SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Geometric Flow Models over Neural Network Weights

Ege Erdogan

Supervisors
David Rügamer (LMU Munich)
Bastian Grossenbacher Rieck (University of Fribourg)

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

Munich, 08.01.2025 Ege Erdogan

Abstract

Deep generative models such as flow and diffusion models have proven to be effective in modeling high-dimensional and complex data types such as videos or proteins, and this has motivated their use in different data modalities, such as neural network weights. A generative model of neural network weights would be useful for a diverse set of applications, such as Bayesian deep learning, learned optimization, and transfer learning. However, the existing work on weight-space generative models often ignores the symmetries of neural network weights, or only takes into account a subset of them. Modeling those symmetries, such as permutation symmetries between subsequent layers in an MLP, the filters in a convolutional network, or scaling symmetries arising with the use of non-linear activations, holds the potential to make weight-space generative modeling more efficient by effectively reducing the dimensionality of the problem.

In this light, we aim to design generative models in weight-space that more comprehensively respect the symmetries of neural network weights. We build on recent work on generative modeling with flow matching, and weight-space graph neural networks to design three different weight-space flows. Each of our flows takes a different approach to modeling the geometry of neural network weights, and thus allows us to explore the design space of weight-space flows in a principled way. Our results confirm that modeling the geometry of neural networks more faithfully leads to more effective flow models that can generalize to different tasks and architectures, and we show that while our flows obtain competitive performance with orders of magnitude fewer parameters than previous work, they can be further improved by scaling them up. We conclude by listing potential directions for future work on weight-space generative models.

Our implementation is available at https://github.com/ege-erdogan/weightflow.

Contents

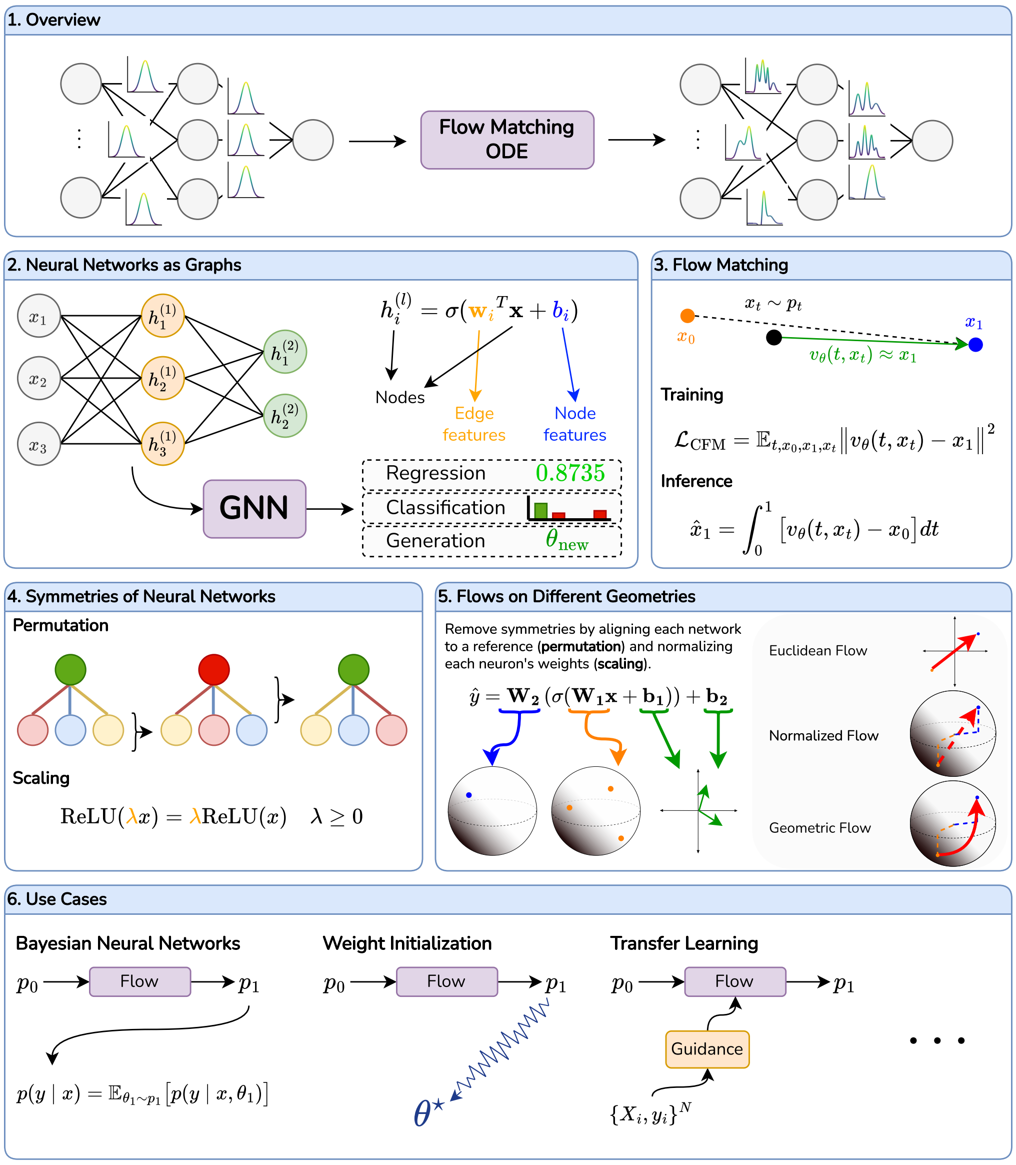

1. Overview
Flow Matching ODE
2. Neural Networks as Graphs
x_1
x_2
x_3
$h_1^{(1)}$
$h_2^{(1)}$
$h_3^{(1)}$
$h_1^{(2)}$
$h_2^{(2)}$
$h_i^{(l)} = \sigma(\mathbf{w}_i{}^T\mathbf{x} + b_i)$
Nodes
Edge features
Node features
GNN
Regression 0.8735
Classification
Generation θ_{new}
3. Flow Matching
$x_t \sim p_t$
x_0
x_1
$v_\theta(t, x_t) \approx x_1$
Training
$\mathcal{L}_{\text{CFM}} = \mathbb{E}_{t,x_0,x_1,x_t}\|v_\theta(t,x_t) - x_1\|^2$
Inference
$\hat{x}_1 = \int_0^1 [v_\theta(t,x_t) - x_0]\,dt$
4. Symmetries of Neural Networks
Permutation
Scaling
$\text{ReLU}(\lambda x) = \lambda\text{ReLU}(x) \quad \lambda \geq 0$
5. Flows on Different Geometries
Remove symmetries by aligning each network to a reference (permutation) and normalizing each neuron's weights (scaling).
$\hat{y} = \mathbf{W_2}\,(\sigma(\mathbf{W_1}\mathbf{x} + \mathbf{b_1})) + \mathbf{b_2}$
Euclidean Flow
Normalized Flow
Geometric Flow
6. Use Cases
Bayesian Neural Networks
p_0 Flow p_1
$p(y \mid x) = \mathbb{E}_{\theta_1 \sim p_1}[p(y \mid x, \theta_1)]$
Weight Initialization
p_0 Flow p_1
$\theta^\star$
Transfer Learning
p_0 Flow p_1
Guidance
$\{X_i, y_i\}^N$
. . .

Figure 1.: **Overview of our weight-space flow.** We aim to learn a flow in weight-space *(1)*, processing neural networks with GNNs *(2)*, using flow matching *(3)* and taking into account the symmetries of neural network weights *(4)*. We propose three different flows *(5)* and potential use cases include Bayesian neural networks, learned weight initialization, or transfer learning *(6)*.

1. Introduction

Deep generative models such as diffusion and flow models that have led to significant developments in image generation (Esser et al., 2024) and biological applications such as protein structure prediction (Abramson et al., 2024) have also been applied to neural network weights (Peebles et al., 2022; Schürholt, 2024). However, the existing works do not take into account the geometry of neural networks arising from permutation and scaling symmetries, or only consider the permutation symmetries. However, modeling the symmetries of data often leads to improved performance and more data-efficient training (Brehmer et al., 2024). We attempt to fill this gap by building flow models that learn a vector field to transport a prior over neural network to the posterior for a specific task through the flow matching framework (Lipman, Havasi, et al., 2024), processing neural network weights with permutation-invariant graph neural networks (Kofinas et al., 2024; Lim, Maron, et al., 2023) and propose three candidate designs differing on how they handle the underlying geometry. Figure 1 gives an overview of our approach, and we summarize the main points in this Introduction.

1.1. Motivation

Weight-Space Generative Models as Learned Optimizers

A generative model of neural network weights is in essence a learned optimizer, and learned optimizers constitute an active research area in their own right (Hospedales et al., 2022). However, unlike typical learned optimizers that are often trained by unrolling the gradient-based optimization of neural networks (Finn et al., 2017), generative models are trained directly with data (trained neural network weights), using the outputs of neural network training. This implies weight-space generative models can be improved by utilizing public datasets of neural network checkpoints such as (Peebles et al., 2022; Schürholt, Taskiran, et al., 2022).

Inference with weight-space generative models is also fundamentally different than gradient-based optimization. Rather than predict the task gradients or output optimization steps, a generative model can output trained weights in "one-shot." Furthermore, they open the way for applications of tools from the modern generative models literature to navigate the space of neural network weights.

Weight-Space Generative Models as Probabilistic Models

The fundamental units of modern generative models such as diffusion and flow models are probability distributions, that is they learn to transport one probability distribution such as a Normal distribution to another one such as the distribution of certain images. Weight-space generative models therefore also model a complex probability distribution over neural network weights. They can both efficiently sample from it, and evaluate the likelihood of arbitrary points under this distribution.

Being able to both sample from and evaluate the likelihood function of a complex distribution puts generative models in a unique place among similar methods. The likelihood function of variational approximations of neural network posteriors can also be evaluated, but they model considerably simpler distributions, while Markov chain Monte Carlo methods can sample from more complex multi-modal distributions but do not allow likelihood computations. Nevertheless, this power of generative models comes with increased computational demands during training and the need for careful architecture design.

Generative Models, Geometry, and Neural Network Weights

Geometric generative models that model the geometry of their domain by invariance/equivariance constraints (Klein et al., 2023), parametrizing their data points on manifolds (Chen and Lipman, 2023), or learning a manifold structure from data points alone (Kapusniak et al., 2024) have been applied to a wide range problems from climate modeling (Bodnar et al., 2024) to protein design (Bose et al., 2024). By baking in the correct inductive biases from the start rather than expecting a model to learn them from scratch or through data augmentation, such geometric models make training more data-efficient.

It is thus a logical first-step, especially for high-dimensional and multi-modal distributions such as neural network posteriors, to model the geometric structure of the data as accurately as possible. In particular for deep neural networks, their non-identifiability - there being multiple parametrizations of the same function, is a key reason behind this multi-modality, and it is hypothesized that the modes in a neural network posterior are actually linearly connected up to function-preserving permutations (Entezari et al., 2022). Together, these considerations motivate that weight-space learning tasks can be considerably simplified through appropriate geometric modeling choices.

1.2. Overview

Symmetries of Neural Networks (Chapter 4)

Neural network weights possess various symmetries, i.e. transformations of the network parameters that leave the function it is computing unchanged, and this topic has been an active research area since the early days of neural network research (Hecht-Nielsen, 1990). For example in an MLP, permuting the neurons in one layer together with the outgoing weights preserves the function, as does similarly permuting the channels in a convolutional neural network. Non-linear activation functions induce further scaling symmetries (Godfrey et al., 2022); e.g. for a constant $\lambda \geq 0$, $\mathrm{ReLU}(\lambda x) = \lambda \mathrm{ReLU}(x)$, which means scaling the input weights to a neuron and applying the inverse scaling to the outgoing weights again preserves neural network's function. In addition to these static symmetries, other forms of symmetries might arise from the structure of the data, or dynamically during periods of training (Zhao, Dehmamy, et al., 2024).

Accounting for these symmetries in a weight-space learning task by using architectures with the correct inductive biases, through data augmentation (Shamsian et al., 2024), or by removing the symmetries by mapping neural networks to canonical representations (Pittorino et al., 2022) as we will do can reduce the effective dimensionality of the problem and make weight-space learning more efficient.

Neural Networks as Graphs (Chapter 5)

A neural network can naturally be modeled as a graph, and this has fueled a recent line of work building graph neural networks (GNNs) that take as input other neural networks (Kofinas et al., 2024; Lim, Maron, et al., 2023; Kalogeropoulos et al., 2024). The nodes often correspond to the neurons in an MLP or channels in a convolutional network and the edges to the weights. Used with the appropriate positional encodings, this graph formalism provides an effective way of handling the permutation symmetries of various kinds of neural networks including transformers and neural networks with residual connections, and has also been extended to account for scaling symmetries (Kalogeropoulos et al., 2024). Furthermore, since a GNN is not restricted to graphs with a certain structure, weight-space GNNs have the additional benefit that the same GNN can be used to process different neural networks, even those with different architectures altogether.

As part of our flows, we process neural networks using GNNs, more specifically the Relational Transformer architecture (Diao and Loynd, 2023; Kofinas et al., 2024) that incorporates an attention mechanism with edge updates.

Generative Modeling with Flow Matching (Chapter 3)

Flow matching (FM) (Lipman, Chen, et al., 2023; Albergo et al., 2023; Liu et al., 2022; Tong et al., 2023) generalizes diffusion models with a more flexible framework and a simple simulation-free regression objective. Given two sample-able marginal distributions p_0 and p_1, a coupling (x_0, x_1) is sampled from a joint distribution, and x_t from the intermediate distribution $p_t(x_t \mid x_0, x_1)$ conditioned on the endpoints with t sampled within the interval $[0, 1]$. A neural network (velocity model) is then trained to predict the conditional velocity $u_t(x_t \mid x_0, x_1)$, such as $x_1 - x_0$. Marginalizing this linear vector field over the joint coupling then results in a more complex vector field that transforms p_0 to p_1, which is estimated by solving a differential equation with the velocity given by the trained velocity model. This flow matching framework can also be extended to data with more general geometries such as Riemannian manifolds (Chen and Lipman, 2023).

We will train our models using flow matching, with a Gaussian prior and the posterior samples collected through the neural networks' optimization trajectories, experimenting with different couplings. Our slightly modified flow matching training setup is described in more detail in Section 3.2.2.

Flows with Different Geometries (Chapter 6)

We propose three flow models that handle the underlying geometric structure in different ways, which enables us to evaluate the effect of geometric considerations more precisely. We consider ReLU MLPs, and first align all neural networks to the same reference network using the re-basin operation (Ainsworth et al., 2023; Peña et al., 2023) that permutes a network's weights to minimize the loss barrier between two networks. To handle the scaling symmetries, we build on the canonicalization procedure of (Pittorino et al., 2022), where each neuron's incoming weights are normalized and its outgoing weights are inversely scaled to preserve the function being computed; the last layer is further normalized globally in classification networks. This gives the neural network a product geometry, with the bias vectors as Euclidean vectors, intermediate neurons' incoming weight vectors on the hypersphere, and the last layer (optionally) on the hypersphere as a whole.

We then compare our three flows: first a Euclidean flow that ignores the scaling symmetries and models each weight vector in Euclidean space, then a Normalized flow with the weights embedded in the product geometry but with the velocity field defined in Euclidean space (i.e. inside the hyperspheres), and finally a Geometric flow with the velocity field defined on the product geometry as well using the Riemannian Flow Matching framework (Chen and Lipman, 2023).

