SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Thesis title

Author

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Thesis title

Titel der Abschlussarbeit

Author:	Author
Examiner:	Supervisor
Supervisor:	Advisor
Submission Date:	Submission date

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, Submission date Author

Abstract

Contents

Notation	Explanation
θ	Parameters of a neural network
$(x, y) \in \mathcal{D}$	Dataset with inputs x and labels/targets y
v_t	Time-dependent vector field
J_v	Jacobian of the vector field v
ϕ_t	Flow map from time 0 to time t

Table 1: Summary of notation used throughout the thesis.

1 Introduction

1.1 Section

2 Bayesian Deep Learning

A primary use case for a generative model over neural network weights is in Bayesian deep learning, where it can allow efficient inference by transporting the prior distribution to the posterior. Thus, to motivate the rest of the discussion, we first give an overview of concepts from Bayesian deep learning. Section 2.1 is a general introduction. It is followed by a review of inference methods (Laplace approximations, variational inference, MCMC-based methods) typically used for Bayesian neural networks, and we conclude with a review of literature around Bayesian deep learning particularly relevant for our work in Section 2.4.

2.1 General Concepts

In typical Bayesian fashion, Bayesian deep learning (refer to (MacKay, 1992; Neal, 1996) for foundational work and (Goan and Fookes, 2020; Arbel et al., 2023) for more recent reviews) aims to quantify the uncertainty in neural networks through probability distributions over their parameters, rather than obtaining a single solution by an SGD-like optimization method. Then, given the *posterior* distribution $p(\theta|\mathcal{D})$ over weights θ conditioned on the dataset $\mathcal{D}$, predictions are obtained via *Bayesian model averaging*:

$$p(y|x,\mathcal{D}) = \mathbb{E}_{\theta\sim p(\theta|\mathcal{D})}\left[p(y|x,\theta)\right] = \int p(y|x,\theta)p(\theta|\mathcal{D})d\theta, \tag{2.1}$$

where the prediction is averaged over the posterior over the weights. Note that since the forward pass through the model $p(y|x,\theta)$, is deterministic, the uncertainty in predictions results solely from the uncertainty over parameters. To bring things together, Bayesian inference over neural network weights consists of three steps:

1. Specify prior $p(\theta)$.
2. Compute/sample posterior $p(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta)p(\theta)$.
3. Average predictions over the posterior.

The last step only requires forward passes through the model and thus is straightforward. The first two steps require deeper consideration.

2.2 Priors

Specifiying a prior mainly consists of two choices: specifying an architecture, and specifying a probability distribution over the weights. The distribution is typically taken to be an isotropic Gaussian, which is an uninformative prior but straightforward to work with.

Different architectural decisions also result in different functions, even if the flattened weight vectors are identical, meaning that the choice of an architecture further specifies a prior in function space. As a simple example, keeping the depth and width of a neural network constant, even just changing the activation function from a ReLU to a sigmoid results in a differnet distribution of functions. The functional distribution can also be specified in a more deliberate way; e.g. a translation-invariant convolutional neural network, or a group-equivariant network (Cohen and Welling, 2016) puts probability mass only on functions satisfying certain equivariance constraints depending on the task at hand.

We keep this discussion short since the choice of a prior has tangential impact in the rest of the presentation, and we refer to recent reviews such as (Fortuin, 2022) for a more detailed treatment of Bayesian neural network priors.

2.3 Inference

- Variational methods
- MCMC-based methods
- Transformation-based methods (normalizing flow)

Laplace
VI
DE
MCMC introduction
HMC
Highlight distinction between chain-based and transformation based sampling, using normalizing flows and connections to optimal transport etc
– Stochastic –
SG-MCMC
SG-HMC

2.4 Properties of Bayesian Neural Network Posteriors

How they are like Boltzmann distributions, not arbitrary.

3 Flow Models

3.1 Overview

We train our flow using *flow matching* (Lipman et al., 2023; Albergo et al., 2023; Liu et al., 2022), which generalizes diffusion models with a more flexible design space. In this section we first formulate the flow matching objective (Sec. 3.2), explain the design choices it enables (Sec. 3.2.3), and describe in more detail how samples (Sec. 3.3) and likelihoods (Sec. 3.4) can be obtained using a flow model.

