ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER'S THESIS
in
Machine Learning for Computer Vision

Alignment of Implicit Neural Representations of 3D Shapes via Permutation Symmetries

Supervisor:
Samuele Salti

Candidate:
Francesco Ballerini

Co-supervisors:
Luca De Luigi
Adriano Cardace
Pierluigi Zama Ramirez
Luigi Di Stefano

ACADEMIC YEAR 2021–2022
Third Session

Research is what I'm doing when I don't know what I'm doing.

Wernher von Braun

Abstract

Multi-Layer Perceptrons (MLPs) have long been known to enjoy permutation symmetries, namely the fact that activations of intermediate layers can be swapped without changing the underlying function. Research works inspired by this property have been recently building up to a conjecture stating that any two Stochastic Gradient Descent (SGD) solutions, up to a permutation, lie in the same low-loss region of the parameter space, i.e. they enjoy Linear Mode Connectivity (LMC). Concurrently, Implicit Neural Representations (INRs) have been emerging as a new unifying paradigm to store and represent various signals of interest (such as images, audio, and 3D surfaces), which has sparked a novel interest in machine learning architectures designed to work on neural networks themselves by processing their weights only. The recently proposed `inr2vec` framework is one such architecture, and has been shown to be able to embed input INRs of 3D shapes into latent codes that can be effectively fed into standard deep learning pipelines. A key requirement for the convergence of `inr2vec` is that input INRs share the same random initialization, which has been pointed out as a potential limitation to its applicability. Hence, the motivation behind our work is to eventually allow `inr2vec` to work with arbitrary in-the-wild INRs, regardless of their initialization. As a first attempt to tackle this complex task, we apply and thoroughly analyze recent combinatorial methods which lay their foundations on the aforementioned conjecture by computing a weight permutation that aims at *aligning* two INRs of the same 3D surface. This approach, although promising in some of its results, is ultimately shown to be unable to move INRs into the same basin of the loss landscape. Finally, we present preliminary results of a novel deep learning methodology devised to achieve LMC by directly learning new weights for one of the two models.

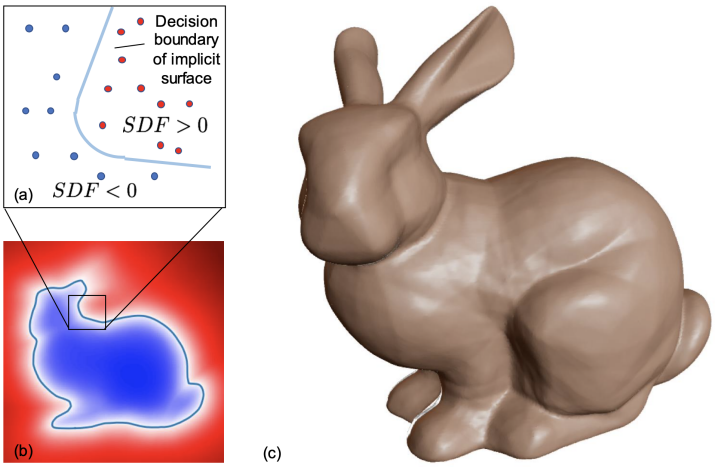
Contents

Chapter 1

Introduction

Since the early days of computer vision, researchers have been processing images as two-dimensional grids of pixels carrying intensity or color measurements. Unfortunately, digital 3D surfaces do not enjoy the same uniformity, leading to a variety of discrete representations—such as voxel grids, point clouds, and meshes—coexisting today, each with its strengths and weaknesses. Voxel grid can be appealing due to their regular structure, while having cubic space complexity; on the opposite side, the unorganized nature of point clouds make them less memory-intensive while also needing more careful processing. Meshes, on the other hand, are also far from straightforward to handle, due to the interplay between their vertex, edge, and face information. Nonetheless, no standard way to store and process 3D surfaces has yet emerged.

INRs For these reasons, a new kind or representation has recently been proposed, called *Implicit Neural Representations* (INRs). INRs leverage Multi-Layer Perceptrons (MLPs) to fit a continuous function that represents implicitly a signal of interest (Xie et al., 2022), and have been shown to be able to effectively encode 3D shapes by fitting *Signed Distance Functions* (SDFs) (Park et al., 2019; Gropp et al., 2020; Takikawa et al., 2021) (Figure 1.1), *Unsigned Distance Functions* (UDFs) (Chibane et al., 2020), and *occupancy fields* (Mescheder et al., 2019; Peng et al., 2020), whose discrete counterparts are meshes, point clouds, and voxel grids, respectively. The biggest advantage of such an approach is that it decouples the memory cost of the representation from its spatial resolution, as a surface

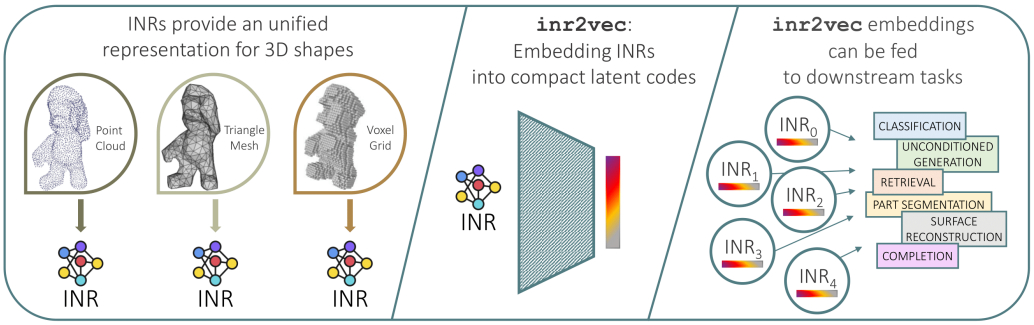


Source: Park et al., 2019

Figure 1.1: **SDF representation of the Stanford Bunny. (a):** Depiction of the underlying implicit surface SDF = 0 trained on sampled points inside (SDF < 0) and outside (SDF > 0) the surface. **(b):** 2D cross-section of the signed distance field. **(c):** Rendered 3D surface recovered from SDF = 0.

with arbitrarily fine resolution can be reconstructed from a fixed number of parameters. Besides, since the same neural network architecture can be used to fit different implicit functions, INRs hold the potential to provide a unified framework to represent 3D shapes, by replacing the complex and/or memory-intensive ad-hoc machinery needed to process 3D discrete representations (Maturana and Scherer, 2015; Qi et al., 2017; Wang et al., 2019; Hu et al., 2022).

`inr2vec` Besides applying INRs to 3D data representation, current research efforts explore whether it is possible to process such INRs directly, in order to solve deep learning downstream tasks like those routinely performed with discrete representations (e.g. doing shape classification on an INR of a 3D surface without reconstructing a discrete representation of the surface itself). Since INRs are neural networks, there is no straightforward way to process them; moreover, a single INR can easily count hundreds of thousands of parameters, most of which are known to be redundant (Section 3.3.2). One possible solution consists in training an individual INR for each shape and then compressing it into a more compact latent vector. This is the approach followed by `inr2vec` (De Luigi et al., 2023), a

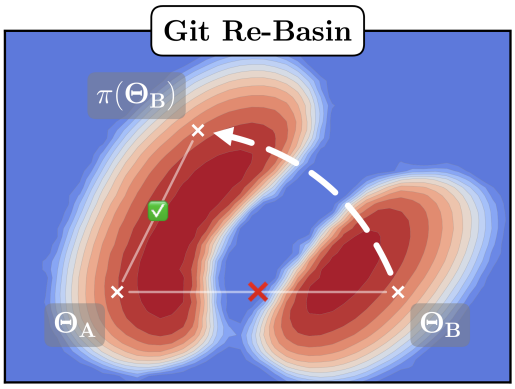


Source: De Luigi et al., 2023

Figure 1.2: **The `inr2vec` framework. Left:** INRs hold the potential to provide a unified representation of 3D shapes. **Center:** `inr2vec` produces a compact representation of an input INR by only looking at its weights. **Right:** `inr2vec` embeddings can be used in combination with standard deep learning machinery to solve a variety of downstream tasks.

representation learning framework based on an encoder designed to produce a task-agnostic embedding representing the input INR by processing its weights only. These embeddings are shown to work in a variety of downstream tasks, such as classification, retrieval, part segmentation, unconditioned generation, surface reconstruction and completion (Figure 1.2).

Shared initialization Since MLPs are *permutation invariant* (Hecht-Nielsen, 1990), i.e. weights and biases can be permuted without changing the network output, there is no guarantee of one-to-one correspondence between the parameters of two INRs trained with different sources of randomness (initialization and data shuffling). Thus, the `inr2vec` encoder would have to deal with input vectors whose elements capture different information across different data samples, making it impossible to train the framework—unless the encoder itself was invariant to weight permutations of input INRs, which, in De Luigi et al.'s formulation, is not the case. Therefore, `inr2vec` requires a shared precomputed random initialization for all input INRs, as this is empirically found to enable the framework convergence. Although this constraint is not unusual (Gropp et al., 2020; Sitzmann et al., 2020a; Dupont et al., 2022), it might conflict with the long-term goal of De Luigi et al.'s work—a future in which INRs have emerged as a standard representation to store



Source: Ainsworth et al., 2022

Figure 1.3: **Entezari et al.'s conjecture.** Given two trained neural networks A and B, the weights and biases of B can be permuted is such a way that B is moved into the same basin of the loss landscape as A.

and communicate 3D shapes, and repositories hosting digital twins of 3D objects encoded by MLPs have become commonly available. Indeed, in such a scenario, it would not be possible to download an arbitrary in-the-wild INR and process it through `inr2vec`, as that INR would need to have been trained starting from the same initialization shared by all the MLPs `inr2vec` was trained on.