Overview of Results (Chapter 7)

We evaluate our flows on a variety of tasks after training them on samples obtained with gradient-based optimization methods and show that for small networks on relatively easier tasks, they can directly generate weights matching or sometimes exceeding the performance of weights optimized with gradient-based methods. On more complex tasks and larger models, while direct generation does not match the quality of optimized weights, Bayesian model averaging over a number of samples leads to comparable predictive performance. Then we show that a flow trained on weights from one task can be transferred to another task, either by using the sampled weights as learned initializations, or by guiding the sampling process with task gradients. Our results finally demonstrate that in particular the Euclidean and Geometric flows can also generalize to different architectures for the same base task, and that there are still gains to be made by further scaling up our models.

1.3. Contributions

Our main contributions are as follows:

- We utilize flow matching in weight-space for the first time. Our Euclidean flow learns to transport weights in Euclidean space directly, then we model the neural networks on a product manifold by removing their scaling symmetries to build our Normalized flow, and finally in our Geometric flow we utilize the Riemannian Flow Matching framework to define a vector field over this product manifold.
- We empirically validate our proposed methods by learning to approximate the posterior distributions of neural networks of various sizes on different tasks. We observe that they can generate samples competitive with weights optimized via gradient-based methods, and that can still be scaled up for better performance.
- We show that our flows can also generalize to different architectures, such as by sampling more accurate weights for a larger mode. This indicates they do not learn to model just a single posterior distribution, but instead learn to model weights on the underlying task more generally.
- Finally we apply guidance with task gradients during sampling as an instance of transfer learning and show that a flow built with weights trained on one dataset can be used to sample weights that are accurate on another dataset.

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, followed by a review of inference methods (Laplace approximations, variational inference, MCMC-based methods) typically used for Bayesian neural networks.

2.1. General Concepts

Bayesian deep learning 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 (refer to (David J C MacKay, 1992a; Neal, 1996) for foundational work and (Goan and Fookes, 2020; Arbel et al., 2023) for more recent reviews). With 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 an expectation over the posterior. 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

Specifying a prior mainly consists of two choices: specifying an architecture, and specifying a probability distribution over the weights. A standard choice for a prior distribution is an isotropic Gaussian, which is an uninformative prior but therefore widely applicable and flexible.

Different architectural decisions also induce different distributions over 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 different 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

Inference in BNNs is a rich research area covering a wide range of methods. In this section we give an overview of key methods used to approximate BNN posteriors, and refer to works such as (Arbel et al., 2023) and (Murphy, 2023) for more comprehensive treatments.

2.3.1. Laplace Approximation

A Laplace approximation to the posterior (David J. C. MacKay, 1992b) is essentially a Gaussian distribution centered at the MAP estimate of the posterior, with the covariance matrix given by the log likelihood's Hessian's inverse. This intuitively corresponds to a local second-order Taylor expansion around the MAP estimate.

A Laplace approximation is simple and fast, but ignores the posterior's multi-modality as it only captures a single mode. Nevertheless, it has been an active topic of research (Daxberger et al., 2021) due to its simplicity and for use cases benefiting from local posterior estimates.

2.3.2. Variational Inference

In a variational approximation, the posterior is estimated with a parametric distribution whose parameters are learned to minimize the KL-divergence between the posterior and the variational approximation; e.g. learning the mean and covariance matrix of a Gaussian distribution. The parameters are often learned to optimize the evidence lower bound (ELBO) using reparametrization tricks (Kingma and Welling, 2022; Blundell et al., 2015).

Variational approximations trade off expressivity for tractable optimization and thus can learn more complex distributions than a Gaussian, e.g. by accounting for symmetries in a neural network's posterior landscape (Gelberg et al., 2024). Nevertheless, KL-based objectives may require a large number of likelihood evaluations and often suffer from mode collapse, where the approximation captures only one mode of the distribution (Felardos et al., 2023).

2.3.3. Sampling-Based Inference

Rather than parametrize or approximate the posterior distribution, sampling-based methods which our flow is a part of, attempt to sample from the posterior directly. Two main approaches to sampling are Monte Carlo methods (Barbu and Zhu, 2020) such as Markov chain Monte Carlo or importance sampling, and map-based methods (Marzouk et al., 2016) such as normalizing flows (Rezende and Mohamed, 2015).

Monte Carlo Methods

Monte carlo methods are designed around a few core ideas such that the samples are asymptotically from the true posterior, and are often used to compute expectations of the form

$$\mathbb{E}_{x\sim p(x)}\left[h(x)\right]. \tag{2.2}$$

A simple approach is Importance Sampling (IS) (Kahn and Marshall, 1953), where samples from a proposal distribution q are re-weighted with $w(x) = p(x)/q(x)$ to estimate the expectation in Equation 2.2:

$$\mathbb{E}_{x\sim q(x)}\left[\frac{p(x)}{q(x)}h(x)\right] = \int q(x)\frac{p(x)}{q(x)}h(x)dx = \int p(x)h(x)dx = \mathbb{E}_{x\sim p(x)}\left[h(x)\right]. \tag{2.3}$$

While IS does not require sampling from the posterior, the rate of convergence for the estimate depends on the variance of the IS weights, i.e. how well the proposal distribution aligns with the true posterior.

Alternatively, Markov chain Monte Carlo (MCMC) methods such Hamiltonian Monte Carlo (Neal, 2011), Stochastic Gradient Monte Carlo (Ma et al., 2015), or Langevin

Monte Carlo (Roberts and Tweedie, 1996) construct a Markov chain of samples that eventually converges (mixes) to the true posterior. This is often achieved through a Metropolis-Hastings correction step (Metropolis et al., 1953; Hastings, 1970) where a proposed step z from the current state x is accepted with probability

$$a(x,z) = \min\left\{1, \frac{p(z)}{p(x)}\frac{q(z,x)}{q(x,z)}\right\} \tag{2.4}$$

where $q(x,z)$ gives the transition probability from x to z.

While chain-based methods are heavily used in practice, they suffer from various shortcomings such as generating correlated rather than independent samples, and requiring a large number of likelihood evaluations some of which is later rejected. Measuring the convergence of a chain is also challenging, lacking concrete rules.

Map-Based Sampling

Map-based methods (Marzouk et al., 2016) are instead use a map $T : \mathbb{R}^n \to \mathbb{R}^n$ that transforms samples from the prior distribution q to samples from the posterior p, such that the push-forward of the prior gives the posterior distribution: $T_\sharp q = p$. Such a map can generate independent, uncorrelated samples without any likelihood evaluations which is a desirable property.

The theory of map-based sampling is closely related to optimal transport (OT) (Peyré and Cuturi, 2020; Santambrogio, 2015) with a history going back to the 18th century (Monge, 1781), where the goal is to find the map T that minimizes some cost $c(x, T(x))$ such as Euclidean distance. Under the later relaxation of the problem by Kantorovich (Kantorovitch, 1958) to joint distributions rather than deterministic maps to minimize the cost over the joint distribution with marginal constraints, the optimal map is guaranteed to exists given mild conditions.

Nevertheless, while a map-based approach has benefits over an MCMC method such as providing less ambigous convergence criteria by reducing the problem to optimization, the optimal map is often hard to find, and therefore approximate maps are used instead. Such maps can be learned as static (Rezende and Mohamed, 2015) or continuous normalizing flows (Chen, Rubanova, et al., 2018) which have seen risign interest recently with developments such as flow matching which we now turn to.

3. Flow Models

3.1. Overview

We train our flow using *flow matching* (Lipman, Chen, 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 likelihoods (Sec. 3.4) can be computed using a flow model.

3.2. Flow Matching

Flow matching, first proposed in (Lipman, Chen, et al., 2023; Albergo et al., 2023; Liu et al., 2022) (see (Lipman, Havasi, et al., 2024) for a recent comprehensive overview), 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

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

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

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, Rubanova, et al., 2018), 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. 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

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

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, Chen, 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

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

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

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

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 are equivalent up to a constant and therefore

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

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 the 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 $\nabla_\theta \mathcal{L}_{\text{CFM}}(\theta)$.

An equivalent approach with the loss computed in the space our data lies is to predict the target x_1 rather than the velocity and compute the loss as $\|v_\theta(t, x_t) - x_1\|^2$, and the velocity as $v_\theta(t, x_t) - x_0$ with x_0 the starting point for integration.

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.

3.3. Riemannian Flow Matching

When our data lies on a manifold, flow matching can be extended to define a flow over the manifold as well. Such an approach can both lead to a better scalable generative model by reducing the effective dimensionality of the problem, and make learning easier by introducing a strong inductive bias to the problem. Discrete and continuous normalizing flows have previously been adapted to Riemannian manifolds (Gemici et al., 2016; Mathieu and Nickel, 2020; Lou et al., 2020), and in this section we begin with a brief overview of Riemannian manifolds (we refer to textbooks on the topic such as (John M. Lee, 2018) for a more rigorous treatment), and then explain how an ODE

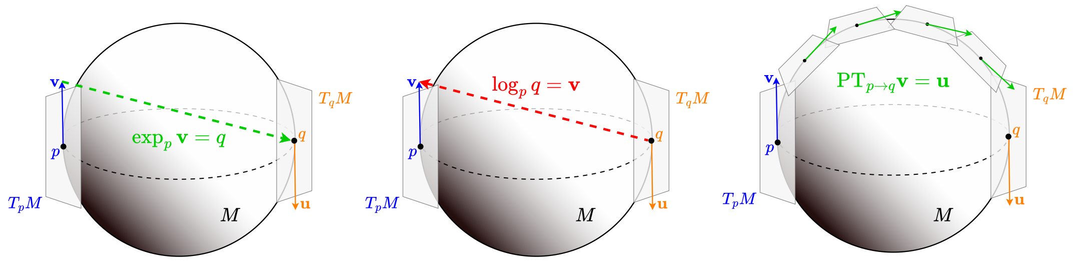


Figure 3.1.: **Visualizing the exponential, logarithmic, and parallel transport maps on a manifold.** The unique geodesic curve γ between p and q with $\dot{\gamma}(0) = \mathbf{v}$ corresponds to the great circle of the sphere coinciding with the upper boundary in the diagrams.

over a Riemannian manifold can be learned following the framework of Riemannian flow matching (Chen and Lipman, 2023).[2]

3.3.1. A Brief Review of Riemannian Manifolds

A smooth *manifold* M is a smooth topological space that is locally Euclidean. The local Euclidean structure is represented by *charts* mapping open sets $U \subset M$ to $\mathbb{R}^n$. A set of charts covering the entire manifold is called an *atlas*, and smoothness arises from the transitions between overlapping charts being smooth functions. Each point $x \in M$ is equipped with a tangent space T_xM that is a vector space containing vectors tangent to M, such as velocities. In addition to being a smooth manifold, a Riemannian manifold also contains a Riemannian metric g that defines an inner product $\langle u, v\rangle_g = u^T G v$ for $u, v \in T_xM$ and positive definite matrix G. A metric most importantly enables us to define angles and distances over the manifold as $\|u\|_g = \sqrt{\langle u, u\rangle_g} = \sqrt{u^T G u}$ and $\cos\theta = \frac{\langle u,v\rangle_g}{\|u\|_g \|v\|_g}$.

A *curve* is a smooth function $\gamma : [0,1] \to M$, and tangent vectors $v \in T_xM$ can be expressed as time derivatives $\dot{\gamma}(t)$ of curves with $\gamma(t) = x$. Using the metric g, we can measure the length of a curve by computing the length of the tangent vector at each point along the curve:

$$|\gamma| = \int_0^1 \|\dot{\gamma}(t)\|_g^2 dt. \tag{3.14}$$

The "shortest" curve in this sense connecting two points $x, y \in M$ is called a *geodesic*. We can thus define a distance between two points as the length of a (not necessarily

[2]The presentation in this section is additionally based on the Geometric Generative Models tutorial by Joey Bose, Alexander Tong, and Heli Ben-Hamu at the 2024 Learning On Graphs Conference.

unique) geodesic connecting them.

A manifold is also equipped with three operations that we will make use of: the *exponential map*, the *logarithmic* map, and *parallel transport* (see Figure 3.1 for a visual depiction). The exponential map $\exp_x : T_x M \to M$ at a point $x \in M$ maps a tangent vector v to the point $\gamma(1) \in M$ where γ is the unique geodesic satisfying $\gamma(0) = x$ and $\dot{\gamma}(0) = v$. The logarithmic map $\log_x : M \to T_x M$ maps a point $y \in M$ back to the tangent vector $v := \dot{\gamma}(0)$. It is generally the inverse of the exponential map. Finally the parallel transport map $\mathrm{PT}_{x \to y} : T_x M \to T_y M$ transports a tangent vector $v := \dot{\gamma}(0)$ along the geodesic with $\gamma(1) = y$ keeping the lengths and angles between the transported vectors constant. Together, these three operations allow us to move along geodesics with tangent vectors as the velocities, and transport tangent vectors to the same tangent space to be able to compute distances between them, which are all steps required to build a flow model over a Riemannian manifold.

3.3.2. Flow Models over Manifolds

Riemannian flow matching (RFM) (Chen and Lipman, 2023) extends the flow matching framework to Riemannian manifolds. As the three key design choices, we need to define a coupling q, conditional probability paths p_t, and a conditional vector field u_t.

A coupling can be defined simply by sampling $x_0 \sim p_0$ and $x_1 \sim p_1$ independently, or with a mini-batch optimal transport map with the distances defined above using geodesics. For $t \in [0, 1]$, the geodesic interpolation $x_t \sim p_t(x_t \mid x_0, x_1)$ (analogous to linear interpolation in Euclidean space) between x_0 and x_1 is computed as

$$x_t := \exp_{x_0}(t \log_{x_0} x_1) \tag{3.15}$$

and the conditional vector field u_t as

$$u_t(x_t \mid x_0, x_1) := \frac{\log_{x_t} x_1}{1 - t}. \tag{3.16}$$

Finally to solve the ODE, we compute one Euler integration step as

$$dx_t = u_t(x_t)dt \tag{3.17}$$

$$x_{t+1} = \exp_{x_t}(v_\theta(t, x_t)\Delta t) \tag{3.18}$$

where Δt is the step size.

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

$$\log p_1(z) = \log p_0(z) - \log \det \left| J_f(z) \right| \tag{3.19}$$

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, Rubanova, 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

$$\begin{aligned} \frac{d \log p_t(\phi_t(x))}{dt} &= -\operatorname{div}(v_t(\phi_t(x))) && (3.20) \\ &= -\operatorname{Tr}\left(\frac{dv_t(\phi_t(x))}{dt}\right) && (3.21) \end{aligned}$$

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

$$\log p_1(\phi_1(x)) = \log p_0(\phi_0(x)) - \int_0^1 \operatorname{Tr}(J_v(\phi_t(x)))dt. \tag{3.22}$$

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 very 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:

$$\operatorname{Tr}(J_v) = \mathbb{E}_{p(\epsilon)}\left[\epsilon^T J_v \epsilon\right] \tag{3.23}$$

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 3.22 and compute the likelihoods as

$$\log p_1(\phi_1(x)) = \log p_0(\phi_0(x)) - \int_0^1 \mathbb{E}_{p(\epsilon)}\left[\epsilon^T J_v(\phi_t(x))\epsilon\right] dt. \tag{3.24}$$

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, Chen, et al., 2023; Song et al., 2021).