3.2 Flow Matching

Flow matching, first proposed in (Lipman et al., 2023; Albergo et al., 2023; Liu et al., 2022) aims to solve the problem of *dynamic transport*, i.e. finding a time-dependent vector field to transport the source (prior) distribution p_0 to the target (data) distribution p_1. More formally, the vector field $u_t : [0,1] \times \mathbb{R}^d \to \mathbb{R}^d$ leads to the ordinary differential equation (ODE)

$$dx = u_t(x)dt \tag{3.1}$$

and induces a *flow* $\phi : [0,1] \times \mathbb{R}^d \to \mathbb{R}^d$ that gives the solution to the ODE at time t with starting point x_0, such that

$$\frac{d}{dt}\phi_t(x_0) = u_t(\phi_t(x)) \tag{3.2}$$

$$\phi_0(x_0) = x_0. \tag{3.3}$$

Starting with p_0, transformed distributions p_t can then be defined using this flow with the push-forward operation

$$p_t := [\phi_t]_\#(p_0) \tag{3.4}$$

and the instantaneous change in the density satisfies the *continuity equation*

$$\frac{\partial p}{\partial t} = -\nabla \cdot (p_t u_t) \tag{3.5}$$

which means that probability mass is conserved during the transformation. With these formulations, we say the vector field u_t *generates* the *probability path* (also called *interpolant*) p_t.

3.2.1 The Objective

The formulation above could also be applied to traditional continuous normalizing flows (CNFs) (Chen et al., 2018) as well, and flow matching is an instantiation of CNFs. However, continuous normalizing flows have in the past been trained using objectives which required solving and then backpropagating through the ODE, such as KL-divergence or other likelihood-based objectives, which made training costly. This problem was later addressed with diffusion models and their simpler regression objectives such as score matching and denoising (Sohl-Dickstein et al., 2015; Song et al., 2021; Ho et al., 2020) that proved to be very effective. As we will now demonstrate, the flow matching objective is also formulated as a simulation-free regression objective, and is more flexible than the diffusion objectives.

As explained in Section 3.2, the goal in flow matching is to learn a vector field $v_\theta : [0,1] \times \mathbb{R}^d \to \mathbb{R}^d$ parametrized by a neural network.[1] If we know the ground truth vector field u and can sample from the intermediate p_t's, we can directly optimize the flow matching objective

$$\mathcal{L}_{\text{FM}}(\theta) := \mathbb{E}_{t\sim\mathcal{U}(0,1),x_t\sim p_t(x)}\|v_\theta(t,x_t) - u_t(x_t)\|^2 \tag{3.6}$$

by first sampling a time point t and then $x_t \sim p_t$. However in practice, we neither have a closed form expression for u nor can sample from an arbitrary p_t without integrating the flow.

The *conditional flow matching* (CFM) framework first introduced in (Lipman et al., 2023) and then extended in (Tong et al., 2023) solves this problem by formulating the intermediate probability paths as mixtures of simpler paths,

$$p_t(x) = \int p_t(x \mid z)q(z)dz \tag{3.7}$$

where z is the conditioning variable and $q(z)$ a distribution over z (e.g. with $z := (x_0, x_1), q(z) = p_0(x_0)p_1(x_1), u_t(x \mid z) = x_1 - x_0$, and $p_t(x \mid z) = \mathcal{N}(x \mid (1-t)x_0 + tx_1, \sigma^2)$). Then similar to how p_t's were generated by the vector field u_t, the conditional probability paths $p_t(x \mid z)$ are generated by conditional vector fields $u_t(x \mid z)$, and as shown in (Tong et al., 2023) u_t can be decomposed in terms of these conditional vector fields as

$$u_t(x) = \mathbb{E}_{z\sim q(z)}\frac{u_t(x \mid z)p_t(x \mid z)}{p_t(x)}. \tag{3.8}$$

Then similar to Equation 3.6, we have the conditional flow matching objective

$$\mathcal{L}_{\text{CFM}}(\theta) := \mathbb{E}_{t\sim\mathcal{U}(0,1),z\sim q(z),x_t\sim p_t(x|z)}\|v_\theta(t,x_t) - u_t(x_t \mid z)\|^2. \tag{3.9}$$

[1] For conciseness, we interchangeably use the subscripts for vector fields to denote time ($u_t(x)$) and parameters ($v_\theta(t,x)$).

That is, we first sample a conditioning variable z, and then regress to the *conditional* vector field $u_t(x \mid z)$. Thus we obtain a tractable objective by defining sample-able conditional probability paths and a tractable conditional vector field. Moreover, as shown in (Tong et al., 2023), the FM and CFM objectives equivalent up to a constant and thus

$$\nabla_\theta \mathcal{L}_{\text{FM}}(\theta) = \nabla_\theta \mathcal{L}_{\text{CFM}}(\theta), \tag{3.10}$$

meaning we do not lose the expressive power of the FM objective by regressing only to the conditional vector fields. As we will show in Section 3.2.3, the choice of these conditional probability paths, vector fields, along with the conditioning variable itself, makes the flow matching approach particularly flexible.