Model alignment As a way to address this concern, the goal of our work is to answer the following question: given two MLPs A and B trained starting from different initializations—where $init_A$ is the initialization shared by the INRs `inr2vec` was trained on and $init_B \neq init_A$—what would the weights of B look like if instead $init_A = init_B$? If we knew it, we could *align* B's weights to A's as a processing step and then give B as input to `inr2vec`. Ideally, this procedure should work across shapes, however in this work we will focus on aligning INRs representing the same shape.

Several recent works on model alignment (Tatro et al., 2020; Ainsworth et al., 2022) and fusion (Singh and Jaggi, 2020; Liu et al., 2022) leverage the aforementioned permutation invariance, and empirical observations have led to a conjecture stating that, for a given neural network, any two solutions found by Stochastic Gradient Descent (SGD) lie in the same low-loss region (called *basin*) of the parameter

space, up to a weight permutation (Entezari et al., 2022) (Figure 1.3). Motivated by this line of research as a path towards a solution to the `inr2vec` initialization problem, our work mainly focuses on applying Ainsworth et al.'s methods for model alignment to INRs of 3D shapes like those expected as input by `inr2vec`.

Chapter 2

Theoretical foundations

In this chapter, we delve into the theoretical concepts that justify the notion of model alignment, namely *permutation invariance* of MLPs (Hecht-Nielsen, 1990) and *linear mode connectivity* (Draxler et al., 2018; Garipov et al., 2018; Frankle et al., 2020). Then, we formally derive the two alignment methods proposed by Ainsworth et al. (2022), which will be the focus of our experiments in the next chapter.

2.1 Permutation invariance of MLPs

Consider a 3-layer MLP with parameters $\boldsymbol{\theta}$

$$\begin{aligned}\mathbf{x}_1 &= \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)\\ \mathbf{x}_2 &= \sigma(\mathbf{W}_2\mathbf{x}_1 + \mathbf{b}_2)\\ \mathbf{x}_3 &= \mathbf{W}_3\mathbf{x}_2 + \mathbf{b}_3 = f_{\boldsymbol{\theta}}(\mathbf{x})\end{aligned}$$

or equivalently

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{W}_3\sigma(\mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2) + \mathbf{b}_3$$

where σ is any element-wise nonlinearity, $\mathbf{x}_i \in \mathbb{R}^{d_i \times 1}$ for $i = 0, 1, 2, 3$ (with $\mathbf{x}_0 = \mathbf{x}$), and $\mathbf{W}_i \in \mathbb{R}^{d_i \times d_{i-1}}, \mathbf{b}_i \in \mathbb{R}^{d_i \times 1}$ for $i = 1, 2, 3$. Let $\mathbf{P}_1 \in \mathbb{R}^{d_1 \times d_1}, \mathbf{P}_2 \in \mathbb{R}^{d_2 \times d_2}$ be *permutation matrices*, i.e. orthogonal matrices that act on rows when applied on the left and on columns when applied on the right (Example 2.1).

Example 2.1 (Permutation matrices)**.** Given matrix

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}$$

assume we want to move row 1 to row 2, row 2 to row 3, and row 3 to row 1, i.e.

$$\pi(1) = 2,\ \pi(2) = 3,\ \pi(3) = 1$$

We can do so by building matrix

$$\mathbf{P} = \begin{bmatrix} \mathbf{e}_{\pi^{-1}(1)} \\ \mathbf{e}_{\pi^{-1}(2)} \\ \mathbf{e}_{\pi^{-1}(3)} \end{bmatrix} = \begin{bmatrix} \mathbf{e}_3 \\ \mathbf{e}_1 \\ \mathbf{e}_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

and computing the product

$$\mathbf{PX} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} = \begin{bmatrix} x_{31} & x_{32} & x_{33} \\ x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{bmatrix}$$

If instead we multiply $\mathbf{X}$ by $\mathbf{P}^\top$, we will get the same permutation, but this time applied to the columns of $\mathbf{X}$, i.e.

$$\mathbf{XP}^\top = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} x_{13} & x_{11} & x_{12} \\ x_{23} & x_{21} & x_{22} \\ x_{33} & x_{31} & x_{32} \end{bmatrix}$$

□

We can then write

$$
\begin{aligned}
\mathbf{x}_1 &= \mathbf{P}_1^\top \mathbf{P}_1 \mathbf{x}_1 && (\mathbf{P}_1^\top = \mathbf{P}_1^{-1}) \\
&= \mathbf{P}_1^\top \mathbf{P}_1 \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) \\
&= \mathbf{P}_1^\top \sigma(\mathbf{P}_1 \mathbf{W}_1 \mathbf{x} + \mathbf{P}_1 \mathbf{b}_1) && (\sigma \text{ and } \mathbf{P} \text{ commute}) \\
\mathbf{x}_2 &= \mathbf{P}_2^\top \mathbf{P}_2 \mathbf{x}_2 \\
&= \mathbf{P}_2^\top \mathbf{P}_2 \sigma(\mathbf{W}_2 \mathbf{x}_1 + \mathbf{b}_2) \\
&= \mathbf{P}_2^\top \sigma(\mathbf{P}_2 \mathbf{W}_2 \mathbf{x}_1 + \mathbf{P}_2 \mathbf{b}_2) \\
&= \mathbf{P}_2^\top \sigma(\mathbf{P}_2 \mathbf{W}_2 \mathbf{P}_1^\top \sigma(\mathbf{P}_1 \mathbf{W}_1 \mathbf{x} + \mathbf{P}_1 \mathbf{b}_1) + \mathbf{P}_2 \mathbf{b}_2) \\
\mathbf{x}_3 &= \mathbf{W}_3 \mathbf{x}_2 + \mathbf{b}_3 \\
&= \underbrace{\mathbf{W}_3 \mathbf{P}_2^\top}_{\mathbf{W}_3'} \sigma(\underbrace{\mathbf{P}_2 \mathbf{W}_2 \mathbf{P}_1^\top}_{\mathbf{W}_2'} \sigma(\underbrace{\mathbf{P}_1 \mathbf{W}_1}_{\mathbf{W}_1'} \mathbf{x} + \underbrace{\mathbf{P}_1 \mathbf{b}_1}_{\mathbf{b}_1'}) + \underbrace{\mathbf{P}_2 \mathbf{b}_2}_{\mathbf{b}_2'}) + \underbrace{\mathbf{b}_3}_{\mathbf{b}_3'} \\
&= f_{\boldsymbol{\theta}}(\mathbf{x})
\end{aligned}
$$

Therefore, if we take a second MLP with parameters $\boldsymbol{\theta}'$ such that

$$
\begin{aligned}
\mathbf{W}_1' &= \mathbf{P}_1 \mathbf{W}_1 \\
\mathbf{b}_1' &= \mathbf{P}_1 \mathbf{b}_1 \\
\mathbf{W}_2' &= \mathbf{P}_2 \mathbf{W}_2 \mathbf{P}_1^\top \\
\mathbf{b}_2' &= \mathbf{P}_2 \mathbf{b}_2 \\
\mathbf{W}_3' &= \mathbf{W}_3 \mathbf{P}_2^\top \\
\mathbf{b}_3' &= \mathbf{b}_3
\end{aligned}
$$

we have $f_{\boldsymbol{\theta}'}(\mathbf{x}) = f_{\boldsymbol{\theta}}(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^{d_0 \times 1}$. The result we derived for a 3-layer MLP can be extended to a generic L-layer MLP:

Theorem 2.1 (Permutation invariance of MLPs)**.** *Let $\boldsymbol{\theta}, \boldsymbol{\theta}'$ be the parameters of two L-layer MLPs such that*

$$
\begin{aligned}
\mathbf{W}_\ell' &= \mathbf{P}_\ell \mathbf{W}_\ell \mathbf{P}_{\ell-1}^\top \\
\mathbf{b}_\ell' &= \mathbf{P}_\ell \mathbf{b}_\ell
\end{aligned}
$$

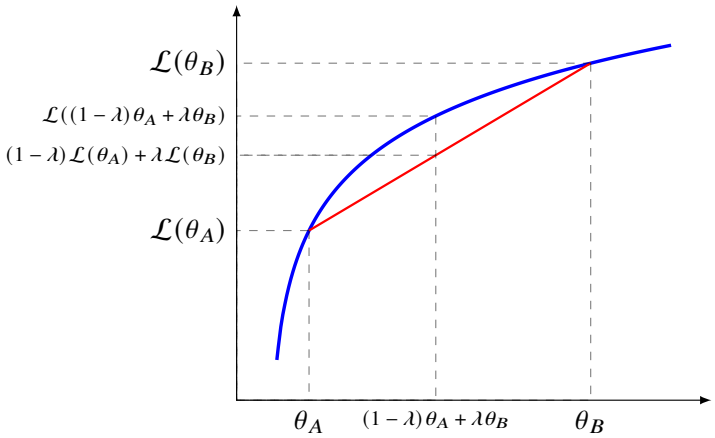


Figure 2.1: **Loss barrier.** The barrier along the linear path between $\boldsymbol{\theta}_A$ and $\boldsymbol{\theta}_B$ is the highest difference between the loss occurred when linearly connecting $\boldsymbol{\theta}_A, \boldsymbol{\theta}_B$ and the linear interpolation of the loss values at each of them. For visualization purposes, a 1D weight space is depicted.