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

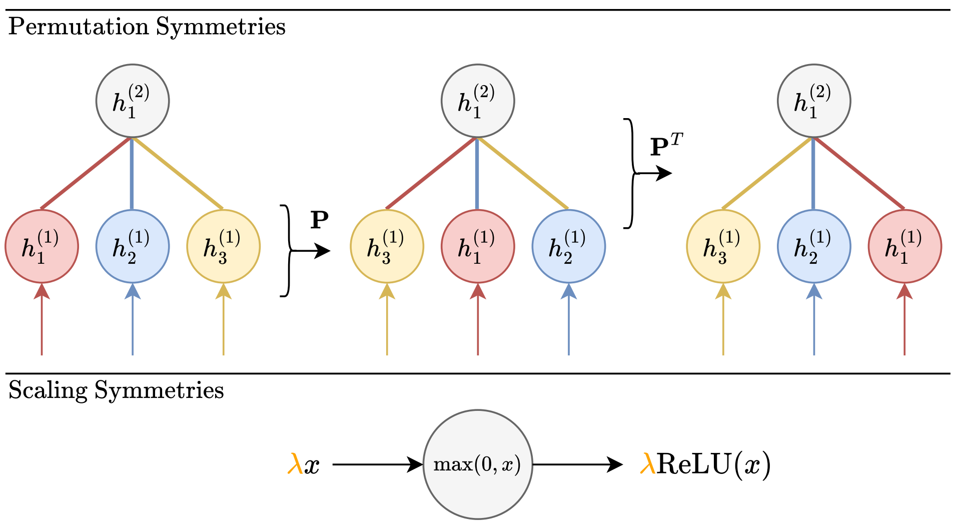


Figure 4.1.: **Visual illustration of example permutation and scaling symmetries of neural networks.** *Top*: Permuting the neurons at one layer and then applying the same permutation to the outgoing weights preserves the function being computed. *Bottom:* For a ReLU activation, multiplying the input with a non-negative constant and the output with its inverse preserves the function.

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

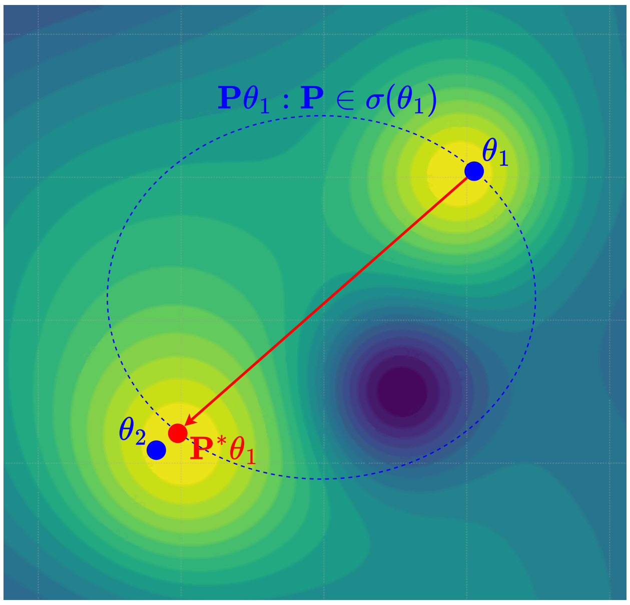


Figure 4.2.: **Linear mode connectivity**. The hypothesis asserts that up to permutations, low-loss points in a neural network's loss landscape are linearly connected.

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).

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. Weight-Space Learning

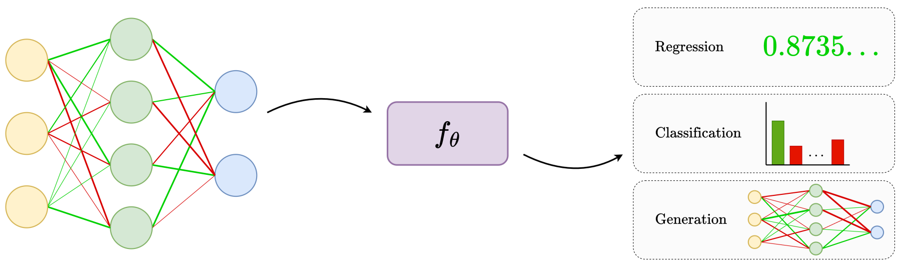


Figure 5.1.: **Weight-space learning**. Neural network weights are processed using other neural neural networks for tasks such as regression (e.g. predicting the loss of unseen weights), classification (e.g.), or generation (e.g. learned optimization).

Training neural networks on other neural networks' weights has been an active research area for a long time (Ha et al., 2016; Krueger et al., 2018), but has seen increasing interest recently with new neural network architectures being proposed (Lim, Maron, et al., 2023; Kofinas et al., 2024; Zhou, K. Yang, et al., 2023; Zhou, Finn, et al., 2024). This wave of interest, combined with developments around other problems such as generative modeling, has led to a wider range of applications of weight-space learning, including generative modeling of neural network weights (Peebles et al., 2022; Erkoç et al., 2023) and machine unlearning using weight-space models (Rangel et al., 2024). In this section, we provide a review of recent work on weight-space architectures and weight-space generative models.

5.1. Architectures

5.1.1. Graph Neural Networks

A graph neural network (GNN) (Wu et al., 2022) takes as input a graph (V, E) with nodes $n_i \in V$ and edges $e_{ij} \in E$ with i, j node indices, and operate by iteratively updating the node and edge features of d_V and d_E dimensions respectively. Although edge features may be omitted, a general node and edge GNN update step can be expressed as

$$n_i^{l+1} = \phi_N \left(n_i, \bigoplus_{j \in N_i} \phi_M(e_{ij}^l, n_i^l, n_j^l) \right) \tag{5.1}$$

$$e_{ij}^{l+1} = \phi_E \left(e_{ij}^l, e_{ij}^l, n_i^{l+1}, n_j^{l+1} \right) \tag{5.2}$$

where ϕ_N, ϕ_M, ϕ_E are the node, message, and update neural networks, and $\bigoplus$ is a permutation-invariant aggregation operation. N_i is the *neighborhood* of node i which the update message is aggregated over. While directly using the connectivity structure of the input graph is a typical choice, methods using the entire graph (Diao and Loynd, 2023) or dynamically learning a structure (known as graph rewiring) (Gutteridge et al., 2023) also exist. Note that the weights of the neural networks are shared across nodes/edges and update steps, making the number of parameters in a GNN independent of the input graph's size.

Different GNNs are characterized by how they construct the neural networks ϕ_N, ϕ_M, ϕ_E. Graph convolutional networks (Kipf and Welling, 2016) set $\phi_M(e_{ij}^l, n_i^l, n_j^l) = \phi_M(n_j^l)$ and $\bigoplus$ to be averaging. Graph attention networks (Velikovi et al., 2018) replace the average with self-attention. Transformers (Vaswani et al., 2017) can also be classified as GNNs, where $N_i = V$ and ϕ_M is self-attention.

5.1.2. Graph Neural Networks in Weight-Space

Since a neural network can be represented as a graph via its computational graph, GNNs make for a natural choice in constructing weight-space architectures, and there has been a recent line of work in this direction (Zhou, K. Yang, et al., 2023; Lim, Maron, et al., 2023; Kofinas et al., 2024; Kalogeropoulos et al., 2024). We build on the architecture of (Kofinas et al., 2024) and explain its workings in this section.

Construcing Graphs from Neural Networks

An MLP with L layers, consisting of weight matrices $\{\mathbf{W}^{(1)}, ..., \mathbf{W}^{(L)}\}$ and bias vectors $\{\mathbf{b}^{(1)}, ..., \mathbf{b}^{(L)}\}$ with $\mathbf{W}^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$ and $b^l \in \mathbb{R}^{d_l}$ can be considered a graph directly following its computational graph, with nodes corresponding to the neurons and edges to the parameters. Then in its simplest form, edge features are individual parameters, and node features are the individual bias components, with $d_V = d_E = 1$, although including additional features such as "probe features" that correspond to the activations of certain inputs are also possible. This construction, since it follows the computational graph, respects the permutation symmetries of nodes in subsequent layers as described in Chapter 4.

For convolutional neural networks (CNNs), the permutation symmetries occur at the level of individual channels. Permuting the filters in one layer permutes the channels at the subsequent layer, and permuting the filters of the subsequent layer with the same permutation preserves the function being computed. We can obtain this symmetry as a graph by associating nodes with individual channels and edges with the filters, where a single edge's features correspond to the flattened weights in a zero-padded filter to make all edge features the same size.

Learning over Neural Network Graphs

A standard GNN can be used to learn a function over NN weights with the graph constructed as above. However, not all GNNs incorporate edge features. Kofinas et al. (2024) propose two architectures, one extending the *Principal Neighborhood Aggregation* network (PNA) (Corso et al., 2020) with edge updates, and another a slightly updated version of a *Relational Transformer* (Diao and Loynd, 2023).

PNA (Corso et al., 2020) builds a message-passing scheme by combining various aggregators and scalers to obtain the aggregation operation

$$\bigoplus = \begin{bmatrix} I \\ S(D, \alpha = 1) \\ S(D, \alpha = -1) \end{bmatrix} \otimes \begin{bmatrix} \texttt{mean} \\ \texttt{std} \\ \texttt{max} \\ \texttt{min} \end{bmatrix} \tag{5.3}$$

where

$$S(d, \alpha) = \left(\frac{\log(d+1)}{\delta} \right)^{\alpha} \tag{5.4}$$

scales the messages to reduce the effect of exponential changes in the messages. Updates then follow typical message-passing as in Equation 5.1, including edge features. The original formulation does not include edge updates, but Kofinas et al. (2024) add an update mechanism as in Equation 5.2, as well as feature-wise linear modulation (Brockschmidt, 2020) based on the edge features.

The **Relational Transformer** (Diao and Loynd, 2023) is an extension of the traditional Transformer (Vaswani et al., 2017) to graphs with edge features. Node and edge updates follow the typical message-passing operations in Equations 5.1, 5.2. Unlike typical attention where QKV vectors are obtained through node features alone, Diao and Loynd (2023) construct the QKV vectors by concatenating node and edge features; i.e.

$$q_{ij} = [n_i, e_{ij}]\mathbf{W}^Q \quad k_{ij} = [n_i, e_{ij}]\mathbf{W}^K \quad v_{ij} = [n_i, e_{ij}]\mathbf{W}^V \tag{5.5}$$

where each weight matrix has two components, one for edges and one for nodes to obtain

$$q_{ij} = \left(n_i \mathbf{W}_N^Q + e_{ij}\mathbf{W}_E^Q\right) \quad k_{ij} = \left(n_i \mathbf{W}_N^K + e_{ij}\mathbf{W}_E^K\right) \quad v_{ij} = \left(n_i \mathbf{W}_N^V + e_{ij}\mathbf{W}_E^V\right). \tag{5.6}$$

The graph can be taken to be fully-connected, ignoring the original structure, but the architecture can be adapted to other forms of connectivity just by changing which nodes and edges each update is conditioned on.

5.2. Weight-Space Generative Models

One of the earliest approaches to using a neural network to learn a generative model in weight-space is the *Bayesian Hypernetworks* of Krueger et al. (2018). A Bayesian hypernetwork converts noise to a sample weight θ from the approximate posterior $q(\theta)$. Making the hypernetwork invertible makes it possible to compute the log-determinant of its inverse Jacobian, similar to a normalizing flow. The hypernetwork and the base network can then be trained using backpropagation as a single model, where the hypernetwork outputs weights and the primary network is used to evaluate the "true" likelihood in a differentiable way.

Although learning a generative model only with access to a likelihood function is still a very active research area (e.g. works such as (Tong et al., 2023)), simulation-free regression objectives such as score/flow-matching have seen increasing interest, as they are cheaper to optimize and can avoid certain failure modes of likelihood-based objectives such as mode-seeking. This interest has also led to various applications of such methods in weight-space.

Using the denoising diffusion objective, Peebles et al. (2022) train a GPT-2 model (Radford et al., 2019) (omitting causal masking) over neural network weights, where each token is a vector of weights. The model, named **G.pt**, is trained using checkpoints from a large number of training runs augmented with permutations to obtain different representations of the same function. The model can then also be "prompted" for specific losses and succeeds at generating weights that correlate with the prompted losses.

In an alternative approach, Schürholt, Mahoney, et al. (2024) aim to train an autoencoder that learns to map neural network weights to a lower-dimensional latent space, which can then facilitate various downstream tasks including generation. Each complete weight vector is tokenized and split into chunks, and each chunk has a separate latent representation. New weights are then sampled by fitting a kernel density estimator around the embeddings of the weights given as a prompt. The samples can then be iteratively refined by using the best samples as the new prompt and repeating this procedure.

6. Building Flows in Weight-Space

We now bring the preceding background together and describe how we can build flows in weight-space. We propose three alternative formulations based on how the scaling symmetries are handled: first a Euclidean flow defined in weight-space directly, and then the Normalized and Geometric flows with the weights normalized to remove the scaling symmetries, differing on whether the vector field is also defined in Euclidean space (Normalized flow) or on the manifold of normalized weights (Geometric flow). We conclude by describing by how we train and sample from our flows.

6.1. Training With and Without Samples

Although simulation-free methods to train generative models with score/flow-matching objectives only given access to an unnormalized density function exist (Akhound-Sadegh et al., 2024), we train our models on samples obtained using typical gradient-based optimization of neural networks. This is an easier setup and allows us to measure the impact of various design choices more clearly, and potentially also benefit from publicly available datasets of neural network weights (Schürholt, Taskiran, et al., 2022; Peebles et al., 2022). Thus to train our flow models, we independently train a number of neural networks on the base task and intermittently save the weights along each optimization trajectory, discarding the initial steps based on validation loss.

6.2. Flows with Different Geometries

The first step in designing a flow is to determine the space our data lies in. We propose three different approaches with different geometric properties, illustrated in Figure 6.1: first a Euclidean flow operating directly over the trained weights, and then two different methods operating with the normalized weights as described in Section 4 following the process of (Pittorino et al., 2022). The last two methods further differ on whether the vector space is also over these normalized weights or not.

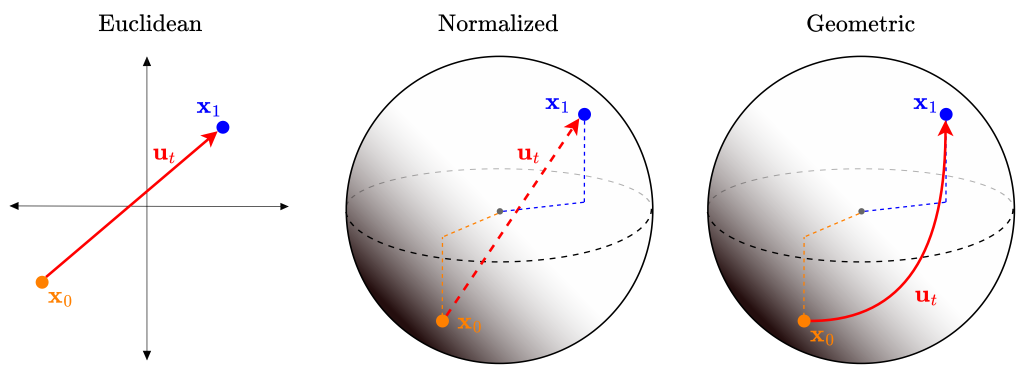


Figure 6.1.: **Flows with different geometric structures.** *Left:* Euclidean flow defined over in weight-space without modifying the weights. *Center:* Normalized flow after removing scaling symmetries, with the vector field defined as in Euclidean space. *Right:* Geometric flow, with the vector field defined over the particular geometry as well.

6.2.1. Euclidean Flow

Our first method we call the Euclidean flow is defined in Euclidean space, and the trained weights are used directly, without any transformations except permutation alignment (re-basin) to a common reference. Following the presentation in Section 3.2, we define the ground truth vector field as $x_1 - x_0$. Such a flow can be straightforwardly applied to any neural network, but ignores the scaling symmetries resulting from activations such as ReLU.

6.2.2. Normalized Flow

A simple modification to the Euclidean flow above is to first apply the normalization procedure of (Pittorino et al., 2022) as shown in Figure 6.2. For ReLU networks, we normalize the incoming weight vector and bias of each intermediate neuron, and scale the outgoing weight vector up by the same normalization factor to preserve the function being computed by the neural network. Since there is no such symmetry for the last layer, we normalize the entire last layer weights for a classification network as that would preserve the output class.