3.2.2 Training

To sum up the discussion in the previous section, a step of training a flow model using CFM objective (Equation 3.9) proceeds as follows:

1. Sample $t \sim \mathcal{U}(0,1), z \sim q(z)$, and $x_t \sim p_t(x \mid z)$.
2. Compute $\mathcal{L}_{\text{CFM}}(\theta) = \|v_\theta(t, x_t) - u_t(x_t \mid z)\|^2$.
3. Update θ with $\nabla_\theta \mathcal{L}_{\text{CFM}}(\theta)$.

3.2.3 Couplings and Conditional Paths

With this framework established, the three main design choices for building a flow matching model are choosing the coupling $q(z)$, the conditional "ground truth" vector field $u_t(x \mid z)$, and the conditional probability paths $p_t(x \mid z)$. Starting with an arbitrary source distribution p_0 and target distribution p_1, Tong et al. (2023) propose three different ways of constructing conditional paths from couplings between p_0 and p_1, of which we focus on two (independent and optimal transport couplings). In all setups, the condition variable z corresponds to a pair (x_0, x_1) of source and target points.

Independent Coupling. The simplest way of obtaining is to sample independently from p_0 and p_1; i.e. $q(z) = p_0(x_0)p_1(x_1)$, with the conditional paths and the vector field defined as

$$p_t(x \mid z) = \mathcal{N}(x \mid (1-t)x_0 + tx_1, \sigma^2) \tag{3.11}$$
$$u_t(x \mid z) = x_1 - x_0. \tag{3.12}$$

The conditional paths and the coupling defined this was are easily easy to sample from, but have undesirable properties such as crossing paths which which can lead to high

variance in the ground truth vector field for a specific point and time. Moreover in practice, independent couplings can lead to curved paths that incur higher integration errors, as there is no notion of straightness considered in this formulation.

Optimal Transport. To obtain straighter and shorter paths that are easier to integrate, Tong et al. (2023) propose to use the static 2-Wasserstein optimal transport map π as the coupling; i.e.

$$q(z) = \pi(x_0, x_1), \tag{3.13}$$

with the conditional paths and vector field defined as in Equations 3.11 and 3.12. The flow model thus obtained solves the dynamic optimal OT problem as $\sigma^2 \to 0$ (Proposition 3.4 in (Tong et al., 2023)). However, computing the exact OT map for the entire dataset is challenging, especially in high dimensions as in our problem. It can instead be approximated using mini-batches (Fatras et al., 2021). This means at the end the OT problem is solved only to an approximation, but nevertheless results in straighter paths that cross less often, since intuitively an $x_0 \sim p_0$ is more likely to be coupled with $x_1 \sim p_1$ closer to it rather than an x_1 chosen uniformly random.

3.2.4 Connection to Diffusion (maybe remove)

3.2.5 Extensions (mini lit review of example uses)

3.3 Sampling / Integration

3.4 Computing Likelihoods

In earlier normalizing flows that aim to learn a static mapping between the two distributions (Rezende and Mohamed, 2015), given source samples $x_0 \sim p_0$, likelihoods of the generated samples $z = f(x_0) \approx p_1$ can be computed exactly via the change of variables formula

$$\log p_1(z) = \log p_0(z) - \log \det \left| J_f(z) \right| \tag{3.14}$$

where J_f is the Jacobian of f. Thus, we can obtain exact likelihoods for the generated samples by taking the determinant of the Jacobian of the normalizing flow. Since Jacobian computations can be costly, this has motivated work on designing normalizing flows with easier to compute Jacobians, such as RealNVP (Dinh et al., 2017).

In a *continuous normalizing flow* on the other hand, the *instantaneous change of variables* formula (Chen et al., 2018) defines the change in probability mass through time. Given that the vector field v_t is continuous in t and uniformly Lipschitz continuous in $\mathbb{R}^d$, it

holds that

$$\frac{d \log p_t(\phi_t(x))}{dt} = -\operatorname{div}(v_t(\phi_t(x))) \tag{3.15}$$

$$= -\operatorname{Tr}\left(\frac{dv_t(\phi_t(x))}{dt}\right) \tag{3.16}$$

where $\frac{dv_t(\phi_t(x))}{dt} =: J_v(\phi_t(x))$ is the Jacobian of the vector field. Then we integrate over time to compute the full change in probability:

$$\log p_1(\phi_1(x)) = \log p_0(\phi_0(x)) - \int_0^1 \operatorname{Tr}(J_v(\phi_t(x)))dt. \tag{3.17}$$

Then we can integrate the Jacobian trace of the vector field through time (simultaneously with sampling) to obtain exact likelihoods for the generated samples.