for $\ell = 1, \ldots, L$ *with* $\mathbf{P}_0 = \mathbf{P}_L = \mathbf{I}$*. Then,*

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = f_{\boldsymbol{\theta}'}(\mathbf{x})$$

for all $\mathbf{x}$*.*

2.2 Linear mode connectivity

Definition 2.1 (Loss barrier)**.** Let $\boldsymbol{\theta}_A, \boldsymbol{\theta}_B$ be two solutions found when training an MLP by running SGD with different random initializations and data orders, and let $\mathcal{L}$ be the loss on either the training or test set. The *loss barrier* between $\boldsymbol{\theta}_A$ and $\boldsymbol{\theta}_B$ is defined as (Figure 2.1)

$$\mathcal{B}(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B) = \max_{\lambda \in [0,1]} \left\{ \mathcal{L}((1-\lambda)\boldsymbol{\theta}_A + \lambda\boldsymbol{\theta}_B) - ((1-\lambda)\mathcal{L}(\boldsymbol{\theta}_A) + \lambda\mathcal{L}(\boldsymbol{\theta}_B)) \right\}$$

Remark. Notice that

$$\mathcal{B}(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B) \begin{cases} > 0 & \text{if } \mathcal{L} \text{ is concave between } \boldsymbol{\theta}_A \text{ and } \boldsymbol{\theta}_B \\ = 0 & \text{if } \mathcal{L} \text{ is linear between } \boldsymbol{\theta}_A \text{ and } \boldsymbol{\theta}_B \\ < 0 & \text{if } \mathcal{L} \text{ is convex between } \boldsymbol{\theta}_A \text{ and } \boldsymbol{\theta}_B \end{cases}$$

Definition 2.2 (Linear mode connectivity (LMC))**.** $\boldsymbol{\theta}_A, \boldsymbol{\theta}_B$ are said to be *linear mode connected* if

$$\mathcal{B}(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B) \approx 0$$

Conjecture 2.1 (Entezari et al., 2022)**.** *There exists a permutation* $\pi = \{\mathbf{P}_1, \dots, \mathbf{P}_{L-1}\}$ *such that* $\boldsymbol{\theta}_A$ *and* $\pi(\boldsymbol{\theta}_B)$ *are linear mode connected.*

2.3 Alignment methods

In this section, we introduce two methods that build upon Conjecture 2.1 and aim at computing the aligning permutation by minimizing some measure of distance between model layers, one based on activations and the other on weight values.

2.3.1 Activation matching

The method we are going to describe aims at matching activations between MLPs, which captures the intuitive notion that two models should learn similar features in order to accomplish the same task.

Let $\mathbf{X}_\ell^A, \mathbf{X}_\ell^B \in \mathbb{R}^{d_\ell \times n}$ be the features produced in output by the ℓth layer of MLPs $\boldsymbol{\theta}_A$ and $\boldsymbol{\theta}_B$, respectively (where n is the batch size), and let $\mathbf{X}_{:,i}^A, \mathbf{X}_{:,i}^B$ be the ith column (i.e. feature) of $\mathbf{X}_\ell^A$ and $\mathbf{X}_\ell^B$, respectively. The goal is to optimize

$$\mathbf{P}_\ell^* = \arg\min_{\mathbf{P}_\ell} \sum_{i=1}^{n} \|\mathbf{X}_{:,i}^A - \mathbf{P}_\ell \mathbf{X}_{:,i}^B\|_2^2$$

Algorithm 2.1: ACTIVATION-MATCHING (Ainsworth et al., 2022)

Input: θ_A, θ_B
Output: $\pi = \{\mathbf{P}_1, \dots \mathbf{P}_{L-1}\}$ minimizing $\|\mathbf{X}_\ell^A - \mathbf{P}_\ell \mathbf{X}_\ell^B\|_F^2$

$\mathbf{P}_1 \leftarrow \mathbf{I}, \dots, \mathbf{P}_{L-1} \leftarrow \mathbf{I}$
for $\ell \leftarrow 1$ **to** $L-1$ **do**
  $\mathbf{P}_\ell \leftarrow \text{SolveLAP}(\mathbf{X}_\ell^A (\mathbf{X}_\ell^B)^\top)$
end

for $\ell = 1, \dots, L-1$. Notice that

$$\begin{aligned}\sum_{i=1}^n \|\mathbf{X}_{:,i}^A - \mathbf{P}_\ell \mathbf{X}_{:,i}^B\|_2^2 &= \|\mathbf{X}_\ell^A - \mathbf{P}_\ell \mathbf{X}_\ell^B\|_F^2 \\ &= \|\mathbf{X}_\ell^A\|_F^2 - 2\langle \mathbf{X}_\ell^A, \mathbf{P}_\ell \mathbf{X}_\ell^B \rangle_F + \underbrace{\|\mathbf{P}_\ell \mathbf{X}_\ell^B\|_F^2}_{=\|\mathbf{X}_\ell^B\|_F^2}\end{aligned}$$

where $\langle \mathbf{A}, \mathbf{B} \rangle_F = \sum_i \sum_j a_{ij} b_{ij}$ is the Frobenius inner product. It follows that

$$\mathbf{P}_\ell^* = \arg\max_{\mathbf{P}_\ell} \langle \mathbf{X}_\ell^A, \mathbf{P}_\ell \mathbf{X}_\ell^B \rangle_F$$

which can be further rewritten by observing that

$$\begin{aligned}\langle \mathbf{X}_\ell^A, \mathbf{P}_\ell \mathbf{X}_\ell^B \rangle_F &= \text{tr}((\mathbf{X}_\ell^A)^\top \mathbf{P}_\ell \mathbf{X}_\ell^B) && (\langle \mathbf{A}, \mathbf{B} \rangle_F = \text{tr}(\mathbf{A}^\top \mathbf{B})) \\ &= \text{tr}((\mathbf{P}_\ell \mathbf{X}_\ell^B)^\top \mathbf{X}_\ell^A) && (\text{tr}(\mathbf{A}) = \text{tr}(\mathbf{A}^\top)) \\ &= \text{tr}((\mathbf{X}_\ell^B)^\top \mathbf{P}_\ell^\top \mathbf{X}_\ell^A) \\ &= \text{tr}(\mathbf{P}_\ell^\top \mathbf{X}_\ell^A (\mathbf{X}_\ell^B)^\top) && (\text{tr}(\mathbf{ABC}) = \text{tr}(\mathbf{BCA})) \\ &= \langle \mathbf{P}_\ell, \mathbf{X}_\ell^A (\mathbf{X}_\ell^B)^\top \rangle_F && (2.1)\end{aligned}$$

Therefore, what we want to compute is ultimately

$$\mathbf{P}_\ell^* = \arg\max_{\mathbf{P}_\ell} \langle \mathbf{P}_\ell, \mathbf{X}_\ell^A (\mathbf{X}_\ell^B)^\top \rangle_F$$

which is an instance of the *linear assignment problem* (Problem 2.1). Algorithm 2.1 shows a pseudocode for the procedure.

Problem 2.1 (Linear assignment problem (LAP))**.** Given

$$\begin{aligned} A &= \text{set of agents} \\ T &= \text{set of tasks} \\ w_{ij} &= \text{weight of assigning agent } i \text{ to task } j \\ x_{ij} &= \begin{cases} 1 & \text{if agent } i \text{ is assigned to task } j \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

the corresponding assignment problem can be formulated as an integer linear program, i.e.

$$\begin{aligned} &\text{maximize } \textstyle\sum_{i\in A}\sum_{j\in T} w_{ij}x_{ij} \\ &\text{with constraints:} \\ &\quad \textstyle\sum_{j\in T} x_{ij} = 1 \quad \forall i \in A \qquad \text{(each agent is assigned to one task)} \\ &\quad \textstyle\sum_{i\in A} x_{ij} = 1 \quad \forall j \in T \qquad \text{(each task is assigned to one agent)} \\ &\quad x_{ij} \in \mathbb{Z} \cap [0,1] \quad \forall i \in A, j \in T \end{aligned}$$

□

Once we have $\mathbf{P}_\ell^*$ for $\ell = 1, \ldots, L-1$, all we are left to do is to permute $\boldsymbol{\theta}_B$ as in Theorem 2.1, i.e.

$$\begin{aligned} \mathbf{W}_\ell^{B'} &= \mathbf{P}_\ell^* \mathbf{W}_\ell^B \mathbf{P}_{\ell-1}^{*\top} \\ \mathbf{b}_\ell^{B'} &= \mathbf{P}_\ell^* \mathbf{b}_\ell^B \end{aligned}$$

with $\mathbf{P}_0^* = \mathbf{P}_L^* = \mathbf{I}$.

Computationally, this entire process is relatively lightweight: the $\mathbf{X}_\ell^A$ and $\mathbf{X}_\ell^B$ matrices can be computed in a single pass over a training batch, and the LAP can be solved in polynomial time (Kuhn, 2010). Also, conveniently, the aligning at each layer is independent of that at every other layer, resulting in a separable and straightforward optimization problem; this advantage will not be enjoyed by the following method.