This procedure embeds neural network weights in a product geometry. Each intermediate neuron's incoming weight vector, and the last layer in its entirety, now lie on unit hyperspheres of different dimensions. The bias vectors have no such structure and we still treat them as Euclidean vectors. Our Normalized flow works with source

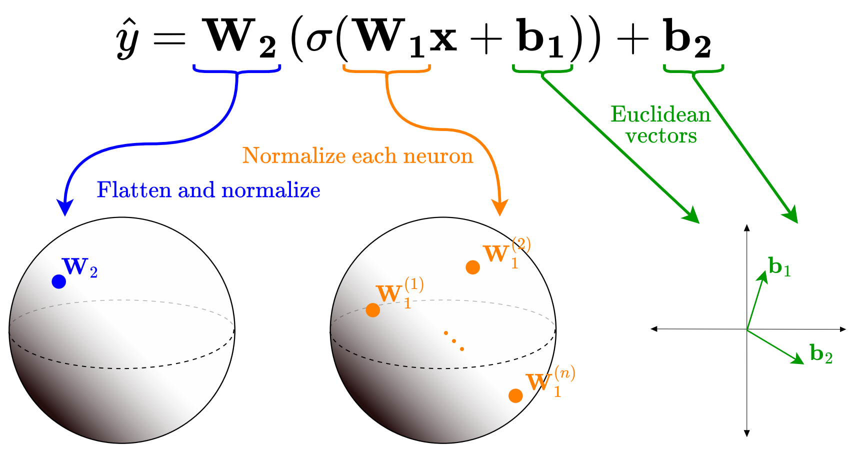


Figure 6.2.: **Removing scaling symmetries from ReLU networks.** Each intermediate neuron lies on a unit hypersphere, as well as the last layer as a whole. Bias vectors are also rescaled to preserve the neural network's function but remain Euclidean vectors.

and target distributions on this product geometry, but the vector field is still defined in Euclidean space; i.e. inside the hyperspheres for neurons and in Euclidean space for biases. This is an intermediate step between the previous Euclidean flow and the Geometric flow we will describe next.

6.2.3. Geometric Flow

Since hyperspheres, as well as the Euclidean space and a product of Riemannian manifolds, are Riemannian manifolds, we can model vector fields over them using the framework of Riemannian flow matching (Chen and Lipman, 2023) (Section 3.3). For the bias vectors in Euclidean space, we again define $u_t := x_1 - x_0$ and $x_t = tx_1 + (1-t)x_0$. For intermediate neurons and the last layer on different hyperspheres, we have $x_t = \exp_{x_0}(t \log_{x_0} x_1)$ and $u_t = \log_{x_t}(x_1)/(1-t)$. This formulation, while computationally more expensive, has the added potential benefit that all inputs including the intermediate points x_t to our model will lie on this particular geometry, reducing the effective dimensionality of the problem.

While we focus only on ReLU networks, this geometric formulation can be extended to other non-linearities as they also induce different kinds of scaling symmetries (Godfrey et al., 2022).

6.3. Model Architecture

For each of our three flows, we experimented with the PNA and Relational Transformer architectures and decided to use the Relational Transformer model due to its superior performance with fewer parameters. We thus model the vector field using a Relational Transformer with edge updates (Diao and Loynd, 2023; Kofinas et al., 2024), and embed the time by appending it to each node and edge feature. For an MLP as the base model, each edge in the graph corresponds to a single weight, and nodes to individual bias values. We project each node and edge feature to d_E dimensions using an MLP, append t to each node and edge, and run a number of node/edge update steps before projecting the final node/edge states back to one dimension using an MLP. To distinguish nodes at different layers, we add layer-specific learned positional embeddings to edge features.

6.4. Training

We train our models using the conditional flow matching **objective** (Equation 3.9), but rather than predict the velocity u_t, we predict the target point x_1 and compute the velocity during integration using the initial point x_0 and intermediate point x_t. This makes training easier to analyze as the error in weight-space is more directly informative, and we can constrain the outputs to the particular geometry for the Normalized and Geometric flows.

As **prior** distributions p_0, we use isotropic Gaussian distributions of different variances, ensuring that the prior distribution is the same as the distribution used to initialize the weights while training the base model, although this is not a strict requirement and our flow models can be trained with arbitrary priors. We also experiment with independent and mini-batch optimal transport **couplings**, although especially in high-dimensions the bias induced by the mini-batch approximation to optimal transport can be significant (Fatras et al., 2021).

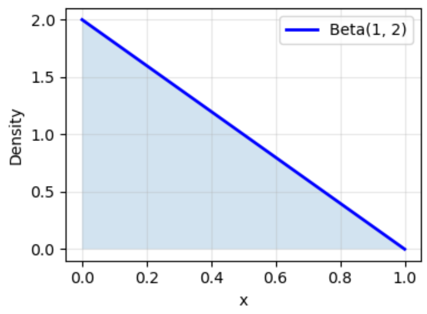


Figure 6.3.: **PDF of Beta**$(1,2)$**.** We sample $t \sim \text{Beta}(1,2)$ rather than uniformly when training our flow model.

As another modification over the standard flow matching framework, we sample **time** $t \in (0,1)$ during training not uniformly, but following $\text{Beta}(1,2)$ (Figure 6.3). Since x_t with smaller t are

further away from x_1, estimation error increases
for smaller time values. Sampling $t \sim \mathrm{Beta}(1,2)$ results in a larger number of parameter updates for smaller t relative to larger t, which is a more effective distribution of the computational budget since the predictions for larger t are closer to x_1 and thus converge in fewer steps. Importantly, sampling $t \sim \mathrm{Beta}(1,2)$ does not bias the training process, since a prediction at a certain time is independent of the model behavior at other times.

6.5. Sampling

To sample from our flow models, we solve the ODE (Equation 3.1) using an Euler solver with step sizes depending on the complexity of the task, although higher-order solvers can also be used without any modification to the rest of the setup. We also experiment with **stochastic sampling** by adding noise before computing Euler updates, on the entire trajectory or a subset of it (Karras et al., 2022).

6.5.1. Guidance

We can also guide the sampling process with loss gradients from the base task (Wang et al., 2024; Kulyt et al., 2024; Yu et al., 2024), so that for base model f and velocity model v_θ Euler updates take the form

$$x_{t+\Delta t} = x_t + \left(v_\theta(t, x_t) + \lambda \nabla_{x_t} \mathcal{L}(f, x_t)\right) \Delta t \tag{6.1}$$

where $\mathcal{L}(f, x_t)$ is computed with training data points from the base task (note that x_t are neural network weights from the base task), and λ is a coefficient controlling the strength of this guidance. If the task loss $\mathcal{L}$ is computed with mini-batches, this guidance also adds an implicit stochasticity to each step of the sampling process.

7. Experimental Results

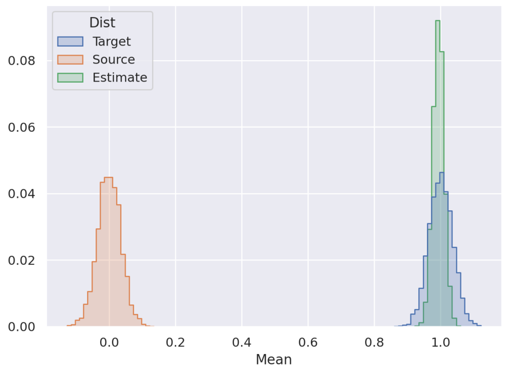
We now evaluate our three flows and various choices on different models and datasets. The main experimental outcomes can be summarized as follows:

- As a sanity check, we can train a flow between two Gaussian distributions, and stochastic sampling helps capture the variance in the posterior (Section 7.1).
- On an easier task with a small model, all three flows can directly sample weights matching and even exceeding the quality of optimized weights (Section 7.2).
- The Geometric and Normalized flows' sample qualities plateau after a smaller number of integration steps, indicating they learn shorter paths than the Euclidean flow (Section 7.2).
- On a more difficult task and larger models, while individual samples perform worse than optimized weights, Bayesian model averaging with a small number of samples results in comparable performance (Section 7.3.1).
- A similar sample quality can also be transferred to other tasks through guiding with task gradients during sampling (Section 7.3.3).
- Using the samples weights as initialization to train models on a different task leads to faster convergence (Section 7.3.3).
- The Euclidean and Geometric flows successfully generalize to a larger architecture on the same dataset, indicating they learn properties not only of the specific architecture but the underlying task (Section 7.3.4).
- There is still expected performance gains from using larger models than the ones we have used in our evaluations (Section 7.3.5).

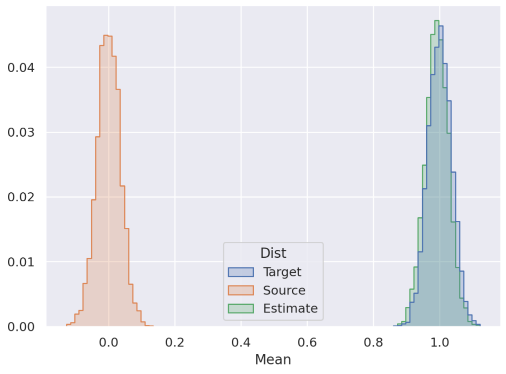
We describe the detailed experimental setups in Appendix A.

7.1. Euclidean Flow Between Two Gaussians

To verify our approach of learning a flow model in weight-space, we begin our evaluation the toy task of learning a flow between two Gaussian distributions. The neural



(a) Deterministic sampling ($\varepsilon = 0$)



(b) Stochastic sampling ($\varepsilon = 0.05$)

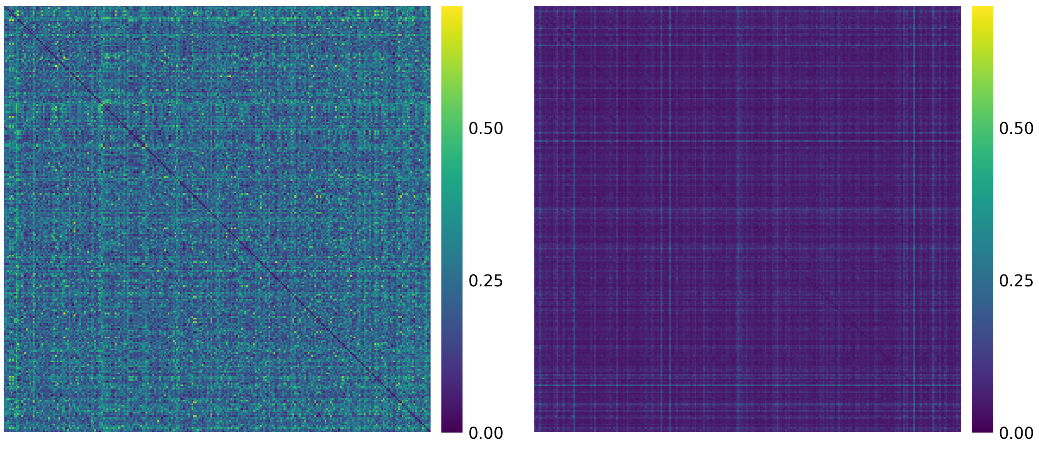
Figure 7.1.: **Histograms for the means of the weights generated by a Euclidean flow trained between two Gaussian distributions.** The flow fails to capture the variance in the target distribution with deterministic sampling, but this is corrected by stochastic sampling with $\varepsilon = 0.05$.

network is a small MLP with 30 input, two output dimensions, and two hidden layers of 16 neurons. We sample $X_0 \sim p_0 := \mathcal{N}(0, \mathbf{I})$ and $X_1 \sim p_1 := \mathcal{N}(1, \mathbf{I})$, and train our Euclidean flow to map p_0 to p_1 with independent coupling $q(x_0, x_1) = p_0(x_0)p_1(x_1)$. This is a relatively simpler task than learning over actual weights since each dimension of the weight vectors is sampled independently.

Figure 7.1 shows histograms of the means of the weights sampled from the flow with 100 Euler steps, and either deterministic or stochastic ($\varepsilon = 0.05$) sampling. Independent of the sampling method used, the flow covers the high-density center of the target distribution well, but the weights sampled deterministically fail to capture the variance in the target distribution. Stochastic sampling however appears to correct for this over-saturation and leads to more diverse samples. Overall, these results validates the feasibility of learning a flow model in weight-space using graph neural networks, and we move on to tasks involving actual learned weights.

7.2. Classification with a Small Model

After the toy Gaussian example, we move on to a relatively simple classification task to demonstrate that our flows can generate high-quality samples and compare the various design choices outlined in Chapter 6. The target model is an MLP with two hidden layers of 16 dimensions each and ReLU activations, on the binary classification task of the UCI Wisconsin Breast Cancer Diagnostic dataset (Street et al., 1993). We train



(a) Unaligned (0.244 ± 0.009) (b) Aligned (0.070 ± 0.001)

Figure 7.2.: **Loss barriers for 250 weights across different optimization trajectories** (mean and standard deviations in parentheses) on the UCI Wisconsin Breast Cancer Diagnostics dataset. Aligning all weights to a single reference significantly reduces the loss barriers between the weights.

Flow	**Accuracy**	**Loss**
Euclidean	0.998 ± 0.006	0.101 ± 0.050
Euclidean (aligned)	0.998 ± 0.006	0.070 ± 0.040
Euclidean (aligned + OT)	0.993 ± 0.010	0.053 ± 0.028
Normalized	0.993 ± 0.009	0.027 ± 0.014
Normalized (aligned)	0.989 ± 0.011	0.030 ± 0.015
Normalized (aligned + OT)	0.988 ± 0.018	0.044 ± 0.047
Geometric	0.992 ± 0.011	0.019 ± 0.009
Geometric (aligned)	0.993 ± 0.001	0.018 ± 0.001
Geometric (aligned + OT)	0.991 ± 0.011	0.020 ± 0.012
Target	0.992 ± 0.010	0.048 ± 0.032

Table 7.1.: **Test accuracy and loss (± std) of the samples generated by flows with different design choices.** Euclidean and Normalized flows result in the most accurate weights, with Normalized flow generating lower-loss weights.

all flows on the same dataset consisting of weights sampled from 100 independent trajectories optimized with Adam (Kingma and Ba, 2017).

In addition to the availability of a large number of trained weights, this setup represents a preferable scenario in the sense that aligning all weights to a single reference eliminates the loss barriers between them to a large extent. Figure 7.2 shows the pairwise loss barriers (midpoint of the linear interpolation) before and after alignment for 250 randomly sampled weights from the set of Adam-optimized weights. The loss barriers are reduced considerably (from an average of 0.244 to 0.07) which supports the linear mode connectivity hypothesis for this setup. This should presumably make it easier to approximate the posterior distribution, as in the extreme case of zero-loss barriers between *all* pairs of weights and assuming they capture all the modes, the posterior would be convex.

Sample Quality. Table 7.1 compares the predictive quality of the samples generated by different flows with or without alignment and mini-batch OT couplings. All flows can generate samples with almost perfect accuracy, matching or exceeding (although not significantly) the performance of Adam-optimized weights. Noticeably, the Geometric flow generates samples with lower loss than the Normalized flow, highlighting the benefit of modeling the vector field over the product geometry of normalized weights as well. Aligning all the weights to a reference before training, and training the flow with mini-batch OT couplings both decrease the loss of the generated samples for the Euclidean flow, but the same effect is not visible for the Normalized and Geometric flows.