3.4.1 Faster Likelihoods Through Trace Estimation

However, materializing the full Jacobian of the vector field can be prohibitively expensive, especially if the task is high dimensional (as in our case) since the log determinant computation has a time complexity of $O(d^3)$ (Grathwohl et al., 2018) without any restrictions on the structure of the Jacobian.

To alleviate this problem, (Grathwohl et al., 2018) propose to use the *Hutchinson trace estimator* (Hutchinson, 1990) for an unbiased estimate of the Jacobian trace of a square matrix:

$$\operatorname{Tr}(J_v) = \mathbb{E}_{p(\epsilon)}\left[\epsilon^T J_v \epsilon\right] \tag{3.18}$$

where $p(\epsilon)$ is chosen such that $\mathbb{E}[\epsilon] = 0$ and $\operatorname{Cov}(\epsilon) = I$, typically a Gaussian or a Rademacher distribution. Then, we can use this estimator in place of the explicit trace computation in Equation 3.17 and compute the likelihoods as

$$\log p_1(\phi_1(x)) = \log p_0(\phi_0(x)) - \int_0^1 \mathbb{E}_{p(\epsilon)}\left[\epsilon^T J_v(\phi_t(x))\epsilon\right] dt. \tag{3.19}$$

The performance benefit of using the Hutchinson trace estimator results from the fact that the Jacobian-vector product $J_v\epsilon$ can be computed very efficiently by automatic differentiation (Baydin et al., 2018), giving the whole approach a time complexity of $O(d)$ only. Due to this significant performance improvement and being an unbiased estimate, the Hutchinson trace estimator has been widely used in the diffusion/flow model literature (Lipman et al., 2023; Song et al., 2021).

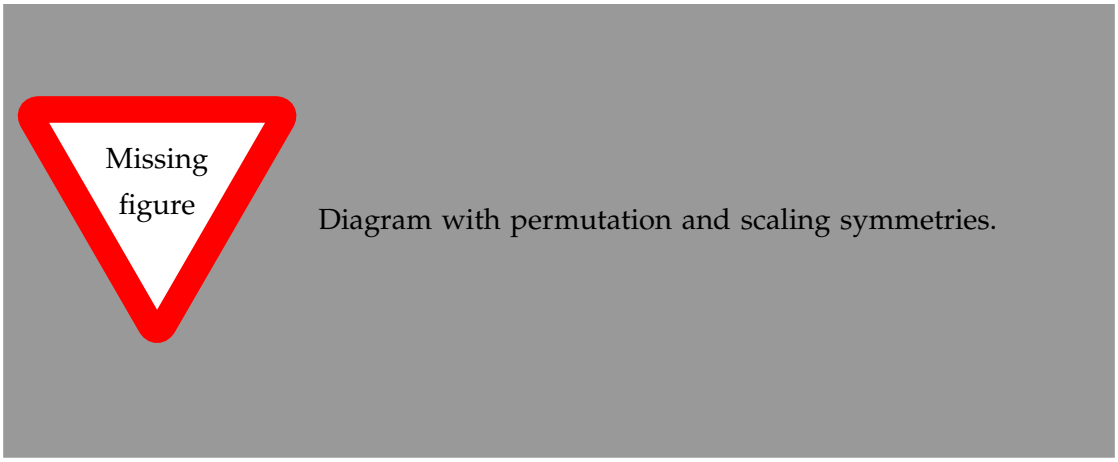

Figure 4.1: Symmetries of neural networks.

4 Symmetries of Neural Network Weights

Symmetries in data, such as rotation symmetries in molecules or translation symmetries in images, can be used to obtain useful inductive biases for ML models (Bronstein et al., 2021; Weiler, 2023). By restricting the search space to functions that respect those symmetries, such inductive biases make training more data-efficient (Brehmer et al., 2024). Since our data modality is neural network weights, taking the symmetries of neural networks into account can likewise make learning in weight space more effective.

Given a neural network f with parameters θ, we define a *symmetry* as an operation ϕ that leaves the function computed by the neural network unchanged; i.e. $f_{\phi(\theta)}(x) = f_\theta(x)$. We are further interested only in the static symmetries that hold for any θ and thus can more reliably be used as inductive biases, rather than dynamic symmetries specific to a certain value of θ. Such symmetries of neural network weights have been studied for a long time (Hecht-Nielsen, 1990), and are still key considerations for understanding the loss landscape and training dynamics of neural networks (Brea et al., 2019; Simsek et al., 2021; Lim, Putterman, et al., 2024; Zhao, Gower, et al., 2024).