Algorithm 2.2: WEIGHT-MATCHING (Ainsworth et al., 2022)

Input: $\boldsymbol{\theta}_A, \boldsymbol{\theta}_B$
Output: $\pi = \{\mathbf{P}_1, \dots \mathbf{P}_{L-1}\}$ minimizing (approximately) $\|\boldsymbol{\theta}_A - \pi(\boldsymbol{\theta}_B)\|_2^2$
$\mathbf{P}_1 \leftarrow \mathbf{I}, \dots, \mathbf{P}_{L-1} \leftarrow \mathbf{I}$
repeat
  for $\ell \in \text{RandomPermutation}(1, \dots, L-1)$ **do**
    $\mathbf{P}_\ell \leftarrow \text{SolveLAP}(\mathbf{W}_\ell^A \mathbf{P}_{\ell-1} (\mathbf{W}_\ell^B)^\top + (\mathbf{W}_{\ell+1}^A)^\top \mathbf{P}_{\ell+1} \mathbf{W}_{\ell+1}^B)$
  end
until *convergence*

2.3.2 Weight matching

Following the intuition that similar rows of weight matrices should compute similar features, this method focuses on aligning weights instead of activations.

Ideally, we would like to solve the optimization problem

$$\begin{aligned}
\pi^* &= \arg\min_{\pi} \|\boldsymbol{\theta}_A - \pi(\boldsymbol{\theta}_B)\|_2^2 \\
&= \arg\max_{\pi} \langle \boldsymbol{\theta}_A, \pi(\boldsymbol{\theta}_B) \rangle_2 \\
&= \arg\max_{\mathbf{P}_1, \dots, \mathbf{P}_{L-1}} \textstyle\sum_{\ell=1}^{L} \langle \mathbf{W}_\ell^A, \mathbf{P}_\ell \mathbf{W}_\ell^B \mathbf{P}_{\ell-1}^\top \rangle_F
\end{aligned}$$

where, for the sake of simplicity, we did not consider bias terms. Since the above expression is NP-hard, we approximate the desired solution by focusing on a single $\mathbf{P}_\ell$ while holding the other fixed, i.e.

$$\begin{aligned}
\tilde{\mathbf{P}}_\ell &= \arg\max_{\mathbf{P}_\ell} \langle \mathbf{W}_\ell^A, \mathbf{P}_\ell \mathbf{W}_\ell^B \tilde{\mathbf{P}}_{\ell-1}^\top \rangle_F + \langle \mathbf{W}_{\ell+1}^A, \tilde{\mathbf{P}}_{\ell+1} \mathbf{W}_{\ell+1}^B \mathbf{P}_\ell^\top \rangle_F \\
&= \arg\max_{\mathbf{P}_\ell} \langle \mathbf{P}_\ell, \mathbf{W}_\ell^A \tilde{\mathbf{P}}_{\ell-1} (\mathbf{W}_\ell^B)^\top \rangle_F + \langle \mathbf{P}_\ell, (\mathbf{W}_{\ell+1}^A)^\top \tilde{\mathbf{P}}_{\ell+1} \mathbf{W}_{\ell+1}^B \rangle_F \qquad \text{(as in 2.1)} \\
&= \arg\max_{\mathbf{P}_\ell} \langle \mathbf{P}_\ell, \mathbf{W}_\ell^A \tilde{\mathbf{P}}_{\ell-1} (\mathbf{W}_\ell^B)^\top + (\mathbf{W}_{\ell+1}^A)^\top \tilde{\mathbf{P}}_{\ell+1} \mathbf{W}_{\ell+1}^B \rangle_F
\end{aligned}$$

which, again, is an LAP (Algorithm 2.2), and the resulting permutation matrices need to be applied to $\boldsymbol{\theta}_B$ as in Theorem 2.1.

Notice that, unlike Algorithm 2.1, this procedure ignores the data distribution entirely, as it does not compute a forward pass—which makes it even faster and

well suited to run exclusively on CPU.

Chapter 3

Experimental results

Experimental settings (De Luigi et al., 2023) In all the experiments reported in this chapter, INRs are MLPs with 3 hidden layers with 512 nodes each and SIREN activation function (Sitzmann et al., 2020b), which are trained on ModelNet40 shapes (Wu et al., 2015), as we choose to focus on point clouds. UDF values $y_i \in \mathbb{R}$ of points $\mathbf{x}_i \in \mathbb{R}^3$ are converted into values y_i^{bce} continuously spanned in the $[0, 1]$ range, with 0 and 1 representing the predefined maximum distance from the surface and the surface level set (i.e. distance equal to zero), respectively. The MLP is trained to minimize the Binary Cross Entropy (BCE) between those labels and the predicted values, namely

$$\mathcal{L}_{\text{bce}} = -\frac{1}{N}\sum_{i=1}^{N} y_i^{\text{bce}} \ln(\hat{y}_i) + (1 - y_i^{\text{bce}}) \ln(1 - \hat{y}_i)$$

where $\hat{y}_i = \sigma(f_{\boldsymbol{\theta}}(\mathbf{x}_i))$, with σ being the sigmoid function. Specifically, 500K $(\mathbf{x}_i, y_i^{\text{bce}})$ pairs are sampled by taking 250K points close to the surface, 200K at a medium-far distance for the surface, 25K far from the surface, and 25K scattered uniformly in the volume. Then, the MLP is trained with Adam optimizer (Kingma and Ba, 2015) and learning rate 1e−4 for 500 steps, where at each step the loss is computed on 10K pairs randomly sampled from the 500K precomputed ones.

Initialization To test the effect of initialization on LMC (Definition 2.2), we consider two MLPs A and B, trained on the same shape but with a different

sampling of the corresponding point cloud, and divide our experiments into the following categories:

- `same`: both $init_A$ and $init_B$ are equal to $init_{\texttt{inr2vec}}$, i.e. the initialization shared by all the MLPs `inr2vec` was trained on.
- `diff`: $init_A = init_{\texttt{inr2vec}}$, whereas $init_B \neq init_A$ is different for each shape.
- `diff-weightm`: same as `diff`, but B's weights have been rearranged according to the permutation computed by WEIGHT-MATCHING (Algorithm 2.2).
- `diff-actm`: same as `diff-weightm`, but ACTIVATION-MATCHING has been applied instead (Algorithm 2.1).

3.1 Behavior along the linear path

We study the performance of models encountered when linearly connecting A and B, i.e. MLPs with parameters $(1-\lambda)\boldsymbol{\theta}_A + \lambda\boldsymbol{\theta}_B$, where $\lambda \in [0, 1]$. In particular, we divide the $[0, 1]$ interval into 25 evenly spaced values of λ and focus on the MLPs found at those locations.

3.1.1 Loss

When plotting the loss between $\boldsymbol{\theta}_A$ ($\lambda = 0$) and $\boldsymbol{\theta}_B$ ($\lambda = 1$), we observe the following phenomena (Figure 3.1):

- `same`: the loss is mostly flat, i.e. $\boldsymbol{\theta}_A$ and $\boldsymbol{\theta}_B$ are linear mode connected. This means that training INRs starting from the same initialization causes the corresponding SGD solutions to lie in the same basin of the loss landscape, which in turn might be a determining factor for the convergence of `inr2vec`.
- `diff`: we consistently observe two peaks (at the 4^{th} and 4^{th} to last values of λ) with a depression in the middle, sometimes even lower than the extremities. This would seem to suggest that $\boldsymbol{\theta}_B$ is "two basins away" from $\boldsymbol{\theta}_A$, and that models midway across the linear path perform on par with (or even better

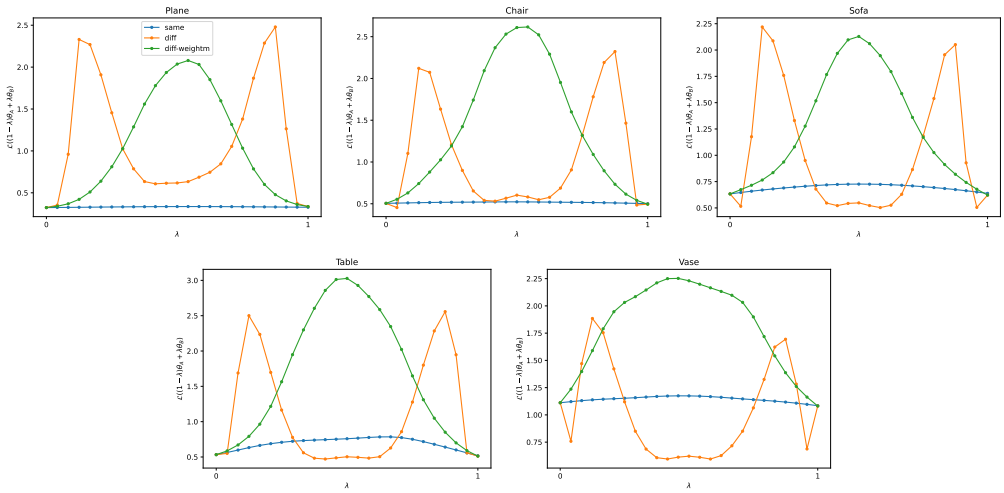


Figure 3.1: **Loss along the linear path between θ_A and θ_B (WEIGHT-MATCHING). Blue** (`same`)**:** $init_A = init_B$. **Orange** (`diff`)**:** $init_A \neq init_B$, where $init_B$ is different for each shape. **Green** (`diff-weightm`)**:** same as `diff`, but θ_B has been rearranged according to the permutation computed by WEIGHT-MATCHING. The loss at each λ is computed as the average of 10K points: 5K close to the surface, 4K at a medium-far distance from the surface, 500 far from the surface, and 500 scattered uniformly in the volume.

than) both θ_A and θ_B. It is especially remarkable how, although $init_B$ is different for each shape, the loss curves always exhibit the same behavior.