Number of Integration Steps & Guidance. Figure 7.3 displays how the predictive performance of generated weights changes with the number of Euler steps performed to integrate the ODE. The main outcomes are:

- Normalized and Geometric flows learn straighter paths compared to the Euclidean flow, apparent from the samples from the Euclidean flow showing a more significant improvement in quality going from 8 to 64 Euler steps.
- For all three flows, guiding the integration with task gradients improves sample quality, but increasing the number of integration does not appear to significantly increase the effect of guidance.

7.3. MNIST Classification

We now move on to a harder classification task and larger models to evaluate different use cases of a weight-space flow. We start with the MNIST dataset, and an MLP with a single hidden layer of 10 neurons and ReLU activations, following the setup used in (Peebles et al., 2022), but we sample a smaller dataset over 20 independent optimization trajectories.

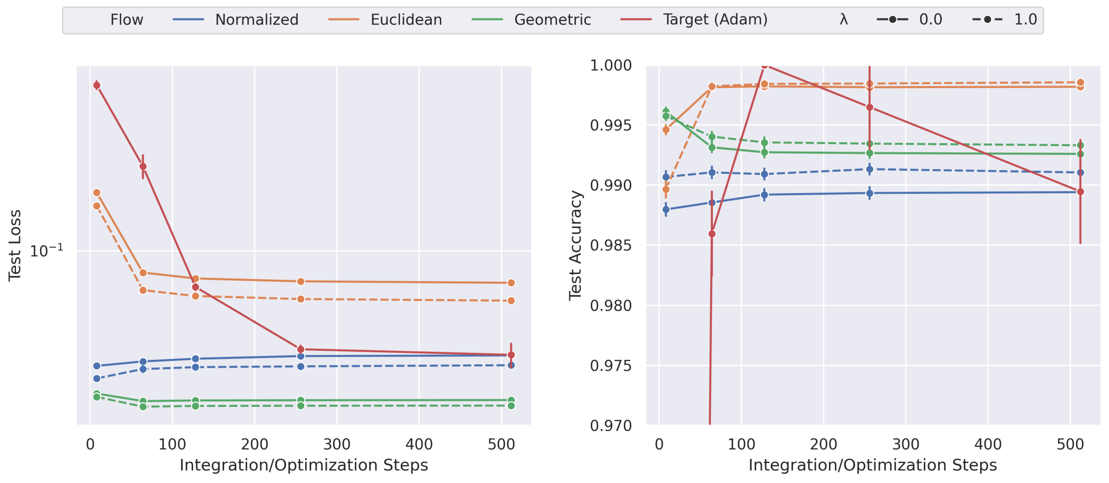


Figure 7.3.: **Comparing the quality of the generated weights with Adam-optimized weights.** Weights generated by the Euclidean and Normalized flows have higher accuracy and lower loss than Adam-optimized weights, while the Geometric flow generated less accurate weights. Guidance during sampling improves sample quality.

This corresponds to a harder task than the UCI classification of the previous section for several reasons. The base MLP has around an order-of-magnitude of more parameters (7,960 rather than 802 parameters) while our GNN is in fact smaller. Moreover, the optimized weights do not reach perfect accuracy unlike in the UCI classification task.

7.3.1. Sample Quality and Diversity

Table 7.2 shows the predictive performance and functional diversity of the weights generated by different kinds of flows trained on aligned weights. Functional diversity is a desirable property of a weight-space generative model, as more diverse weights could correct for each other's errors and make model averaging more effective. We focus on functional rather than parametric diversity since even if two neural networks do not have identical parameters, they could be computing the same function, and thus provide no benefit for model averaging. To measure the functional diversity in a set of neural networks, we compute the average pairwise Jensen-Shannon divergence (JSD) (Endres and Schindelin, 2003; Mishtal and Arel, 2012) between the output (class probabilities) distributions of neural networks in the set. For a set of weights $S =$

Flow	Accuracy	Loss	Diversity
Euclidean (aligned)	0.737 ± 0.085	0.814 ± 0.286	0.00102 ± 0.00114
Euclidean (aligned + OT)	0.757 ± 0.077	0.753 ± 0.245	0.00085 ± 0.00094
Normalized (aligned)	0.753 ± 0.074	1.537 ± 0.046	0.00086 ± 0.00055
Normalized (aligned + OT)	0.706 ± 0.078	1.608 ± 0.047	0.00117 ± 0.00074
Geometric (aligned)	0.737 ± 0.070	1.457 ± 0.040	0.00073 ± 0.00049
Geometric (aligned + OT)	0.786 ± 0.064	1.443 ± 0.046	0.00093 ± 0.00063
Target	0.933 ± 0.009	0.231 ± 0.027	0.00002 ± 0.00001

Table 7.2.: **Test accuracy, loss (± std), and diversity of the samples generated by flows for classification on the MNIST dataset after alignment.** Geometric flow with OT couplings results in the best-performing samples, but most flows are within one standard deviation of each other.

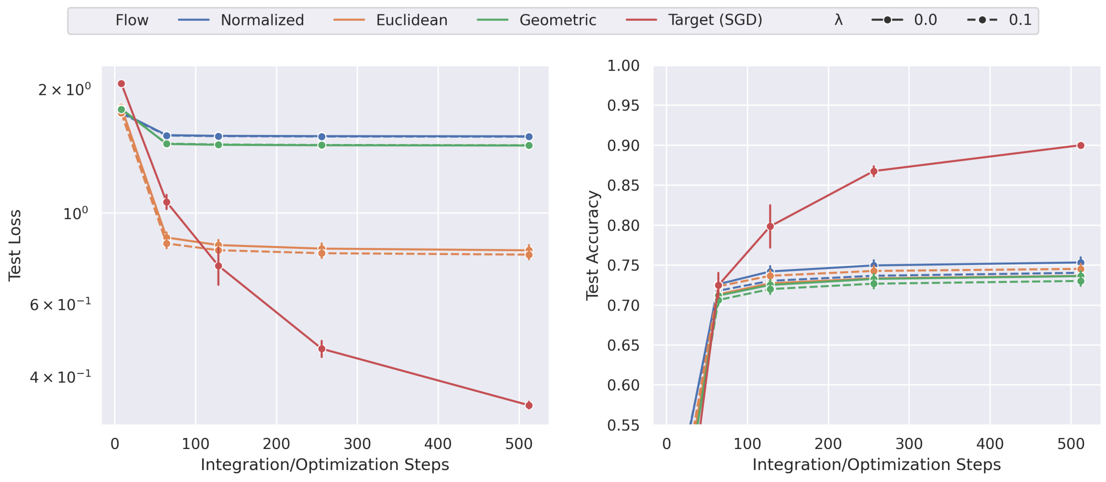


Figure 7.4.: **Comparing the quality of the generated weights with SGD-optimized weights.** After a very small number of steps, sampled weights are comparable in performance with SGD-optimized weights but plateau after around 64 steps. Geometric and Normalized flows again appear to lead to straighter paths.

$\{\theta_i\}_{i=1}^N$ and model f, we fix input points $\{x_i\}_{i=1}^K$ and compute

$$\text{Diversity}(S) := \frac{1}{N^2} \sum_{i,j} \text{JSD}\left(\{f(\theta_i, x_l)\}_{l=1}^K, \{f(\theta_j, x_l)\}_{l=1}^K \right) \tag{7.1}$$

with JSD defined as

$$\mathrm{JSD}(A,B) := \frac{1}{2}\sum_{x\in A} p_A(x)\log\left(\frac{2p_A(x)}{p_A(x)+p_B(x)}\right) + \frac{1}{2}\sum_{x\in B} p_B(x)\log\left(\frac{2p_B(x)}{p_A(x)+p_B(x)}\right) \tag{7.2}$$

and we fit kernel density estimators to both sets to estimate the densities p_A and p_B.

Table 7.2 displays the test accuracy, loss, and diversity (Equation 7.1) of the samples generated with each flow, trained after alignment, and with independent as well as mini-batch OT couplings. While the accuracy of sampled weights are generally within one standard deviation of each other, the Geometric flow with OT couplings results in the highest accuracy. However, unlike the previous UCI classification task, the sampled weights individually perform worse than optimized weights.

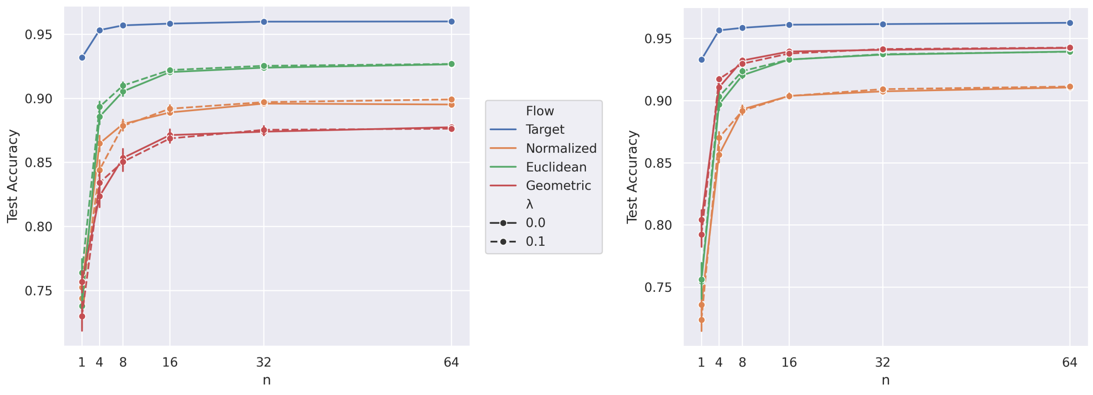
The output distributions of the generated weights shows higher diversity than SGD-optimized weights, although this is also influenced by the generated weights performing worse. Nevertheless, as we will observe in the next chapter, this diversity makes model averaging more effective and considerably increases the predictive performance of generated weights.

Figure 7.4 shows how sample quality varies with the number of steps for our three flows trained with independent couplings, and sampled with and without guidance. Unlike the previous task where the sampled weights were directly of almost perfect accuracy, the sample quality for MNIST flows plateau after a certain number of steps, and SGD-optimized weights show superior performance. Similar to the results on the UCI classification task, Normalized and Geometric flows also show less improvement with a larger number of steps, indicating that the flows have learned straighter paths then the Euclidean flow.

7.3.2. Posterior Predictive Performance

We now evaluate the predictive performance of the sampled weights after Bayesian model averaging; i.e. for each flow, we sample from its push-forward distribution $\hat{p}_1(\theta)$ and compute a Monte Carlo estimate for the expectation $\mathbb{E}_{\hat{p}_1(\theta)}\left[p(y|x,\theta)\right]$, as in Equation 2.1. Figure 7.5 displays the accuracy of predictions averaged over increasing numbers of samples from $\hat{p}_1$, comparing our three flows as well as independent and mini-batch OT couplings. With independent couplings, the Euclidean flow results in the highest accuracies, noticeably reaching the accuracy of individual SGD-optimized weights.

Training the flows with OT couplings improves the predictive performance of all three after model averaging. The improvement for the Geometric flow is more significant than that of the Euclidean and Normalized flows, moving from $\sim$ 87% to 94%.



(a) Independent Couplings

(b) OT Couplings

Figure 7.5.: **Accuracy of the predictions averaged over various numbers of samples from the flows' push-forward distributions, comparing independent and OT couplings.** The Euclidean flow shows the highest accuracy, reaching the accuracy of individual SGD-optimized weights.

Flow	**Accuracy**	**Loss**
Adam-optimized	0.908 ± 0.003	0.334 ± 0.008
With Guidance		
Euclidean	0.754 ± 0.082	0.764 ± 0.252
Normalized	0.724 ± 0.081	1.543 ± 0.045
Geometric	0.730 ± 0.067	1.470 ± 0.043
No guidance	0.080 ± 0.030	9.601 ± 1.402

Table 7.3.: **Predictive performance of samples from the MNIST flows with guidance ($\lambda = 0.1$) from Fashion-MNIST task gradients.** Although the base model (Adam) accuracy is lower than for MNIST, samples obtained with guidance reach an accuracy similar to that of the samples obtained without guidance on MNIST, indicating that a flow trained on one dataset can be transferred to another dataset via guidance.

This supports the results in Table 7.2 that the Geometric flow benefits the most from OT couplings.

7.3.3. Transferability to Different Tasks

As a potential use case of our flows, we now evaluate the effectiveness of a flow trained on weights from a dataset (e.g. MNIST) different than the target dataset (e.g. Fashion-MNIST) using the same architecture, although the GNNs we model our flow with can be applied to architectures other than the one they were trained on. There are three main ways we can evaluate this performance:

1. Evaluate the samples directly.
2. Guide sampling with gradients from the target task.
3. Initialize model with sampled weights and train on the target dataset.

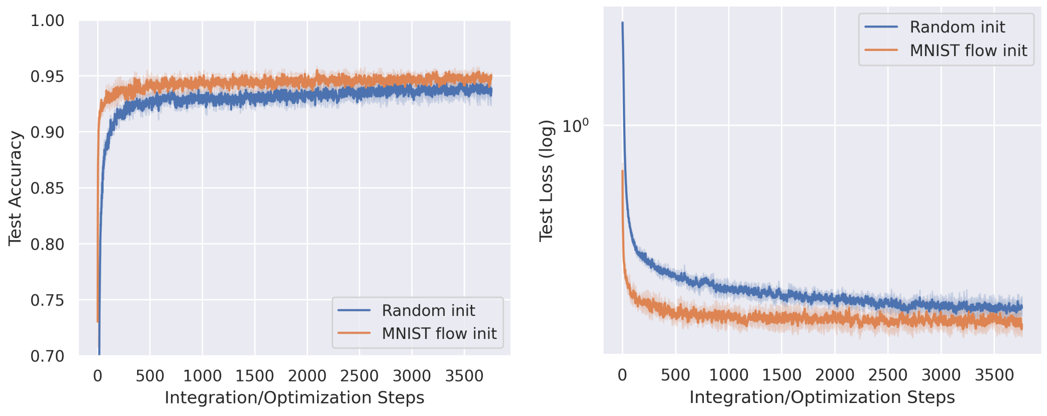
First, Table 7.3 shows the performance (on Fashion-MNIST) of samples from flows trained with MNIST weights but sampled with guidance from the Fashion-MNIST task gradients over 512 Euler steps. First, expectedly, the last row of the table shows that samples obtained directly from the MNIST flows without guidance achieve an average accuracy of 8%, which is not better than random guessing. However, although the accuracy of the base model after optimization with Adam is slightly lower for Fashion-MNIST compared to MNIST, samples from all three flows achieve a test accuracy of at least 72%, which is similar performance of the samples obtained without guidance from the same flows on MNIST.

These results highlight that even though the flows are trained on weights obtained over one dataset, they can capture some salient features shared across optimization landscapes, and can be adapted to different datasets without having to train them from scratch, in this case only by guiding the sampling process with task gradients which is significantly cheaper than training the entire flow from scratch.

Next, we evaluate if the samples from the MNIST flows obtained without any guidance provide a more effective initialization scheme then random initialization for a model trained on Fashion-MNIST. Figure 7.6 displays the test accuracy and loss values during training when the model is initialized from an isotropic Gaussian or with our flow trained on MNIST weights, averaged over five training runs. Although the curves tend to converge towards similar values, the model initialized with the weights from the flow converges faster. This indicates that using our flows for learned initialization on similar tasks can lead to faster convergence and thus more efficient training.

7.3.4. Transferability to Different Architectures

Since a GNN is by design not limited to a certain graph structure, our weight-space flows can also be applied to architectures different than the one they were trained on.



