \boldsymbol{w}_{\text{im}}, \boldsymbol{x}) \neq f(\hat{\boldsymbol{w}} + \boldsymbol{w}_{\text{im}}, \boldsymbol{x})$$

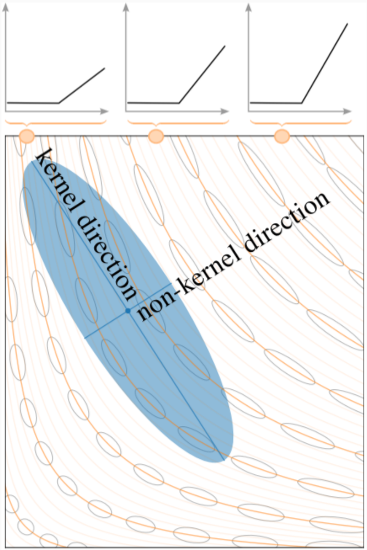


Why Sampled Laplace Underfits

For the nonlinear network, $\boldsymbol{w}_{\text{ker}}$ is *not* filtered out:

$$f(\hat{\boldsymbol{w}} + \boldsymbol{w}_{\text{ker}} + \boldsymbol{w}_{\text{im}}, \boldsymbol{x}) \neq f(\hat{\boldsymbol{w}} + \boldsymbol{w}_{\text{im}}, \boldsymbol{x})$$

The $\text{GGN}_{\hat{\boldsymbol{w}}}$ only captures the tangent space of the reparameterization manifold locally at $\hat{\boldsymbol{w}}$.



Nonlinear model; Laplace's approx.

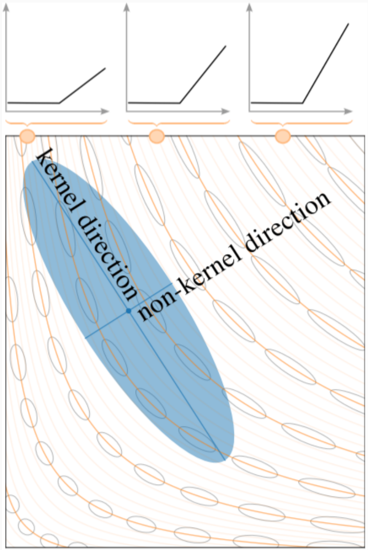
Why Sampled Laplace Underfits

For the nonlinear network, $\boldsymbol{w}_{\text{ker}}$ is *not* filtered out:

$$f(\hat{\boldsymbol{w}} + \boldsymbol{w}_{\text{ker}} + \boldsymbol{w}_{\text{im}}, \boldsymbol{x}) \neq f(\hat{\boldsymbol{w}} + \boldsymbol{w}_{\text{im}}, \boldsymbol{x})$$

The $\text{GGN}_{\hat{\boldsymbol{w}}}$ only captures the tangent space of the reparameterization manifold locally at $\hat{\boldsymbol{w}}$.

Large posterior variance along kernel directions pushes samples beyond the local tangent approximation and off the true reparameterization manifold.



Nonlinear model; Laplace's approx.

Why Sampled Laplace Underfits

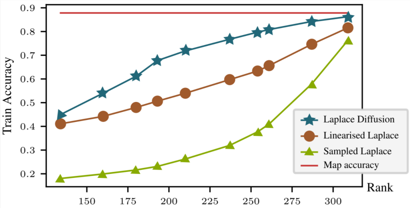
For the nonlinear network, $\boldsymbol{w}_{\mathsf{ker}}$ is *not* filtered out:

$$f(\hat{\boldsymbol{w}} + \boldsymbol{w}_{\mathsf{ker}} + \boldsymbol{w}_{\mathsf{im}}, \boldsymbol{x}) \neq f(\hat{\boldsymbol{w}} + \boldsymbol{w}_{\mathsf{im}}, \boldsymbol{x})$$

The $\mathsf{GGN}_{\hat{\boldsymbol{w}}}$ only captures the tangent space of the reparameterization manifold locally at $\hat{\boldsymbol{w}}$.

Large posterior variance along kernel directions pushes samples beyond the local tangent approximation and off the true reparameterization manifold.

Sampled functions no longer reflect the training data hence underfit!



Higher GGN rank $\rightarrow$ smaller kernel $\rightarrow$ less underfitting. Roy et al. (2024)

Beyond the Linear Case (Roy et al., 2024)

Problem: For nonlinear networks, reparameterization sets are **curved manifolds**, not flat subspaces. A fixed Gaussian posterior cannot adapt to this curvature.