- `diff-weightm`: the loss has a bell-like shape, with a single peak around $\lambda = 0.5$. We are still far from LMC—in fact, the loss barrier is sometimes larger (i.e. the peak is higher) than in the `diff` case. On the flip side, the single peak should mean that WEIGHT-MATCHING has moved θ_B just one basin away from θ_A instead of two, which in some sense can be seen as getting closer to θ_A. It is worth noting that ACTIVATION-MATCHING, on the other hand, produces loss curves that are analogous to those of the `diff` experiment, suggesting that the permutation did not move θ_B significantly (Figure 3.2). Therefore, we will focus on WEIGHT-MATCHING for the remainder of this chapter.

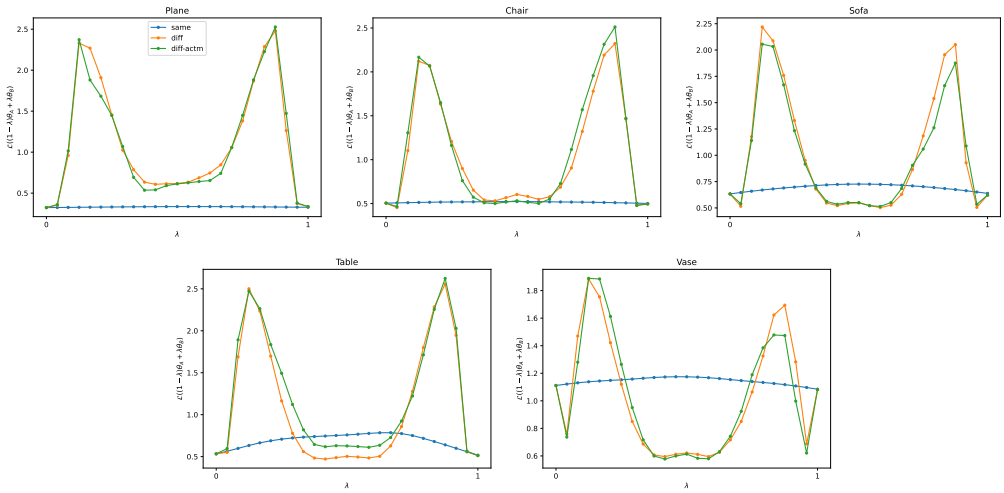


Figure 3.2: **Loss along the linear path between θ_A and θ_B** (ACTIVATION-MATCHING)**. Blue** (`same`)**:** $init_A = init_B$. **Orange** (`diff`)**:** $init_A \neq init_B$, where $init_B$ is different for each shape. **Green** (`diff-actm`)**:** same as `diff`, but θ_B has been rearranged according to the permutation computed by ACTIVATION-MATCHING. The loss at each λ is computed as the average over 10K points: 5K close to the surface, 4K at a medium-far distance from the surface, 500 far from the surface, and 500 scattered uniformly in the volume.

3.1.2 Predictions

The most perplexing result of Figure 3.1 is that, in the `diff` experiment, loss values around $\lambda = 0.5$ are similar to (or lower than) those at $\lambda = 0$ and $\lambda = 1$, which should mean that models midway across the linear path produce predictions of comparable (or superior) quality to those of A and B. However, the point clouds reconstructed from UDF values predicted by such MLPs show no correlation between low central `diff` loss and the quality of the predictions, as point clouds quickly degrade as we move away from $\lambda = 0$ and do not improve when the loss starts decreasing after the first peak; the only acceptable reconstructions are those corresponding to models A, B, and their immediate neighbors (Figure 3.10). On the other hand, the `diff-weightm` point cloud progression is much smoother, and shows a clear correlation between loss increase and decrease in reconstruction quality (Figure 3.11): although WEIGHT-MATCHING does not lead to LMC as in the `same` experiment (Figure 3.9), it moves θ_B into a much more interpretable and well-behaved region of the parameter space.

An explanation for the `diff` loss behavior can instead be found in the distribution of UDF values predicted by MLPs along the linear path: peaks in loss values correspond to a majority of predictions of maximum distance (which is also true for `diff-weightm`), whereas the central depression coincide with a majority of predictions halfway between the maximum distance and the surface itself.

3.2 Additional experiments

Similarity to the identity The empirical evidence of LMC observed when INRs share the same initialization (Figure 3.9) suggests that initialization is indeed the determining factor that leads SGD solutions to lie in the same basin of the loss landscape. By the same token, if WEIGHT-MATCHING truly computes the permutation that moves B into the same low-loss region as A, it should also be able to detect that, when $init_A = init_B$, there is no need to permute B's parameters. This can be verified by counting the number of 1s on the diagonal of permutation matrices, as a measure of similarity to the identity matrix $\mathbf{I}$—and in fact, when the initialization is the same, WEIGHT-MATCHING returns matrices that are either exactly $\mathbf{I}$ or very close to it, whereas when $init_A \neq init_B$ they largely differ from $\mathbf{I}$ (Figure 3.3).

Embedding distance We investigate the effect of WEIGHT-MATCHING on `inr2vec` embeddings by computing $\|\mathbf{emb}_A - \mathbf{emb}_B\|_2$ in the `same`, `diff`, and `diff-weightm` settings, where $\mathbf{emb}_A$ and $\mathbf{emb}_B$ are the outputs of the `inr2vec` encoder when given as inputs $\boldsymbol{\theta}_A$ and $\boldsymbol{\theta}_B$, respectively. Our results are consistent with previous observations: WEIGHT-MATCHING is able to reduce the distance between embeddings when $init_A \neq init_B$, although it does not make them as close as they are when $init_A = init_B$ (Figure 3.4).

3.3 Variations of weight matching

In this section, we go through some preprocessing steps we tested in conjunction with WEIGHT-MATCHING, in an attempt to improve on the results of the previous

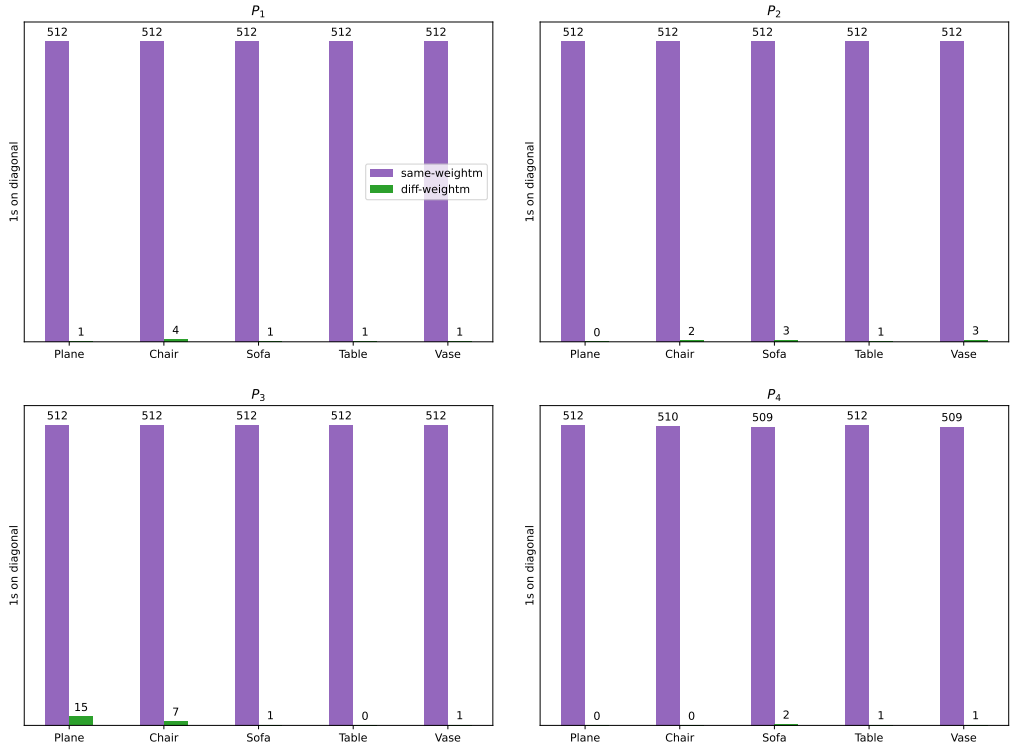


Figure 3.3: **Number of 1s on the diagonal of permutation matrices computed by WEIGHT-MATCHING.** Permutation matrices are 512×512. **Purple** (`same-weightm`)**:** $init_A = init_B$ and $\boldsymbol{\theta}_B$ has been rearranged according to the permutation computed by WEIGHT-MATCHING. **Green** (`diff-weightm`)**:** same as `same-weightm`, but $init_A \neq init_B$, where $init_B$ is different for each shape.

section while keeping the core method unchanged.