4.1 Permutation and Scaling Symmetries of Neural Networks

A typical neural network with non-linear activations has two main kinds of symmetries: *permutation symmetries* and *scaling symmetries*. Permutation symmetries arise from the connectivity structure of the neural network, and scaling symmetries mainly arise from the particular non-linearities.

More formally, consider a two-layer MLP with weight matrices $\mathbf{W}_1, \mathbf{W}_2$ and element-wise activations σ; i.e. $f_\theta(x) = \mathbf{W}_2\sigma(\mathbf{W}_1 x)$, ignoring the biases for simplicity but the following discussion applies to the biases as well. Let $\mathbf{P}$ be an arbitrary permutation matrix. We can permute the hidden neurons, and apply the same permutation to their outgoing weights as well to keep the function unchanged:

$$f_\theta(x) = \mathbf{W}_2\sigma(\mathbf{W}_1 x) = \mathbf{W}_2\mathbf{P}^T\sigma(\mathbf{P}\mathbf{W}_1 x). \tag{4.1}$$

Since σ is applied element-wise, we have $\mathbf{W}_2\mathbf{P}^T\mathbf{P}\sigma(\mathbf{W}_1 x)$ which preserves the function as $\mathbf{P}^T\mathbf{P} = \mathbf{I}$. Similarly, convolutional neural networks' channels can be permuted while preserving the function. More generally, Lim, Maron, et al. (2023) show that the permutation symmetries of any neural network correspond to graph automorphisms of its neural DAG, constructed with each edge corresponding to a parameter (e.g. directly the computational graph for MLPs, or with each filter corresponding to a node for CNNs).

In addition to these permutation symmetries, element-wise activation functions such as ReLU introduce further scaling symmetries to neural networks. For example, for the ReLU activation it holds for any real number $\lambda > 0$ that $\text{ReLU}(\lambda x) = \lambda\text{ReLU}(x)$. This means the input weights to a layer with ReLU activations can be multiplied with a positive number, and the function will remain unchanged as long as the output weights are scaled down with the same factor. Although we will mostly consider ReLU networks in the following sections, it is worth noting that such scaling symmetries exist for activation functions besides ReLU as well. (Godfrey et al., 2022) have shown that symmetries resulting from activation functions can be associated with different *intertwiner groups*, and provide concrete examples of these groups for various activation functions.

4.2 Linear Mode Connectivity

Closely related to the literature on symmetries of neural network weights is the topic of *linear mode connectivity*, concerned with finding (linear) low-loss paths between SGD-optimized weights. Finding such paths is useful for many downstream applications, such as ensembling neural networks (Garipov et al., 2018) by finding accurate weights with different representations without training, or model merging (Stoica et al., 2024).

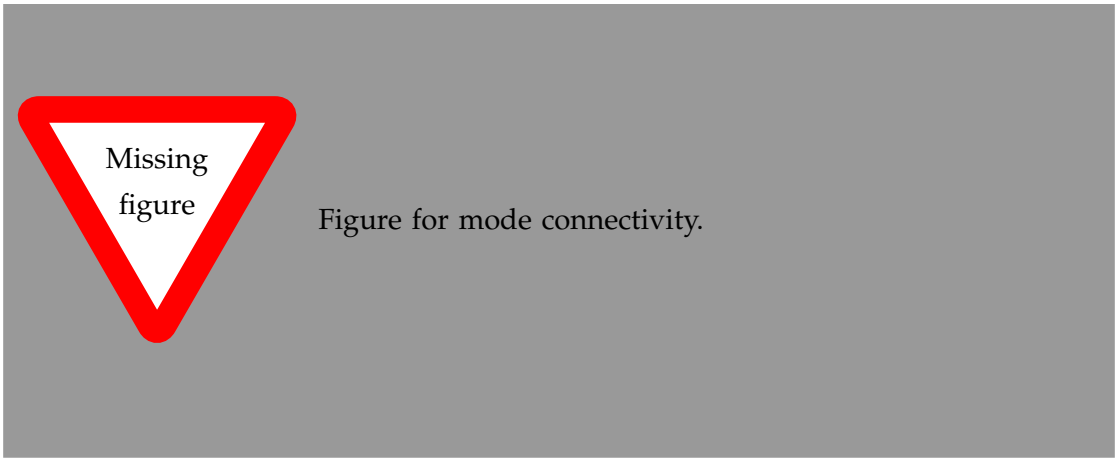

Figure 4.2: Mode connectivity.