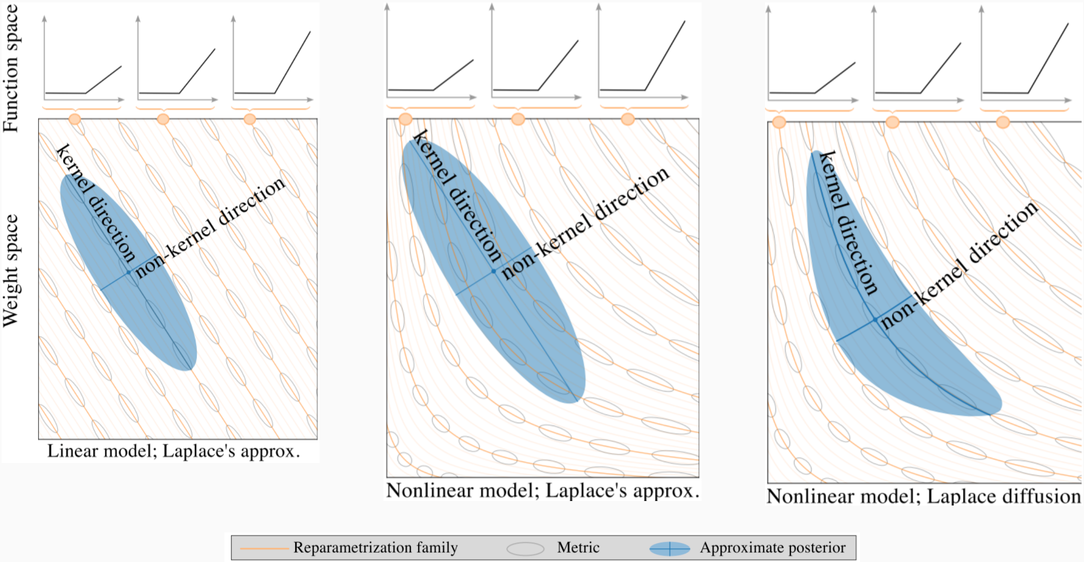
Beyond the Linear Case (Roy et al., 2024)

Problem: For nonlinear networks, reparameterization sets are **curved manifolds**, not flat subspaces. A fixed Gaussian posterior cannot adapt to this curvature.

Solution: Use the GGN as a *pseudo-Riemannian metric* it assigns zero distance to reparameterizations and adapts locally. A diffusion process under this metric gives a **reparameterization invariant posterior**.

The linearized Laplace approximation is a single-step special case of this diffusion.

Reparameterization Non-Invariance (Roy et al., 2024)



Takeaways

BNN: inference is similar to standard Bayesian statistics, however, due to its specific parameterization redundancy, this caused reparameterization challenges for the inference problem.

Takeaways

BNN: inference is similar to standard Bayesian statistics, however, due to its specific parameterization redundancy, this caused reparameterization challenges for the inference problem.

The core problem: In overparameterized networks, the vast majority of weight space consists of reparameterizations. Standard approximate posteriors waste density there.

Takeaways

BNN: inference is similar to standard Bayesian statistics, however, due to its specific parameterization redundancy, this caused reparameterization challenges for the inference problem.

The core problem: In overparameterized networks, the vast majority of weight space consists of reparameterizations. Standard approximate posteriors waste density there.

Why LLA helps: Although the LLA posterior still assign density along reparameterization direction, the predictive variance project them out due to they lie in $\ker(\boldsymbol{J}_{\hat{\boldsymbol{w}}})$. Hence does not cause any underfit iseue.

Takeaways

BNN: inference is similar to standard Bayesian statistics, however, due to its specific parameterization redundancy, this caused reparameterization challenges for the inference problem.

The core problem: In overparameterized networks, the vast majority of weight space consists of reparameterizations. Standard approximate posteriors waste density there.

Why LLA helps: Although the LLA posterior still assign density along reparameterization direction, the predictive variance project them out due to they lie in $\ker(\boldsymbol{J}_{\hat{\boldsymbol{w}}})$. Hence does not cause any underfit iseue.

A solution exists: Using the GGN as a pseudo-Riemannian metric yields a posterior that is reparameterization invariant by construction Roy et al. (2024).

References

A. Immer, M. Korzepa, and M. Bauer. Improving predictions of bayesian neural nets via local linearization. In *International conference on artificial intelligence and statistics*, pages 703–711. PMLR, 2021.

M. E. Khan, A. Immer, E. Abedi, and M. Korzepa. Approximate inference turns deep networks into gaussian processes. *Advances in neural information processing systems*, 32, 2019.

O. Laurent, E. Aldea, and G. Franchi. A symmetry-aware exploration of bayesian neural network posteriors. *arXiv preprint arXiv:2310.08287*, 2023.

D. J. MacKay. A practical bayesian framework for backpropagation networks. *Neural Computation*, 4 (3):448–472, 1992.

T. Papamarkou, M. Skoularidou, K. Palla, L. Aitchison, J. Arbel, D. Dunson, M. Filippone, V. Fortuin, P. Hennig, J. M. Hernández-Lobato, et al. Position: Bayesian deep learning is needed in the age of large-scale ai. In *International Conference on Machine Learning*, 2024.

H. Roy, M. Miani, C. H. Ek, P. Hennig, M. Pförtner, L. Tatzel, and S. Hauberg. Reparameterization invariance in approximate bayesian inference. *Advances in Neural Information Processing Systems*, 37:8132–8164, 2024.