3.3.1 Min-max

Up to this point, we have applied WEIGHT-MATCHING on unnormalized data, i.e. the parameters of the input MLPs in their original values. However, since the initialization scheme of SIREN leads to bigger weights and biases in the first layer (Sitzmann et al., 2020b), this raises the question of whether WEIGHT-MATCHING is incentivized to focus on those bigger values in order to minimize the Euclidean distance at the core of the LAP, while neglecting weights and biases that are smaller but potentially more relevant for the alignment task. To address this concern, we

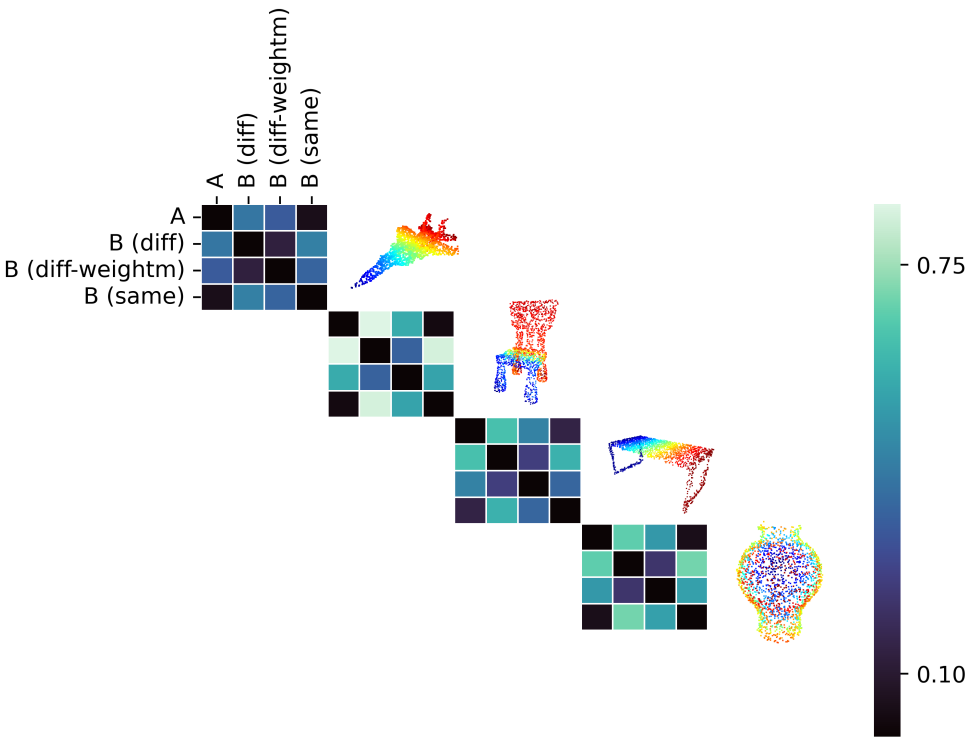


Figure 3.4: **Euclidean distances between `inr2vec` embeddings.** INRs of the same shape are given as input to the `inr2vec` encoder and the Euclidean distances of the output embeddings are compared to show the effect of different initializations with and without weight permutations.

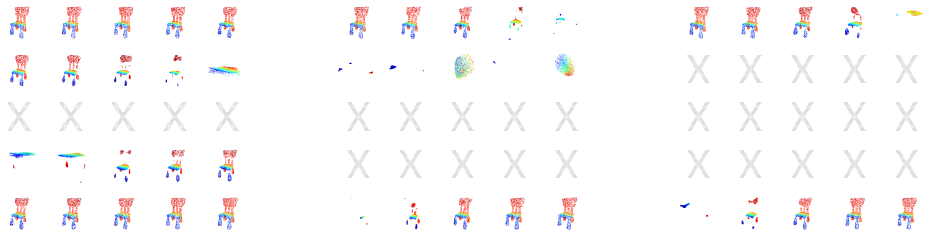
test two versions of *min-max normalization*:

- `minmax1`: the maximum and minimum are computed for each weight matrix and bias vector individually, i.e.

$$\mathbf{W}_\ell \leftarrow \frac{\mathbf{W}_\ell - \min(\mathbf{W}_\ell)}{\max(\mathbf{W}_\ell) - \min(\mathbf{W}_\ell)}$$
$$\mathbf{b}_\ell \leftarrow \frac{\mathbf{b}_\ell - \min(\mathbf{b}_\ell)}{\max(\mathbf{b}_\ell) - \min(\mathbf{b}_\ell)}$$

  where $\min(\mathbf{W}_\ell)$ is the smallest entry of $\mathbf{W}_\ell$ and $\max(\mathbf{W}_\ell)$ the largest.

- `minmax2`: the maximum and minimum are computed for each pair of weight



(a) unnormalized (b) `minmax1` (c) `minmax2`

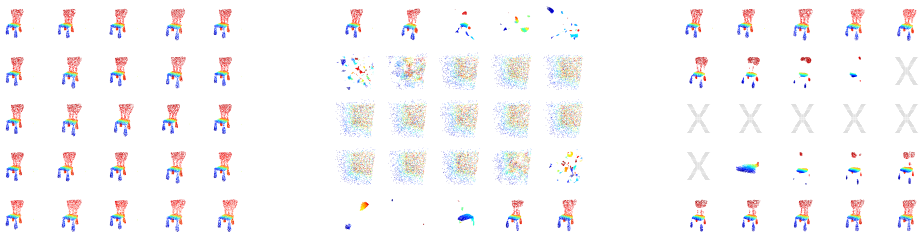
Figure 3.5: **Effect of min-max normalization on reconstructions along the linear path.** Point clouds are reconstructed from INR predictions (top left is θ_A, bottom right is θ_B). **(a):** `diff-weightm` results with no prior normalization applied to θ_A, θ_B. **(b):** `diff-weightm` results, where, before applying WEIGHT-MATCHING, θ_A, θ_B have been normalized by applying min-max independently on each weight matrix and bias vector. **(c):** `diff-weightm` results, where, before applying WEIGHT-MATCHING, θ_A, θ_B have been normalized by applying min-max to each pair of weight matrices and bias vectors from the same layer of MLPs A, B.

matrices and bias vectors from the same layer of models A and B, i.e.

$$\mathbf{W}_\ell^A \leftarrow \frac{\mathbf{W}_\ell^A - \min(\mathbf{W}_\ell^A, \mathbf{W}_\ell^B)}{\max(\mathbf{W}_\ell^A, \mathbf{W}_\ell^B) - \min(\mathbf{W}_\ell^A, \mathbf{W}_\ell^B)}$$
$$\mathbf{W}_\ell^B \leftarrow \frac{\mathbf{W}_\ell^B - \min(\mathbf{W}_\ell^A, \mathbf{W}_\ell^B)}{\max(\mathbf{W}_\ell^A, \mathbf{W}_\ell^B) - \min(\mathbf{W}_\ell^A, \mathbf{W}_\ell^B)}$$
$$\mathbf{b}_\ell^A \leftarrow \frac{\mathbf{b}_\ell^A - \min(\mathbf{b}_\ell^A, \mathbf{b}_\ell^B)}{\max(\mathbf{b}_\ell^A, \mathbf{b}_\ell^B) - \min(\mathbf{b}_\ell^A, \mathbf{b}_\ell^B)}$$
$$\mathbf{b}_\ell^B \leftarrow \frac{\mathbf{b}_\ell^B - \min(\mathbf{b}_\ell^A, \mathbf{b}_\ell^B)}{\max(\mathbf{b}_\ell^A, \mathbf{b}_\ell^B) - \min(\mathbf{b}_\ell^A, \mathbf{b}_\ell^B)}$$

where $\min(\mathbf{W}_\ell^A, \mathbf{W}_\ell^B)$ is the smallest entry of $\mathbf{W}_\ell^A$ and $\mathbf{W}_\ell^B$ combined, while $\max(\mathbf{W}_\ell^A, \mathbf{W}_\ell^B)$ is the largest.

Once θ_A and θ_B have been normalized, they are given as input to WEIGHT-MATCHING and the resulting permutation is applied to the original (i.e. unnormalized) θ_B, which is then compared to the original θ_A. Unfortunately, both flavors of min-max show a behavior along the linear path that is further from LMC than their unnormalized counterpart, suggesting that this kind of normalization acts as an additional obstacle in the search for the aligning permutation (Figure 3.5).



(a) `same` (b) `diff` (c) `diff-weightm`

Figure 3.6: **Effect of pruning on reconstructions along the linear path.** Point clouds are reconstructed from INRs pruned by 80%, i.e. parameters whose absolute value is $\leq$ than the 80^{th} percentile of all weights and biases across all layers are zeroed out. Top left is $\boldsymbol{\theta}_A$, bottom right is $\boldsymbol{\theta}_B$.

3.3.2 Pruning

Deep neural networks are known to provide a vastly redundant parametrization of the underlying function, and methods for removing unnecessary weights from trained models (called *pruning* techniques) can reduce parameter counts by more than 90% without harming accuracy, while decreasing storage requirements and making inference more efficient (LeCun et al., 1989; Hassibi and Stork, 1992; Han et al., 2015; Li et al., 2017). The reason why this over-parametrization is desirable, however, is that most pruned networks are harder to train than the original model, unless they are *winning tickets*, i.e. subnetworks whose initial weights allow them to match the test accuracy of the original network after training for at most the same number of iterations (Frankle and Carbin, 2019).

Following the intuition that the alignment problem should be easier in a pruned (and therefore smaller) parameter space, we apply WEIGHT-MATCHING to pruned MLPs and check whether we are able to achieve LMC in the pruned space. In our setting, pruning N% of parameters means computing the N^{th} percentile of all weights and biases across all layers and then setting to zero those parameters whose absolute value is less or equal than that percentile. We find $N = 80$ to be the most we can prune while still preserving an acceptable prediction quality (Figure 3.12). Our results, however, show that the linear path from $\boldsymbol{\theta}_A$ to $\boldsymbol{\theta}_B$ in the pruned space behaves analogously to its counterpart in the original space, i.e. we observe LMC in the `same` experiment, whereas `diff-weightm`, although not linear mode

connected, exhibits a smoother degradation of reconstruction quality than `diff` (Figure 3.6).