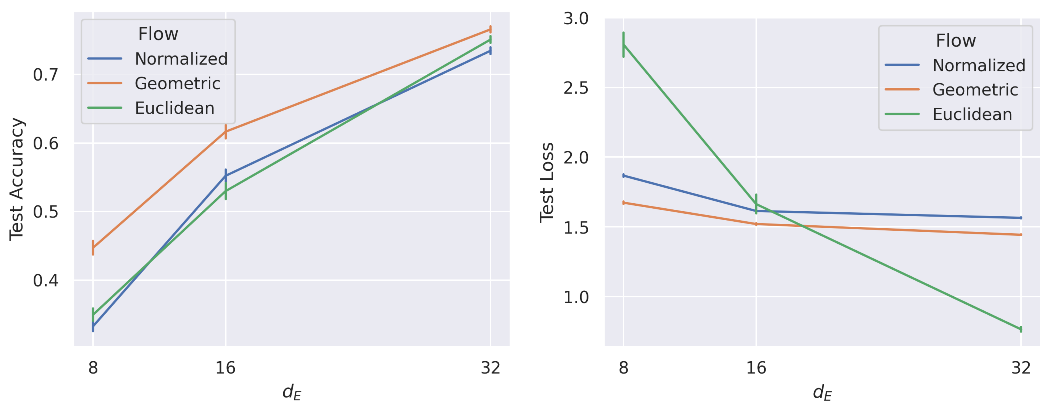
(a) Test Accuracy (b) Test Loss

Figure 7.6.: **Optimization trajectories of Adam on Fashion-MNIST with initial weights sampled from an isotropic Gaussian and the Euclidean flow trained on MNIST weights.** While the curves tend to converge towards similar values, the learned initialization converges faster.

Flow	Accuracy	Loss
Euclidean	0.826 ± 0.076	0.830 ± 0.281
Euclidean (w/ guidance)	0.842 ± 0.079	0.796 ± 0.337
Geometric	0.890 ± 0.030	1.030 ± 0.038
Geometric (w/ guidance)	0.886 ± 0.033	1.031 ± 0.037
Normalized	0.203 ± 0.122	2.652 ± 0.667
Normalized (w/ guidance)	0.184 ± 0.083	2.776 ± 0.575

Table 7.4.: **Generalization of flows trained on MNIST MLPs to MLPs with different architectures.** Euclidean and Geometric flows can generalize to an architecture with a wider hidden layer without requiring guidance while the Normalized flow fails to do so even with guidance.

To evaluate this, we use the flow models from the previous chapters trained on the weights of an MLP with 10 hidden neurons and evaluate them using a larger MLP with 32 hidden neurons. Table 7.4 shows the resulting test accuracies and losses on MNIST. In particular the Euclidean and Geometric flows output samples with even higher accuracy than they did for the smaller MLP. This is not surprising as the larger model is expected to have better performance than the smaller model, but shows that



(a) Test Accuracy (b) Test Loss

Figure 7.7.: **Sample predictive performance with increasing model size**, characterized by the node/edge feature dimension d_E. The upward trajectory in Figure 7.7a and the downwards trajectory in Figure 7.7b implies that there are still performance gains to be achieved by using larger models.

the flows learn to approximate the posterior for not just one architecture, but for the underlying task more generally.

7.3.5. Model Size Scaling

We conclude our evaluations by measuring if there is performance gains to be expected by using larger flow models than the ones we have considered so far. We thus train flow models with different node/edge feature dimensions (d_E) for the same number of iterations, and evaluate their sample qualities. The largest model, with 32 dimensional features, has approximately 146K parameters while the smallest model with 8 dimensional features has approximately 15K parameters.

Figure 7.7 displays the quality of the generated samples for flow models with 8, 16, and 32 dimensional features. While the Geometric flow achieves the most accurate samples for each model size, the upwards trajectories for the test accuracies and the downwards trajectories for the test losses common to all three kinds of flows indicate that there are performance gains to be expected by using larger models.

Considering that the literature on weight-space generative models often uses models with number of parameters on the order of tens or even hundreds of millions, the largest models in Figure 7.7 are relatively small, and still manage to achieve high sample quality.

8. Conclusion

Deep generative models, applied to domains such as images (Esser et al., 2024) and molecules (Abramson et al., 2024) to great effect, often take into account various symmetries in their data such as translation-invariance in images or rotational symmetries for molecules. Generative models applied to other neural networks' weights on the other hand, have so far only considered the permutation symmetries of neural networks, and not the scaling symmetries arising from the use of non-linear activations such as ReLU. We have thus attempted to address this gap, and showed that utilizing this geometric structure of neural networks in addition to the permutation symmetries helps build more effective generative models of neural network weights.

Using the flow matching framework, we have constructed three different kinds of flows with different ways of handling this geometry, and evaluated them in various use cases such as transfer learning, learned initialization, and Bayesian model averaging. Our results have demonstrated that often the fully geometric flow results in better performance, and that there are still gains to be expected by further scaling up our models.

8.1. Future Directions

While we have demonstrated that generative models in weight-space trained via the flow matching objective can generate high-quality samples and that taking into account the geometry of the neural network weights can make such flows more effective, applying modern generative models to neural network weights is still an active research area with many fruitful potential future directions and we outline a list of potential directions for future work in this section.

8.1.1. More Fine-Grained Geometric Considerations

Our flows so far are built for MLPs and only take into account the permutation symmetries between subsequent layers and scaling symmetries resulting from the use of ReLU activations. However, as outlined in previous work (Kofinas et al., 2024; Lim, Maron, et al., 2023), different architectural choices such as convolutions, residual connections, or transformer blocks induce different kinds of symmetries that can be captured by constructing the neural graphs accordingly. The same GNN architectures, such as the Relational Transformer we have used, can then be used for learning tasks over these graphs.

Different activation functions also induce different symmetries (Godfrey et al., 2022) which can be used to embed neural networks into different manifolds, similar to our embedding of ReLU MLPs on a product of hyperspheres and Euclidean space following (Pittorino et al., 2022). Since generative modeling frameworks such as flow matching can readily be applied to different manifolds (Chen and Lipman, 2023), including the use of data-dependent metrics (Kapusniak et al., 2024) rather than explicitly formulating a manifold, extending our flow models to different kinds of symmetries represents another potentially valuable line of work.

Another future direction related to symmetries in weight-space is to account for data-dependent symmetries (Zhao, Ganev, et al., 2023) that arise from the data the model is trained on rather than the static symmetries in the previous paragraph that are valid for all instantiations of the same architecture. Recent work attempting to find these symmetries in an automated way (Zhao, Dehmamy, et al., 2024) can also be useful building blocks in this light.

Finally, certain functional constraints in neural networks such as group equivariance are often imposed through constraints on the neural networks' weights (Weiler, 2023). If the base model has such constraints, accounting for them while designing the flow as well could reduce the effective dimensionality of the problem considerably and lead to more efficient flows.

8.1.2. Generative Modeling and Training

While each our flows is trained on a single architecture and dataset to push-forward a single source distribution to a single target distribution, a single flow that could potentially learn to map different source distributions (e.g. corresponding to the same base architecture on different datasets, or different architectures on the same dataset) could be a more directly useful method. This would essentially correspond to a setting where the model learns to model a vector field not in weight-space, but in the space of probability distributions. Generalizations of the flow matching framework to this setting, such as Meta Flow Matching (Atanackovic et al., 2024) and Wasserstein Flow Matching (Haviv et al., 2024) could be useful building blocks for such a flow, as well as the publicly available datasets of neural network weights (Schürholt, Taskiran, et al., 2022). This approach could lead to weight-space "foundation models" that are trained over a diverse set of tasks and can be adapted to specific use cases by fine-tuning on a single task. While this would be instance of meta-learning (Hospedales et al., 2022), it would be have the added benefit of obtaining probability distributions over solutions rather than point estimates.

Furthermore, the requirement of sample-based training can also be relaxed by utilizing recent work on generative modeling without samples (Vargas et al., 2023; Akhound-Sadegh et al., 2024). In particular, iDEM (Akhound-Sadegh et al., 2024) learns a sampler with access to the energy function and its gradient (and optionally samples from the posterior) without having to simulate the forward and backward trajectories of the sampling process, making it potentially useful in high-dimensional spaces such as neural network weights.

8.1.3. Sampling and Guidance

We sample from our flows using an Euler ODE solver and perform guidance during sampling only using the base task gradients, but the sampling phase has a richer design space that can be utilized to obtain faster and more controllable flows. To begin with, higher-order ODE solvers can be used to reduce the approximation error to the learned vector field at the cost of longer sampling times. Distillation methods such as Flow Map Matching (Boffi et al., 2024) and Consistency Flow Matching (L. Yang et al., 2024) can be utilized to instead speed up the sampling process.

Using a generative modeling framework such as flow matching also enables guidance methods beyond using task gradients. First, any differentiable objective can be used in place of the task loss to guide the sampling process. Performing classifier-free guidance is also possible for flow models (Zheng et al., 2023), to instead condition the sampling process on a variable such as the desired loss, as also done in weight-space with diffusion models in Peebles et al. (2022). Finally an alternative conditioning approach in flow matching is to differentiate through the ODE sampling process to optimize the initial point x_0 so that the solution x_1 minimizes a loss function (Ben-Hamu et al., 2024). This and future work for controllable sampling in flow models can be adapted neural network weights to condition flows on desired quantities or objectives such as adversarial robustness.

A. Experimental Setups

Implementation Details and Computational Resources

Our flows were primarily implemented in PyTorch (Paszke et al., 2019), building on the weight-space GNN and graph construction implementation of (Kofinas et al., 2024) and with our custom flow matching implementation utilizing parts of the `torchcfm` package of (Tong et al., 2023). We have implemented a custom Euler solver to integrate our ODEs and used the `geoopt` package (Kochurov et al., 2020) for geometric operations.

Throughout the experiments, we train all our models using a combination of NVIDIA A100 an H100 GPUs, with each model being trained on a single GPU. The training times range from around 6 hours for the smaller Euclidean flows to approximately 40 hours for the larger Geometric flows, although our implementation was not fully optimized for efficiency.

Flow Between Gaussians (Section 7.1)

Data

- $p_0 = \mathcal{N}(0, \mathbf{I}), p_1 = \mathcal{N}(1, \mathbf{I})$.
- The model is an MLP with dimensions 30 - 16 - 16 - 2 and ReLU activations.

Flow Architecture

- Relational Transformer (Diao and Loynd, 2023; Kofinas et al., 2024) with 5 layers and $d_E = 32$, GeLU activations (Hendrycks and Gimpel, 2023), one attention head.

Flow/Training

- Independent coupling, $t \sim \text{Beta}(1, 2)$, Gaussian probability path with $\sigma = 0.001$, x_1 prediction.

- Trained for 20,000 iterations with batch size 16, using the Adam optimizer with initial learning rate 0.001.

UCI Classification (Section 7.2)

Data

- $p_0 = \mathcal{N}(0, 0.1\mathbf{I})$. To obtain samples from p_1, we sample 100 models from p_0 and train each independently for 50 epochs with Adam. We ignore the first 10 epochs, and record one sample every four iterations after that.
- We use the same MLP architecture as the Gaussian experiments.

Flow Architecture

- Relational Transformer (Diao and Loynd, 2023; Kofinas et al., 2024) with 5 layers and $d_E = 64$, GeLU activations (Hendrycks and Gimpel, 2023), one attention head.

Flow/Training

- $t \sim \text{Beta}(1, 2)$, Gaussian probability path with $\sigma = 0.0001$, x_1 prediction.
- Trained for 250,000 iterations with batch size 32, using the Adam optimizer with initial learning rate 0.001.

MNIST Classification (Section 7.3)

Data

- $p_0 = \mathcal{N}(0, 0.1\mathbf{I})$. For the target samples, we independently train 50 models for 25 epochs using SGD with momentum 0.9, learning rate 0.1, and weight decay 0.00001. We collect one sample every 10 iterations, discarding the first 5 epochs.
- The base model is an MLP with 784 input and 10 output dimensions, and one hidden layer of 10 units.

Flow Architecture

- Relational Transformer (Diao and Loynd, 2023; Kofinas et al., 2024) with 5 layers and $d_E = 32$, GeLU activations (Hendrycks and Gimpel, 2023), one attention head.

Flow/Training

- $t \sim \text{Beta}(1,2)$, Gaussian probability path with $\sigma = 0.00001$, x_1 prediction.
- Trained for $100,000$ iterations with batch size 8.

List of Figures

List of Tables

Bibliography

Abramson, Josh, Jonas Adler, Jack Dunger, Richard Evans, Tim Green, Alexander Pritzel, Olaf Ronneberger, Lindsay Willmore, Andrew J. Ballard, Joshua Bambrick, Sebastian W. Bodenstein, David A. Evans, Chia-Chun Hung, Michael ONeill, David Reiman, Kathryn Tunyasuvunakool, Zachary Wu, Akvil emgulyt, Eirini Arvaniti, Charles Beattie, Ottavia Bertolli, Alex Bridgland, Alexey Cherepanov, Miles Congreve, Alexander I. Cowen-Rivers, Andrew Cowie, Michael Figurnov, Fabian B. Fuchs, Hannah Gladman, Rishub Jain, Yousuf A. Khan, Caroline M. R. Low, Kuba Perlin, Anna Potapenko, Pascal Savy, Sukhdeep Singh, Adrian Stecula, Ashok Thillaisundaram, Catherine Tong, Sergei Yakneen, Ellen D. Zhong, Michal Zielinski, Augustin ídek, Victor Bapst, Pushmeet Kohli, Max Jaderberg, Demis Hassabis, and John M. Jumper (2024). "Accurate Structure Prediction of Biomolecular Interactions with AlphaFold 3." In: *Nature* 630.8016, pp. 493–500. ISSN: 1476-4687. DOI: 10.1038/s41586-024-07487-w (cit. on pp. 2, 45).

Ainsworth, Samuel K., Jonathan Hayase, and Siddhartha Srinivasa (2023). *Git Re-Basin: Merging Models modulo Permutation Symmetries*. DOI: 10.48550/arXiv.2209.04836. arXiv: 2209.04836 [cs]. URL: http://arxiv.org/abs/2209.04836. Pre-published (cit. on pp. 5, 22, 23).

Akhound-Sadegh, Tara, Jarrid Rector-Brooks, Avishek Joey Bose, Sarthak Mittal, Pablo Lemos, Cheng-Hao Liu, Marcin Sendera, Siamak Ravanbakhsh, Gauthier Gidel, Yoshua Bengio, Nikolay Malkin, and Alexander Tong (2024). *Iterated Denoising Energy Matching for Sampling from Boltzmann Densities*. DOI: 10.48550/arXiv.2402.06121. arXiv: 2402.06121 [cs, stat]. URL: http://arxiv.org/abs/2402.06121. Pre-published (cit. on pp. 29, 47).

Albergo, Michael S., Nicholas M. Boffi, and Eric Vanden-Eijnden (2023). *Stochastic Interpolants: A Unifying Framework for Flows and Diffusions*. DOI: 10.48550/arXiv.2303.08797. arXiv: 2303.08797 [cond-mat]. URL: http://arxiv.org/abs/2303.08797. Pre-published (cit. on pp. 5, 11).

Arbel, Julyan, Konstantinos Pitas, Mariia Vladimirova, and Vincent Fortuin (2023). *A Primer on Bayesian Neural Networks: Review and Debates*. DOI: 10.48550/arXiv.2309.16314. arXiv: 2309.16314 [cs, math, stat]. URL: http://arxiv.org/abs/2309.16314. Pre-published (cit. on pp. 7, 8).

Atanackovic, Lazar, Xi Zhang, Brandon Amos, Mathieu Blanchette, Leo J. Lee, Yoshua Bengio, Alexander Tong, and Kirill Neklyudov (2024). *Meta Flow Matching: Integrating Vector Fields on the Wasserstein Manifold*. DOI: 10.48550/arXiv.2408.14608. arXiv: 2408.14608 [cs, stat]. URL: http://arxiv.org/abs/2408.14608. Pre-published (cit. on p. 46).