Garipov et al. (2018) first formulated the problem as finding parametrized curves between NN weights, minimizing the loss across the curve. It was later conjectured by Entezari et al. (2022) that up to permutation symmetries, such low-loss paths are linear. The linear mode connectivity hypothesis has since then given rise to a fruitful research area (Ferbach et al., 2024; Rossi et al., 2023; Zhao, Dehmamy, et al., 2023).

For our purposes, modes being linearly connected would imply that finding the optimal permutations could effectively reduce the number of modes in the weight-space posterior, making it easier to approximate. Accounting for this multi-modality has been shown to improve the effectiveness of Bayesian neural networks (Sommer et al., 2024). With this motivation, we next focus on the literature around finding such permutations.

4.3 Aligning Neural Networks

The problem of finding a permutation of one neural network weights to obtain a linear low-loss path with another neural network, we call *aligning* neural networks, has given rise to a high number of methods over years, the entirety of which could be a thesis in itself. For instance, (Ainsworth et al., 2023) propose various approaches. The first is a data-based approach that matches the activations of models A and B of each layer,

$$\mathbf{P}^* = \arg\min_{\mathbf{P} \in S_d} \sum_{i=1}^{n} \|\mathbf{Z}_A - \mathbf{P}\mathbf{Z}_B\|^2 \tag{4.2}$$

with d dimensional activations $\mathbf{Z}$ for n data points. This is an instance of the *linear assignment problem* which can be solved in polynomial time (Crouse, 2016). An alternative

is to align the weights of the neural networks directly, which can again be reduced to a linear assignment problem and is more efficient since it requires no forward passes over the model, but sacrifices accuracy by ignoring the data.

Peña et al. (2023) propose a more flexible framework, making it possible optimize for any differentiable objective, and we also use their approach in the rest of our work. Peña et al. (2023) start by relaxing the constraint of binary permutation matrix $\mathbf{P} \in \Pi$ to obtain unconstrained $\mathbf{X} \in \mathbb{R}^{m \times n}$. Such a matrix can then be mapped to the space of binary permutation matrices via the *Sinkhorn operator*:

$$S_\tau(\mathbf{X}) = \arg\max_{\mathbf{P} \in \Pi} \langle \mathbf{P}, \mathbf{X} \rangle_F + \tau h(\mathbf{P}) \tag{4.3}$$

where F denotes the Frobenius norm and entropy $h(\mathbf{P}) = -\sum \mathbf{P} \log \mathbf{P}$. Then the optimization is performed over $\mathbb{R}^{m \times n}$, and a binary permutation matrix is obtained through the Sinkhorn operator.

The main advantage of the approach of Peña et al. (2023) is that it can be used with arbitrary differentiable objectives. To align the parameters θ_A and θ_B, with $\pi_\mathbf{P}$ denoting the permutation applied to the weights, Peña et al. (2023) propose three objectives. First is a straightforward weight-matching similar to (Ainsworth et al., 2023)

$$\mathcal{L}_{L2} := \|\theta_A - \pi_\mathbf{P}(\theta_B)\|^2, \tag{4.4}$$

followed by two data-based objectives. Minimizing the mid-point loss between the two weights as

$$\mathcal{L}_{\text{Mid}} := \mathcal{L}\left(\frac{\theta_A + \pi_\mathbf{P}(\theta_B)}{2}\right) \tag{4.5}$$

or the loss at a random intermediate point

$$\mathcal{L}_{\text{Rnd}} := \mathcal{L}\left((1-\lambda)\theta_A + \lambda \pi_\mathbf{P}(\theta_B)\right) \tag{4.6}$$

with $\lambda \sim \mathcal{U}(0,1)$. At the end, particularly the data-based losses result in more effective permutations than the method of (Ainsworth et al., 2023), and we choose to use the approach of Peña et al. (2023) in the rest of our work also considering its flexibility.

4.4 Canonical Representations of Neural Networks

This discussion around permutation and scaling symmetries of neural networks culminates with *canonical representations* of neural networks, i.e. unique representations for each set of permutation/scale-symmetric neural networks, limiting our discussion to ReLU networks for simplicity. Following the work of (Pittorino et al., 2022), this can achieved in two steps given a set of neural networks $\{\theta_i\}_i^N$:

1. Align all neural networks to a single *reference* neural network θ', using the approach of (Peña et al., 2023).

2. For each intermediate layer l and neuron k, scale down the incoming weights and biases by the norm of the incoming weight vector, $|w_k^l|^{-1}$, and the outgoing weights by $|w_k^l|$. Additionally for classification tasks, normalize the last layer's weights to $\sqrt{C}$, with C the number of classes. This does not change the predicted label due to the argmax operation at the last layer.