3.4 Learning linear mode connectivity

Having thoroughly dissected WEIGHT-MATCHING without, however, being able to achieve LMC, we finally step aside from combinatorial methods and turn our attention to an entirely deep-learning based approach: given the usual INRs A and B with $init_A \neq init_B$, $\boldsymbol{\theta}_B$ is fed into an encoder with the same architecture as the one of `inr2vec` and the resulting embedding becomes the input of a *hyper-network* decoder (Ha et al., 2017), which in turn outputs a new set of weights $\hat{\boldsymbol{\theta}}_B$, optimized to be linear mode connected to $\boldsymbol{\theta}_A$. We call this procedure HYPER-MATCHING.

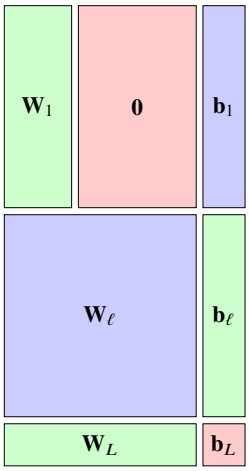


Figure 3.7: Encoder input

Encoder input Similarly to De Luigi et al. (2023), the actual encoder input is a matrix constructed by stacking weights and biases of B on top of each other: $\mathbf{W}_1 \in \mathbb{R}^{H\times 3}, \mathbf{b}_1 \in \mathbb{R}^{H\times 1}, \mathbf{W}_\ell \in \mathbb{R}^{H\times H}, \mathbf{b}_\ell \in \mathbb{R}^{H\times 1}, \mathbf{W}_L \in \mathbb{R}^{1\times H}, \mathbf{b}_L \in \mathbb{R}$ for $\ell = 2, \dots L-1$ are combined into a $((L-1)H+1) \times (H+1)$ matrix (Figure 3.7). In our setting, as we already mentioned, $L = 5$ and $H = 512$.

Training At the beginning of the training procedure, 500K points are sampled from the point cloud with the usual close–medium–far–uniform split, together with their UDF labels. Then, at each step:

- $\boldsymbol{\theta}_B$ goes into the encoder and the encoder output goes into the decoder, which in turn outputs $\hat{\boldsymbol{\theta}}_B$.
- 4 other sets of weights are computed, i.e. $\hat{\boldsymbol{\theta}}_i = (1-\lambda_i)\boldsymbol{\theta}_A + \lambda_i \hat{\boldsymbol{\theta}}_B$, where $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ are randomly sampled from intervals $[0, 0.25)$, $[0.25, 0.5)$,

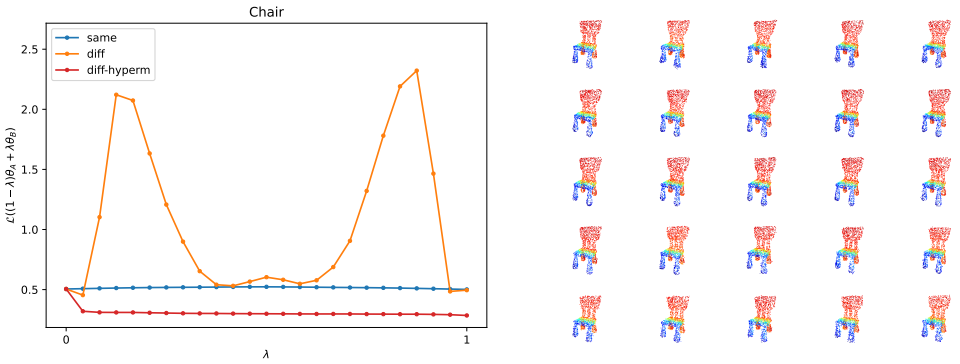


Figure 3.8: **HYPER-MATCHING results. Left:** loss along the linear path (red). `same` and `diff` results are shown for reference. **Right:** Point cloud reconstructions from INR predictions along the linear path. Top left is $\boldsymbol{\theta}_A$, bottom right is $\hat{\boldsymbol{\theta}}_B$.

$[0.5, 0.75)$, $[0.75, 1)$, respectively.

- 10K points are randomly selected out of the 500K previously sampled and the BCE between the MLP predictions at those points and their ground truth labels is computed for each $\hat{\boldsymbol{\theta}}_i$ and for $\hat{\boldsymbol{\theta}}_B$.

Finally, the total loss is the average of those 5 BCE terms, and the whole encoder–decoder architecture is optimized end-to-end for 1K epochs with AdamW optimizer (Loshchilov and Hutter, 2019), weight decay 1e−2, and 1cycle learning rate policy (Smith and Topin, 2017) with initial value 1e−4.

Results As shown by the loss and reconstructions along the linear path between $\boldsymbol{\theta}_A$ and $\hat{\boldsymbol{\theta}}_B$ (Figure 3.8), HYPER-MATCHING results are comparable to those of the `same` experiment and far superior to WEIGHT-MATCHING (Figure 3.11), both in terms of LMC and in the quality of the reconstructions themselves, which do not degrade when going from A to $\hat{B}$. However, this method still produces some undesired outcomes, concerning both `inr2vec` embeddings—which are further away from `same` embeddings than their WEIGHT-MATCHING counterparts (Figure 3.4) for all the shapes we tested—and point clouds reconstructed from the `inr2vec` decoder, as we will further discuss in Chapter 4.

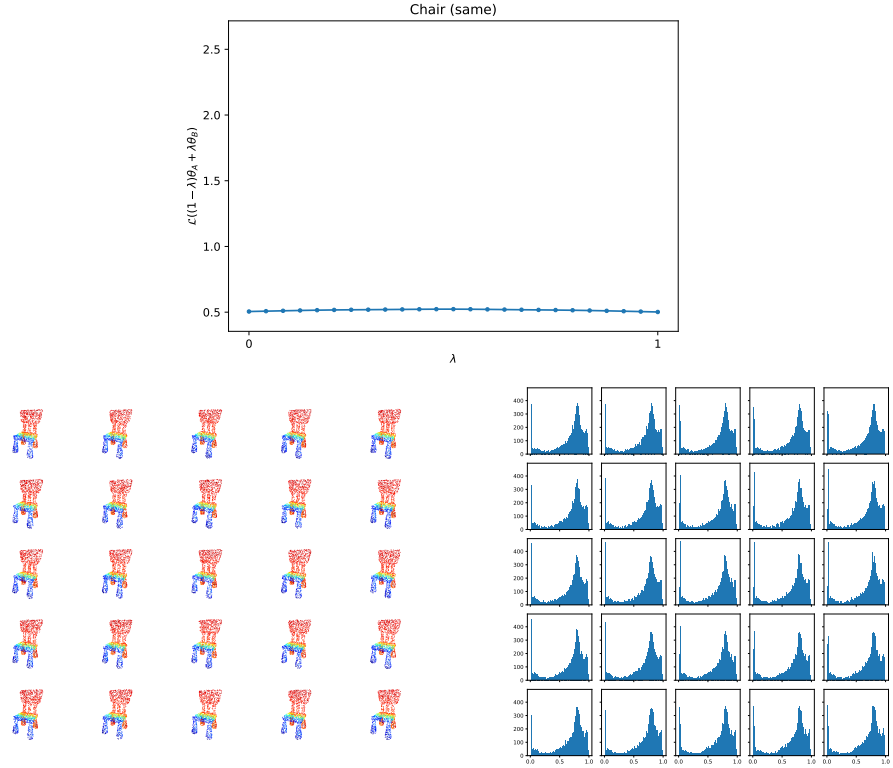


Figure 3.9: **Predictions along the linear path between θ_A and θ_B** (`same`)**. Top:** loss (average of 10K points). **Bottom left:** point clouds reconstructed from INR predictions (top left is θ_A, bottom right is θ_B). **Bottom right:** histograms of UDF values predicted by the corresponding INR (10K values, where 0 is maximum distance and 1 is surface).

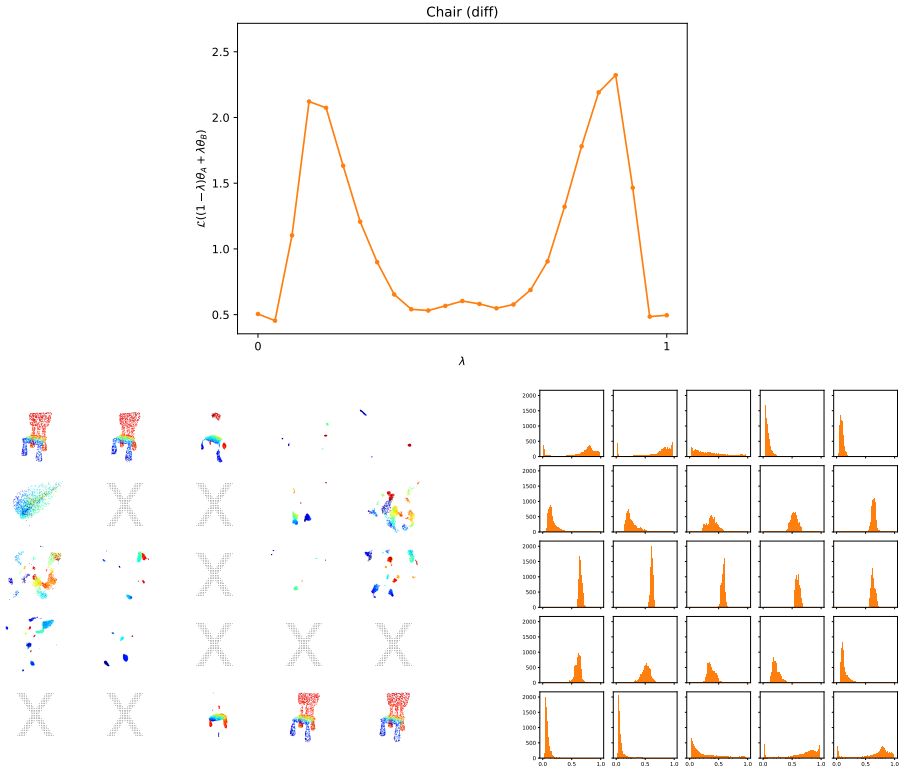


Figure 3.10: **Predictions along the linear path between θ_A and θ_B (`diff`). Top:** loss (average of 10K points). **Bottom left:** point clouds reconstructed from INR predictions (top left is θ_A, bottom right is θ_B). **Bottom right:** histograms of UDF values predicted by the corresponding INR (10K values, where 0 is maximum distance and 1 is surface). Whenever the reconstruction algorithm (Chibane et al., 2020; De Luigi et al., 2023) was not able to produce an output in less than 1 minute, an "X" symbol is shown in place of the corresponding point cloud.