Barbu, Adrian G. and Song-Chun Zhu (2020). *Monte Carlo Methods*. Singapore: Springer. 422 pp. ISBN: 978-981-13-2971-5 978-981-13-2970-8. DOI: 10.1007/978-981-13-2971-5 (cit. on p. 9).

Baydin, Atlm Gunes, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind (2018). "Automatic Dierentiation in Machine Learning: A Survey." In: *Journal of Machine Learning Research* (cit. on p. 18).

Ben-Hamu, Heli, Omri Puny, Itai Gat, Brian Karrer, Uriel Singer, and Yaron Lipman (2024). *D-Flow: Differentiating through Flows for Controlled Generation*. DOI: 10.48550/arXiv.2402.14017. arXiv: 2402.14017 [cs]. URL: http://arxiv.org/abs/2402.14017. Pre-published (cit. on p. 47).

Blundell, Charles, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra (2015). "Weight Uncertainty in Neural Network." In: *Proceedings of the 32nd International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, pp. 1613–1622 (cit. on p. 9).

Bodnar, Cristian, Wessel P. Bruinsma, Ana Lucic, Megan Stanley, Johannes Brandstetter, Patrick Garvan, Maik Riechert, Jonathan Weyn, Haiyu Dong, Anna Vaughan, Jayesh K. Gupta, Kit Tambiratnam, Alex Archibald, Elizabeth Heider, Max Welling, Richard E. Turner, and Paris Perdikaris (2024). *Aurora: A Foundation Model of the Atmosphere*. DOI: 10.48550/arXiv.2405.13063. arXiv: 2405.13063 [physics]. URL: http://arxiv.org/abs/2405.13063. Pre-published (cit. on p. 3).

Boffi, Nicholas M., Michael S. Albergo, and Eric Vanden-Eijnden (2024). *Flow Map Matching*. DOI: 10.48550/arXiv.2406.07507. arXiv: 2406.07507 [cs, math]. URL: http://arxiv.org/abs/2406.07507. Pre-published (cit. on p. 47).

Bose, Avishek Joey, Tara Akhound-Sadegh, Guillaume Huguet, Kilian Fatras, Jarrid Rector-Brooks, Cheng-Hao Liu, Andrei Cristian Nica, Maksym Korablyov, Michael Bronstein, and Alexander Tong (2024). *SE(3)-Stochastic Flow Matching for Protein Backbone Generation*. DOI: 10.48550/arXiv.2310.02391. arXiv: 2310.02391 [cs]. URL: http://arxiv.org/abs/2310.02391. Pre-published (cit. on p. 3).

Brea, Johanni, Berfin Simsek, Bernd Illing, and Wulfram Gerstner (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]. URL: http://arxiv.org/abs/1907.02911. Pre-published (cit. on p. 19).

Brehmer, Johann, Sönke Behrends, Pim de Haan, and Taco Cohen (2024). *Does Equivariance Matter at Scale?* DOI: 10.48550/arXiv.2410.23179. arXiv: 2410.23179. URL: http://arxiv.org/abs/2410.23179. Pre-published (cit. on pp. 2, 19).

Brockschmidt, Marc (2020). "GNN-FiLM: Graph Neural Networks with Feature-wise Linear Modulation." In: *Proceedings of the 37th International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, pp. 1144–1152 (cit. on p. 26).

Bronstein, Michael M., Joan Bruna, Taco Cohen, and Petar Velikovi (2021). *Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges*. DOI: 10.48550/arXiv.2104.13478. arXiv: 2104.13478 [cs, stat]. URL: http://arxiv.org/abs/2104.13478. Pre-published (cit. on p. 19).

Chen, Ricky T. Q. and Yaron Lipman (2023). *Riemannian Flow Matching on General Geometries*. DOI: 10.48550/arXiv.2302.03660. arXiv: 2302.03660 [cs, stat]. URL: http://arxiv.org/abs/2302.03660. Pre-published (cit. on pp. 3, 5, 15, 16, 31, 46).

Chen, Ricky T. Q., Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud (2018). "Neural Ordinary Differential Equations." In: *Advances in Neural Information Processing Systems*. Vol. 31. Curran Associates, Inc. (cit. on pp. 10, 12, 17).

Cohen, Taco and Max Welling (2016). "Group Equivariant Convolutional Networks." In: *Proceedings of The 33rd International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, pp. 2990–2999 (cit. on p. 8).

Corso, Gabriele, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Velikovi (2020). "Principal Neighbourhood Aggregation for Graph Nets." In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., pp. 13260–13271 (cit. on p. 26).

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 (cit. on p. 22).

Daxberger, Erik, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig (2021). "Laplace Redux - Effortless Bayesian Deep Learning." In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., pp. 20089–20103 (cit. on p. 8).

Diao, Cameron and Ricky Loynd (2023). *Relational Attention: Generalizing Transformers for Graph-Structured Tasks*. DOI: 10.48550/arXiv.2210.05062. arXiv: 2210.05062 [cs]. URL: http://arxiv.org/abs/2210.05062. Pre-published (cit. on pp. 4, 25, 26, 32, 48–50).

Dinh, Laurent, Jascha Sohl-Dickstein, and Samy Bengio (2017). "Density Estimation Using Real NVP." In: International Conference on Learning Representations (cit. on p. 17).

Endres, D.M. and J.E. Schindelin (2003). "A New Metric for Probability Distributions." In: *IEEE Transactions on Information Theory* 49.7, pp. 1858–1860. ISSN: 1557-9654. DOI: 10.1109/TIT.2003.813506 (cit. on p. 38).

Entezari, Rahim, Hanie Sedghi, Olga Saukh, and Behnam Neyshabur (2022). *The Role of Permutation Invariance in Linear Mode Connectivity of Neural Networks*. DOI: 10.48550/arXiv.2110.06296. arXiv: 2110.06296 [cs]. URL: http://arxiv.org/abs/2110.06296. Pre-published (cit. on pp. 3, 21).

Erkoç, Ziya, Fangchang Ma, Qi Shan, Matthias NieSSner, and Angela Dai (2023). "HyperDiffusion: Generating Implicit Neural Fields with Weight-Space Diffusion." In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 14300–14310 (cit. on p. 24).

Esser, Patrick, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, and Robin Rombach (2024). "Scaling Rectified Flow Transformers for High-Resolution Image Synthesis." In: Forty-First International Conference on Machine Learning (cit. on pp. 2, 45).

Fatras, Kilian, Younes Zine, Szymon Majewski, Rémi Flamary, Rémi Gribonval, and Nicolas Courty (2021). *Minibatch Optimal Transport Distances; Analysis and Applications*. arXiv: 2101.01792 [cs, stat]. URL: http://arxiv.org/abs/2101.01792. Pre-published (cit. on pp. 14, 32).

Felardos, Loris, Jérôme Hénin, and Guillaume Charpiat (2023). *Designing Losses for Data-Free Training of Normalizing Flows on Boltzmann Distributions*. DOI: 10.48550/arXiv.2301.05475. arXiv: 2301.05475 [cond-mat]. URL: http://arxiv.org/abs/2301.05475. Pre-published (cit. on p. 9).

Ferbach, Damien, Baptiste Goujaud, Gauthier Gidel, and Aymeric Dieuleveut (2024). "Proving Linear Mode Connectivity of Neural Networks via Optimal Transport." In: *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*. International Conference on Artificial Intelligence and Statistics. PMLR, pp. 3853–3861 (cit. on p. 21).

Finn, Chelsea, Pieter Abbeel, and Sergey Levine (2017). "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks." In: *Proceedings of the 34th International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, pp. 1126–1135 (cit. on p. 2).

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 (cit. on p. 8).

Garipov, Timur, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson (2018). "Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs."

In: *Advances in Neural Information Processing Systems*. Vol. 31. Curran Associates, Inc. (cit. on p. 21).

Gelberg, Yoav, Tycho F. A. van der Ouderaa, Mark van der Wilk, and Yarin Gal (2024). "Variational Inference Failures Under Model Symmetries: Permutation Invariant Posteriors for Bayesian Neural Networks." In: ICML 2024 Workshop on Geometry-grounded Representation Learning and Generative Modeling (cit. on p. 9).

Gemici, Mevlana C., Danilo Rezende, and Shakir Mohamed (2016). *Normalizing Flows on Riemannian Manifolds*. DOI: 10.48550/arXiv.1611.02304. arXiv: 1611.02304 [cs, math, stat]. URL: http://arxiv.org/abs/1611.02304. Pre-published (cit. on p. 14).

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] (cit. on p. 7).

Godfrey, Charles, Davis Brown, Tegan Emerson, and Henry Kvinge (2022). "On the Symmetries of Deep Learning Models and Their Internal Representations." In: *Advances in Neural Information Processing Systems* 35, pp. 11893–11905 (cit. on pp. 4, 20, 31, 46).

Grathwohl, Will, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud (2018). *FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models*. DOI: 10.48550/arXiv.1810.01367. arXiv: 1810.01367. URL: http://arxiv.org/abs/1810.01367. Pre-published (cit. on p. 17).

Gutteridge, Benjamin, Xiaowen Dong, Michael Bronstein, and Francesco Di Giovanni (2023). *DRew: Dynamically Rewired Message Passing with Delay*. DOI: 10.48550/arXiv.2305.08018. arXiv: 2305.08018 [cs, stat]. URL: http://arxiv.org/abs/2305.08018. Pre-published (cit. on p. 25).

Ha, David, Andrew Dai, and Quoc V. Le (2016). *HyperNetworks*. DOI: 10.48550/arXiv.1609.09106. arXiv: 1609.09106 [cs]. URL: http://arxiv.org/abs/1609.09106. Pre-published (cit. on p. 24).

Hastings, W. K. (1970). "Monte Carlo Sampling Methods Using Markov Chains and Their Applications." In: *Biometrika* 57.1, pp. 97–109. ISSN: 0006-3444 (cit. on p. 10).

Haviv, Doron, Aram-Alexandre Pooladian, Dana Pe'er, and Brandon Amos (2024). *Wasserstein Flow Matching: Generative Modeling over Families of Distributions*. DOI: 10.48550/arXiv.2411.00698. arXiv: 2411.00698. URL: http://arxiv.org/abs/2411.00698. Pre-published (cit. on p. 46).

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 (cit. on pp. 4, 19).

Hendrycks, Dan and Kevin Gimpel (2023). *Gaussian Error Linear Units (GELUs)*. DOI: 10.48550/arXiv.1606.08415. arXiv: 1606.08415 [cs]. URL: http://arxiv.org/abs/1606.08415. Pre-published (cit. on pp. 48–50).

Ho, Jonathan, Ajay Jain, and Pieter Abbeel (2020). "Denoising Diffusion Probabilistic Models." In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., pp. 6840–6851 (cit. on p. 12).

Hospedales, Timothy, Antreas Antoniou, Paul Micaelli, and Amos Storkey (2022). "Meta-Learning in Neural Networks: A Survey." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.9, pp. 5149–5169. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2021.3079209 (cit. on pp. 2, 46).

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 (cit. on p. 17).

John M. Lee (2018). *Introduction to Riemannian Manifolds*. Vol. 176. Graduate Texts in Mathematics. Cham: Springer International Publishing. ISBN: 978-3-319-91754-2 978-3-319-91755-9. DOI: 10.1007/978-3-319-91755-9 (cit. on p. 14).

Kahn, H. and A. W. Marshall (1953). "Methods of Reducing Sample Size in Monte Carlo Computations." In: *Journal of the Operations Research Society of America* 1.5, pp. 263–278. ISSN: 0096-3984. JSTOR: 166789 (cit. on p. 9).

Kalogeropoulos, Ioannis, Giorgos Bouritsas, and Yannis Panagakis (2024). *Scale Equivariant Graph Metanetworks*. arXiv: 2406.10685 [cs]. URL: http://arxiv.org/abs/2406.10685. Pre-published (cit. on pp. 4, 25).

Kantorovitch, L. (1958). "On the Translocation of Masses." In: *Management Science* 5.1, pp. 1–4. JSTOR: 2626967 (cit. on p. 10).

Kapusniak, Kacper, Peter Potaptchik, Teodora Reu, Leo Zhang, Alexander Tong, Michael Bronstein, Avishek Joey Bose, and Francesco Di Giovanni (2024). *Metric Flow Matching for Smooth Interpolations on the Data Manifold*. DOI: 10.48550/arXiv.2405.14780. arXiv: 2405.14780 [cs, stat]. URL: http://arxiv.org/abs/2405.14780. Pre-published (cit. on pp. 3, 46).

Karras, Tero, Timo Aila, Miika Aittala, and Samuli Laine (2022). "Elucidating the Design Space of Diffusion-Based Generative Models." In: 36th Conference on Neural Information Processing Systems (cit. on p. 33).

Kingma, Diederik P. and Jimmy Ba (2017). "Adam: A Method for Stochastic Optimization." arXiv: 1412.6980 [cs] (cit. on p. 36).

Kingma, Diederik P. and Max Welling (2022). *Auto-Encoding Variational Bayes*. DOI: 10.48550/arXiv.1312.6114. arXiv: 1312.6114 [cs, stat]. URL: http://arxiv.org/abs/1312.6114. Pre-published (cit. on p. 9).

Kipf, Thomas N. and Max Welling (2016). "Semi-Supervised Classification with Graph Convolutional Networks." In: International Conference on Learning Representations (cit. on p. 25).

Klein, Leon, Andreas Krämer, and Frank Noe (2023). "Equivariant Flow Matching." In: *Advances in Neural Information Processing Systems* 36, pp. 59886–59910 (cit. on p. 3).

Kochurov, Max, Rasul Karimov, and Serge Kozlukov (2020). *Geoopt: Riemannian Optimization in PyTorch*. DOI: 10.48550/arXiv.2005.02819. arXiv: 2005.02819 [cs]. URL: http://arxiv.org/abs/2005.02819. Pre-published (cit. on p. 48).

Kofinas, Miltiadis, Boris Knyazev, Yan Zhang, Yunlu Chen, Gertjan J. Burghouts, Efstratios Gavves, Cees G. M. Snoek, and David W. Zhang (2024). *Graph Neural Networks for Learning Equivariant Representations of Neural Networks*. DOI: 10.48550/arXiv.2403.12143. arXiv: 2403.12143 [cs, stat]. URL: http://arxiv.org/abs/2403.12143. Pre-published (cit. on pp. 2, 4, 24–26, 32, 45, 48–50).

Krueger, David, Chin-Wei Huang, Riashat Islam, Ryan Turner, Alexandre Lacoste, and Aaron Courville (2018). *Bayesian Hypernetworks*. DOI: 10.48550/arXiv.1710.04759. arXiv: 1710.04759 [stat]. URL: http://arxiv.org/abs/1710.04759. Pre-published (cit. on pp. 24, 27).

Kulyt, Paulina, Francisco Vargas, Simon Valentin Mathis, Yu Guang Wang, José Miguel Hernández-Lobato, and Pietro Liò (2024). *Improving Antibody Design with Force-Guided Sampling in Diffusion Models*. DOI: 10.48550/arXiv.2406.05832. arXiv: 2406.05832. URL: http://arxiv.org/abs/2406.05832. Pre-published (cit. on p. 33).

Lim, Derek, Haggai Maron, Marc T. Law, Jonathan Lorraine, and James Lucas (2023). *Graph Metanetworks for Processing Diverse Neural Architectures*. DOI: 10.48550/arXiv.2312.04501. arXiv: 2312.04501 [cs, stat]. URL: http://arxiv.org/abs/2312.04501. Pre-published (cit. on pp. 2, 4, 20, 24, 25, 45).