With these two operations, the permutation and scaling symmetries are "broken," as all the neural networks that compute the same function up to permutation and scaling are now mapped to the same point in weight space. Nevertheless, while the scaling symmetry is broken exactly, the permutation symmetry is broken up to an approximation since the alignment methods (Ainsworth et al., 2023; Peña et al., 2023) only output approximate solutions. For this reason, as we will describe in the next section, we use graph neural networks to fully account for the permutation symmetries. This canonicalization also gives a specific geometric structure to the set of neural network weights. Each neuron, characterized by its incoming weights, now lies on the unit hypersphere, and each layer in turn has a product geometry of hyperspheres. This enables the computation of geodesic paths and distances between neural networks.

5 Graph Neural Networks & MetaNets

6 Generative Models in Weight-Space

7 Method & Design Choices

8 Results

9 Discussion

10 Conclusion & Future Work

Abbreviations

List of Figures

List of Tables

Bibliography

Ainsworth, Samuel K. et al. (2023). *Git Re-Basin: Merging Models modulo Permutation Symmetries*. DOI: 10.48550/arXiv.2209.04836. arXiv: 2209.04836 [cs].

Albergo, Michael S. et al. (2023). *Stochastic Interpolants: A Unifying Framework for Flows and Diffusions*. DOI: 10.48550/arXiv.2303.08797. arXiv: 2303.08797 [cond-mat].

Arbel, Julyan et al. (2023). *A Primer on Bayesian Neural Networks: Review and Debates*. DOI: 10.48550/arXiv.2309.16314. arXiv: 2309.16314 [cs, math, stat].

Baydin, Atılım Gunes et al. (2018). "Automatic Differentiation in Machine Learning: A Survey." In: *Journal of Machine Learning Research*.

Brea, Johanni et al. (2019). *Weight-Space Symmetry in Deep Networks Gives Rise to Permutation Saddles, Connected by Equal-Loss Valleys across the Loss Landscape*. DOI: 10.48550/arXiv.1907.02911. arXiv: 1907.02911 [cs, stat].

Brehmer, Johann et al. (2024). *Does Equivariance Matter at Scale?* DOI: 10.48550/arXiv.2410.23179. arXiv: 2410.23179.

Bronstein, Michael M. et al. (2021). *Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges*. DOI: 10.48550/arXiv.2104.13478. arXiv: 2104.13478 [cs, stat].

Chen, Ricky T. Q. et al. (2018). "Neural Ordinary Differential Equations." In: *Advances in Neural Information Processing Systems*. Vol. 31. Curran Associates, Inc.

Cohen, Taco and Max Welling (2016). "Group Equivariant Convolutional Networks." In: *Proceedings of The 33rd International Conference on Machine Learning*. PMLR, pp. 2990–2999.

Crouse, David F. (2016). "On Implementing 2D Rectangular Assignment Algorithms." In: *IEEE Transactions on Aerospace and Electronic Systems* 52.4, pp. 1679–1696. ISSN: 1557-9603. DOI: 10.1109/TAES.2016.140952.

Dinh, Laurent et al. (2017). "Density Estimation Using Real NVP." In: *International Conference on Learning Representations*.

Entezari, Rahim et al. (2022). *The Role of Permutation Invariance in Linear Mode Connectivity of Neural Networks*. DOI: 10.48550/arXiv.2110.06296. arXiv: 2110.06296 [cs].

Fatras, Kilian et al. (2021). *Minibatch Optimal Transport Distances; Analysis and Applications*. arXiv: 2101.01792 [cs, stat].

Ferbach, Damien et al. (2024). "Proving Linear Mode Connectivity of Neural Networks via Optimal Transport." In: *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 3853–3861.

Fortuin, Vincent (2022). "Priors in Bayesian Deep Learning: A Review." In: *International Statistical Review* 90.3, pp. 563–591. ISSN: 0306-7734, 1751-5823. DOI: 10.1111/insr.12502.

Garipov, Timur et al. (2018). "Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs." In: *Advances in Neural Information Processing Systems*. Vol. 31. Curran Associates, Inc.

Goan, Ethan and Clinton Fookes (2020). "Bayesian Neural Networks: An Introduction and Survey." In: vol. 2259, pp. 45–87. DOI: 10.1007/978-3-030-42553-1_3. arXiv: 2006.12024 [cs, stat].

Godfrey, Charles et al. (2022). "On the Symmetries of Deep Learning Models and Their Internal Representations." In: *Advances in Neural Information Processing Systems* 35, pp. 11893–11905.

Grathwohl, Will et al. (2018). *FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models*. DOI: 10.48550/arXiv.1810.01367. arXiv: 1810.01367.

Hecht-Nielsen, Robert (1990). "ON THE ALGEBRAIC STRUCTURE OF FEEDFORWARD NETWORK WEIGHT SPACES." In: *Advanced Neural Computers*. Ed. by Rolf Eckmiller. Amsterdam: North-Holland, pp. 129–135. ISBN: 978-0-444-88400-8. DOI: 10.1016/B978-0-444-88400-8.50019-4.

Ho, Jonathan et al. (2020). "Denoising Diffusion Probabilistic Models." In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., pp. 6840–6851.

Hutchinson, M.F. (1990). "A Stochastic Estimator of the Trace of the Influence Matrix for Laplacian Smoothing Splines." In: *Communications in Statistics - Simulation and Computation* 19.2, pp. 433–450. ISSN: 0361-0918. DOI: 10.1080/03610919008812866.

Lim, Derek, Haggai Maron, et al. (2023). *Graph Metanetworks for Processing Diverse Neural Architectures*. DOI: 10.48550/arXiv.2312.04501. arXiv: 2312.04501 [cs, stat].

Lim, Derek, Moe Putterman, et al. (2024). *The Empirical Impact of Neural Parameter Symmetries, or Lack Thereof*. DOI: 10.48550/arXiv.2405.20231. arXiv: 2405.20231 [cs, stat].

Lipman, Yaron et al. (2023). *Flow Matching for Generative Modeling*. DOI: 10.48550/arXiv.2210.02747. arXiv: 2210.02747 [cs, stat].

Liu, Xingchao et al. (2022). *Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow*. DOI: 10.48550/arXiv.2209.03003. arXiv: 2209.03003.

MacKay, David J C (1992). "Bayesian Methods for Adaptive Models." In.

Neal, Radford M. (1996). *Bayesian Learning for Neural Networks*.

Peña, Fidel A. Guerrero et al. (2023). "Re-Basin via Implicit Sinkhorn Differentiation." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20237–20246.

Pittorino, Fabrizio et al. (2022). "Deep Networks on Toroids: Removing Symmetries Reveals the Structure of Flat Regions in the Landscape Geometry." In: *Proceedings of the 39th International Conference on Machine Learning*. PMLR, pp. 17759–17781.

Rezende, Danilo and Shakir Mohamed (2015). "Variational Inference with Normalizing Flows." In: *Proceedings of the 32nd International Conference on Machine Learning*. PMLR, pp. 1530–1538.

Rossi, Simone et al. (2023). *On Permutation Symmetries in Bayesian Neural Network Posteriors: A Variational Perspective*. DOI: 10.48550/arXiv.2310.10171. arXiv: 2310.10171 [cs, stat].

Simsek, Berfin et al. (2021). "Geometry of the Loss Landscape in Overparameterized Neural Networks: Symmetries and Invariances." In: *Proceedings of the 38th International Conference on Machine Learning*. PMLR, pp. 9722–9732.

Sohl-Dickstein, Jascha et al. (2015). "Deep Unsupervised Learning Using Nonequilibrium Thermodynamics." In.

Sommer, Emanuel et al. (2024). *Connecting the Dots: Is Mode-Connectedness the Key to Feasible Sample-Based Inference in Bayesian Neural Networks?* DOI: 10.48550/arXiv.2402.01484. arXiv: 2402.01484 [cs, stat].

Song, Yang et al. (2021). *Score-Based Generative Modeling through Stochastic Differential Equations*. arXiv: 2011.13456 [cs, stat].

Stoica, George et al. (2024). *ZipIt! Merging Models from Different Tasks without Training*. DOI: 10.48550/arXiv.2305.03053. arXiv: 2305.03053 [cs].

Tong, Alexander et al. (2023). *Improving and Generalizing Flow-Based Generative Models with Minibatch Optimal Transport*. DOI: 10.48550/arXiv.2302.00482. arXiv: 2302.00482 [cs].

Weiler, Maurice (2023). *Equivariant and Coordinate Independent Convolutional Networks*.

Zhao, Bo, Nima Dehmamy, et al. (2023). "Understanding Mode Connectivity via Parameter Space Symmetry." In: *UniReps: The First Workshop on Unifying Representations in Neural Models*.

Zhao, Bo, Robert M Gower, et al. (2024). "IMPROVING CONVERGENCE AND GENERALIZATION USING PARAMETER SYMMETRIES." In.