Lim, Derek, Moe Putterman, Robin Walters, Haggai Maron, and Stefanie Jegelka (2024). *The Empirical Impact of Neural Parameter Symmetries, or Lack Thereof*. DOI: 10.48550/arXiv.2405.20231. arXiv: 2405.20231 [cs, stat]. URL: http://arxiv.org/abs/2405.20231. Pre-published (cit. on p. 19).

Lipman, Yaron, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le (2023). *Flow Matching for Generative Modeling*. DOI: 10.48550/arXiv.2210.02747. arXiv: 2210.02747 [cs, stat]. URL: http://arxiv.org/abs/2210.02747. Pre-published (cit. on pp. 5, 11, 12, 18).

Lipman, Yaron, Marton Havasi, Peter Holderrieth, Neta Shaul, Matt Le, Brian Karrer, Ricky T. Q. Chen, David Lopez-Paz, Heli Ben-Hamu, and Itai Gat (2024). *Flow Matching Guide and Code*. DOI: 10.48550/arXiv.2412.06264. arXiv: 2412.06264 [cs]. URL: http://arxiv.org/abs/2412.06264. Pre-published (cit. on pp. 2, 11).

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

arXiv: 2209.03003. URL: http://arxiv.org/abs/2209.03003. Pre-published (cit. on pp. 5, 11).

Lou, Aaron, Derek Lim, Isay Katsman, Leo Huang, Qingxuan Jiang, Ser-Nam Lim, and Christopher De Sa (2020). *Neural Manifold Ordinary Differential Equations*. DOI: 10.48550/arXiv.2006.10254. arXiv: 2006.10254 [stat]. URL: http://arxiv.org/abs/2006.10254. Pre-published (cit. on p. 14).

Ma, Yi-An, Tianqi Chen, and Emily Fox (2015). "A Complete Recipe for Stochastic Gradient MCMC." In: *Advances in Neural Information Processing Systems*. Vol. 28. Curran Associates, Inc. (cit. on p. 9).

MacKay, David J C (1992a). "Bayesian Methods for Adaptive Models." In: (cit. on p. 7).

– (1992b). "A Practical Bayesian Framework for Backpropagation Networks." In: *Neural Computation* 4.3, pp. 448–472. ISSN: 0899-7667. DOI: 10.1162/neco.1992.4.3.448 (cit. on p. 8).

Marzouk, Youssef, Tarek Moselhy, Matthew Parno, and Alessio Spantini (2016). "Sampling via Measure Transport: An Introduction." In: *Handbook of Uncertainty Quantification*. Ed. by Roger Ghanem, David Higdon, and Houman Owhadi. Cham: Springer International Publishing, pp. 1–41. ISBN: 978-3-319-11259-6. DOI: 10.1007/978-3-319-11259-6_23-1 (cit. on pp. 9, 10).

Mathieu, Emile and Maximilian Nickel (2020). *Riemannian Continuous Normalizing Flows*. DOI: 10.48550/arXiv.2006.10605. arXiv: 2006.10605 [stat]. URL: http://arxiv.org/abs/2006.10605. Pre-published (cit. on p. 14).

Metropolis, Nicholas, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller (1953). "Equation of State Calculations by Fast Computing Machines." In: *The Journal of Chemical Physics* 21.6, pp. 1087–1092. ISSN: 0021-9606. DOI: 10.1063/1.1699114 (cit. on p. 10).

Mishtal, Aaron and Itamar Arel (2012). "Jensen-Shannon Divergence in Ensembles of Concurrently-Trained Neural Networks." In: *2012 11th International Conference on Machine Learning and Applications*. 2012 11th International Conference on Machine Learning and Applications. Vol. 2, pp. 558–562. DOI: 10.1109/ICMLA.2012.198 (cit. on p. 38).

Monge, G. (1781). "Memoire Sur La Theorie Des Deblais et Des Remblais." In: *Mem. Math. Phys. Acad. Royale Sci.*, pp. 666–704 (cit. on p. 10).

Murphy, Kevin P. (2023). *Probabilistic Machine Learning #2: Advanced Topics*. Adaptive Computation and Machine Learning Series. Cambridge, Massachusetts: The MIT Press. 1 p. ISBN: 978-0-262-37599-3 978-0-262-37600-6 (cit. on p. 8).

Neal, Radford M. (1996). *Bayesian Learning for Neural Networks*. URL: https://link.springer.com/book/10.1007/978-1-4612-0745-0 (cit. on p. 7).

– (2011). *MCMC Using Hamiltonian Dynamics*. DOI: 10.1201/b10905. arXiv: 1206.1901 [physics, stat] (cit. on p. 9).

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. DOI: 10.48550/arXiv.1912.01703. arXiv: 1912.01703 [cs]. URL: http://arxiv.org/abs/1912.01703. Pre-published (cit. on p. 48).

Peebles, William, Ilija Radosavovic, Tim Brooks, Alexei A. Efros, and Jitendra Malik (2022). *Learning to Learn with Generative Models of Neural Network Checkpoints*. DOI: 10.48550/arXiv.2209.12892. arXiv: 2209.12892 [cs]. URL: http://arxiv.org/abs/2209.12892. Pre-published (cit. on pp. 2, 24, 27, 29, 37, 47).

Peña, Fidel A. Guerrero, Heitor Rapela Medeiros, Thomas Dubail, Masih Aminbeidokhti, Eric Granger, and Marco Pedersoli (2023). "Re-Basin via Implicit Sinkhorn Differentiation." In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 20237–20246 (cit. on pp. 5, 22, 23).

Peyré, Gabriel and Marco Cuturi (2020). *Computational Optimal Transport*. arXiv: 1803.00567 [stat]. URL: http://arxiv.org/abs/1803.00567. Pre-published (cit. on p. 10).

Pittorino, Fabrizio, Antonio Ferraro, Gabriele Perugini, Christoph Feinauer, Carlo Baldassi, and Riccardo Zecchina (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*. International Conference on Machine Learning. PMLR, pp. 17759–17781 (cit. on pp. 4, 5, 23, 29, 30, 46).

Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever (2019). "Language Models Are Unsupervised Multitask Learners." In: (cit. on p. 27).

Rangel, Jose Miguel Lara, Stefan Schoepf, Jack Foster, David Krueger, and Usman Anwar (2024). *Learning to Forget Using Hypernetworks*. DOI: 10.48550/arXiv.2412.00761. arXiv: 2412.00761. URL: http://arxiv.org/abs/2412.00761. Pre-published (cit. on p. 24).

Rezende, Danilo and Shakir Mohamed (2015). "Variational Inference with Normalizing Flows." In: *Proceedings of the 32nd International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, pp. 1530–1538 (cit. on pp. 9, 10, 16).

Roberts, Gareth O. and Richard L. Tweedie (1996). "Exponential Convergence of Langevin Distributions and Their Discrete Approximations." In: *Bernoulli* 2.4, pp. 341–363. ISSN: 1350-7265 (cit. on p. 10).

Rossi, Simone, Ankit Singh, and Thomas Hannagan (2023). *On Permutation Symmetries in Bayesian Neural Network Posteriors: A Variational Perspective*. DOI: 10.48550/arXiv.

2310.10171. arXiv: 2310.10171 [cs, stat]. URL: http://arxiv.org/abs/2310.10171. Pre-published (cit. on p. 21).

Santambrogio, Filippo (2015). *Optimal Transport for Applied Mathematicians: Calculus of Variations, PDEs, and Modeling*. Vol. 87. Progress in Nonlinear Differential Equations and Their Applications. Cham: Springer International Publishing. ISBN: 978-3-319-20827-5 978-3-319-20828-2. DOI: 10.1007/978-3-319-20828-2 (cit. on p. 10).

Schürholt, Konstantin (2024). *Hyper-Representations: Learning from Populations of Neural Networks*. DOI: 10.48550/arXiv.2410.05107. arXiv: 2410.05107. URL: http://arxiv.org/abs/2410.05107. Pre-published (cit. on p. 2).

Schürholt, Konstantin, Michael W. Mahoney, and Damian Borth (2024). *Towards Scalable and Versatile Weight Space Learning*. arXiv: 2406.09997 [cs]. URL: http://arxiv.org/abs/2406.09997. Pre-published (cit. on p. 27).

Schürholt, Konstantin, Diyar Taskiran, Boris Knyazev, Xavier Giró-i-Nieto, and Damian Borth (2022). "Model Zoos: A Dataset of Diverse Populations of Neural Network Models." In: *Advances in Neural Information Processing Systems* 35, pp. 38134–38148 (cit. on pp. 2, 29, 46).

Shamsian, Aviv, Aviv Navon, David W. Zhang, Yan Zhang, Ethan Fetaya, Gal Chechik, and Haggai Maron (2024). *Improved Generalization of Weight Space Networks via Augmentations*. DOI: 10.48550/arXiv.2402.04081. arXiv: 2402.04081 [cs]. URL: http://arxiv.org/abs/2402.04081. Pre-published (cit. on p. 4).

Simsek, Berfin, François Ged, Arthur Jacot, Francesco Spadaro, Clement Hongler, Wulfram Gerstner, and Johanni Brea (2021). "Geometry of the Loss Landscape in Overparameterized Neural Networks: Symmetries and Invariances." In: *Proceedings of the 38th International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, pp. 9722–9732 (cit. on p. 19).

Sohl-Dickstein, Jascha, Eric A Weiss, Niru Maheswaranathan, and Surya Ganguli (2015). "Deep Unsupervised Learning Using Nonequilibrium Thermodynamics." In: (cit. on p. 12).

Sommer, Emanuel, Lisa Wimmer, Theodore Papamarkou, Ludwig Bothmann, Bernd Bischl, and David Rügamer (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]. URL: http://arxiv.org/abs/2402.01484. Pre-published (cit. on p. 22).

Song, Yang, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole (2021). *Score-Based Generative Modeling through Stochastic Differential Equations*. arXiv: 2011.13456 [cs, stat]. URL: http://arxiv.org/abs/2011.13456. Pre-published (cit. on pp. 12, 18).

Stoica, George, Daniel Bolya, Jakob Bjorner, Pratik Ramesh, Taylor Hearn, and Judy Hoffman (2024). *ZipIt! Merging Models from Different Tasks without Training*. DOI: 10.

48550/arXiv.2305.03053. arXiv: 2305.03053 [cs]. URL: http://arxiv.org/abs/2305.03053. Pre-published (cit. on p. 21).

Street, W. Nick, W. H. Wolberg, and O. L. Mangasarian (1993). "Nuclear Feature Extraction for Breast Tumor Diagnosis." In: *Biomedical Image Processing and Biomedical Visualization*. Biomedical Image Processing and Biomedical Visualization. Vol. 1905. SPIE, pp. 861–870. DOI: 10.1117/12.148698 (cit. on p. 35).

Tong, Alexander, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrid Rector-Brooks, Kilian Fatras, Guy Wolf, and Yoshua Bengio (2023). *Improving and Generalizing Flow-Based Generative Models with Minibatch Optimal Transport*. DOI: 10.48550/arXiv.2302.00482. arXiv: 2302.00482 [cs]. URL: http://arxiv.org/abs/2302.00482. Pre-published (cit. on pp. 5, 12–14, 27, 48).

Vargas, Francisco, Shreyas Padhy, Denis Blessing, and Nikolas Nüsken (2023). *Transport Meets Variational Inference: Controlled Monte Carlo Diffusions*. arXiv: 2307.01050 [cs, stat]. URL: http://arxiv.org/abs/2307.01050. Pre-published (cit. on p. 47).

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, ukasz Kaiser, and Illia Polosukhin (2017). "Attention Is All You Need." In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc. (cit. on pp. 25, 26).

Velikovi, Petar, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio (2018). *Graph Attention Networks*. DOI: 10.48550/arXiv.1710.10903. arXiv: 1710.10903 [cs, stat]. URL: http://arxiv.org/abs/1710.10903. Pre-published (cit. on p. 25).

Wang, Yan, Lihao Wang, Yuning Shen, Yiqun Wang, Huizhuo Yuan, Yue Wu, and Quanquan Gu (2024). *Protein Conformation Generation via Force-Guided SE(3) Diffusion Models*. DOI: 10.48550/arXiv.2403.14088. arXiv: 2403.14088 [cs, q-bio]. URL: http://arxiv.org/abs/2403.14088. Pre-published (cit. on p. 33).

Weiler, Maurice (2023). *Equivariant and Coordinate Independent Convolutional Networks* (cit. on pp. 19, 46).

Wu, Lingfei, Peng Cui, Jian Pei, and Liang Zhao, eds. (2022). *Graph Neural Networks: Foundations, Frontiers, and Applications*. Singapore: Springer Nature Singapore. ISBN: 978-981-16-6053-5 978-981-16-6054-2. DOI: 10.1007/978-981-16-6054-2 (cit. on p. 24).

Yang, Ling, Zixiang Zhang, Zhilong Zhang, Xingchao Liu, Minkai Xu, Wentao Zhang, Chenlin Meng, Stefano Ermon, and Bin Cui (2024). *Consistency Flow Matching: Defining Straight Flows with Velocity Consistency*. DOI: 10.48550/arXiv.2407.02398. arXiv: 2407.02398 [cs]. URL: http://arxiv.org/abs/2407.02398. Pre-published (cit. on p. 47).

Yu, Ziyang, Wenbing Huang, and Yang Liu (2024). *Force-Guided Bridge Matching for Full-Atom Time-Coarsened Dynamics of Peptides*. DOI: 10.48550/arXiv.2408.15126.

arXiv: 2408.15126 [physics, q-bio]. URL: http://arxiv.org/abs/2408.15126. Pre-published (cit. on p. 33).

Zhao, Bo, Nima Dehmamy, Robin Walters, and Rose Yu (2023). "Understanding Mode Connectivity via Parameter Space Symmetry." In: UniReps: The First Workshop on Unifying Representations in Neural Models (cit. on p. 21).

– (2024). "Finding Symmetry in Neural Network Parameter Spaces." In: UniReps: 2nd Edition of the Workshop on Unifying Representations in Neural Models (cit. on pp. 4, 46).

Zhao, Bo, Iordan Ganev, Robin Walters, Rose Yu, and Nima Dehmamy (2023). *Symmetries, Flat Minima, and the Conserved Quantities of Gradient Flow*. DOI: 10.48550/arXiv.2210.17216. arXiv: 2210.17216 [cs, math]. URL: http://arxiv.org/abs/2210.17216. Pre-published (cit. on p. 46).

Zhao, Bo, Robert M Gower, Robin Walters, and Rose Yu (2024). "IMPROVING CONVERGENCE AND GENERALIZATION USING PARAMETER SYMMETRIES." In: (cit. on p. 19).

Zheng, Qinqing, Matt Le, Neta Shaul, Yaron Lipman, Aditya Grover, and Ricky T. Q. Chen (2023). *Guided Flows for Generative Modeling and Decision Making*. DOI: 10.48550/arXiv.2311.13443. arXiv: 2311.13443 [cs, stat]. URL: http://arxiv.org/abs/2311.13443. Pre-published (cit. on p. 47).

Zhou, Allan, Chelsea Finn, and James Harrison (2024). *Universal Neural Functionals*. DOI: 10.48550/arXiv.2402.05232. arXiv: 2402.05232 [cs]. URL: http://arxiv.org/abs/2402.05232. Pre-published (cit. on p. 24).

Zhou, Allan, Kaien Yang, Yiding Jiang, Kaylee Burns, Winnie Xu, Samuel Sokota, J. Zico Kolter, and Chelsea Finn (2023). "Neural Functional Transformers." In: *Advances in Neural Information Processing Systems* 36, pp. 77485–77502 (cit. on pp. 24, 25).