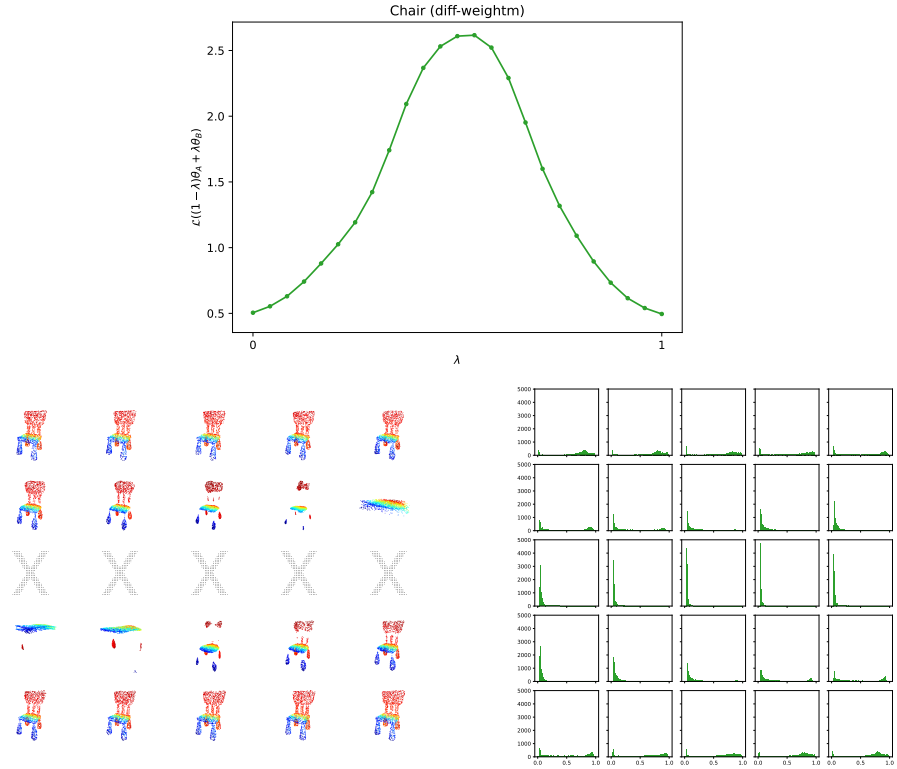


Figure 3.11: **Predictions along the linear path between θ_A and θ_B** (`diff-weightm`)**. Top:** loss (average of 10K points). **Bottom left:** point clouds reconstructed from INR predictions (top left is θ_A, bottom right is θ_B). **Bottom right:** histograms of UDF values predicted by the corresponding INR (10K values, where 0 is maximum distance and 1 is surface). Whenever the reconstruction algorithm (Chibane et al., 2020; De Luigi et al., 2023) was not able to produce an output in less than 1 minute, an "X" symbol is shown in place of the corresponding point cloud.

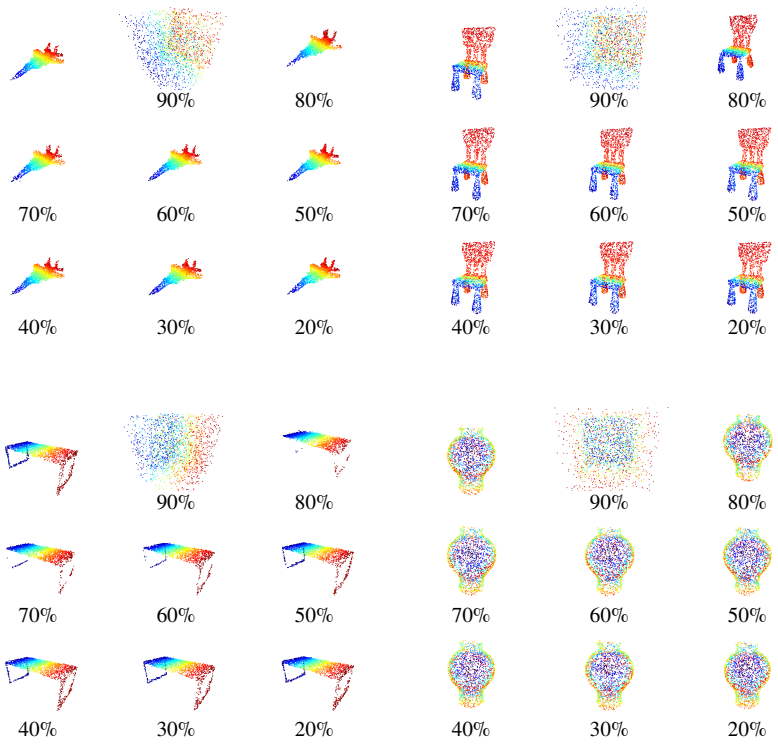


Figure 3.12: **Effect of pruning on reconstruction quality.** Point clouds are reconstructed from INRs pruned by N%, i.e. parameters whose absolute value is $\leq$ than the N^{th} percentile of all weights and biases across all layers are zeroed out. Top left of each shape corresponds to the original (i.e. unpruned) INR.

Chapter 4

Conclusions and future work

We tackled the task of aligning MLPs trained starting from different initializations in the context of INRs of 3D shapes. Although we mainly focused on combinatorial methods computing permutations via LAPs, a purely deep-learning based approach has proved to be much more effective in achieving LMC between INRs, while also holding the potential to work with multiple shapes by extending its training set—which is ultimately what would be needed in order for `inr2vec` to handle arbitrary INRs. However, there is a catch: all the methods described in this work produce an INR B such that, when given as input to `inr2vec`, the point cloud reconstructed from the decoder output is extremely noisy, to the point of being unrecognizable (and the quality of decoder reconstructions progressively degrades along the linear path from A to B). This observation suggests one of two possibilities: either HYPER-MATCHING needs further refinement—so that, in addition to achieving LMC, also produces sensible point cloud reconstructions out of the `inr2vec` decoder—or, on a more fundamental level, LMC is a necessary condition for the convergence of `inr2vec`—as convergence occurs when INRs share the same initialization, in which case LMC is observed—but is not sufficient, as other unknown factors are at play. Both questions constitute a natural continuation of our work and are left for future investigation.

Bibliography

Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, volume 2, 1989. 23

Robert Hecht-Nielsen. On the algebraic structure of feedforward network weight spaces. In *Advanced Neural Computers*, pages 129–135, 1990. 3, 6

Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems*, volume 5, 1992. 23

Harold W. Kuhn. *The Hungarian Method for the Assignment Problem*, pages 29–47. Springer, Berlin, Heidelberg, 2010. 12

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*, pages 1135–1143, 2015. 23

Diederick P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. 15

Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *Ieee/rsj International Conference on Intelligent Robots and Systems*, pages 922–928, 2015. 2

Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric

shapes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, 2015. 15

David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *International Conference on Learning Representations*, 2017. 24

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations*, 2017. 23

Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 5105–5114, 2017. 2

Leslie N. Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates, 2017, arXiv:1708.07120. 25

Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred Hamprecht. Essentially no barriers in neural network energy landscape. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 1309–1318, 10–15 Jul 2018. 6

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 8803–8812, 2018. 6

Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018. 36

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019. 23

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. 25

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019. 1

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 1, 2

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 2019. 2

Julian Chibane, Aymen Mir, and Gerard Pons-Moll. Neural unsigned distance fields for implicit function learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, December 2020. 1, 27, 28

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *Proceedings of the 37th International Conference on Machine Learning*, 2020. 6

Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. In *Proceedings of Machine Learning and Systems 2020*, pages 3569–3579, 2020. 1, 3

Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *European Conference on Computer Vision (ECCV)*, 2020. 1

Sidak Pal Singh and Martin Jaggi. Model fusion via optimal transport. *Advances in Neural Information Processing Systems*, 33, 2020. 4

Vincent Sitzmann, Eric R. Chan, Richard Tucker, Noah Snavely, and Gordon Wetzstein. Metasdf: Meta-learning signed distance functions. In *Proc. NeurIPS*, 2020a. 3

Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Proc. NeurIPS*, 2020b. 15, 20

Norman Tatro, Pin-Yu Chen, Payel Das, Igor Melnyk, Prasanna Sattigeri, and Rongjie Lai. Optimizing mode connectivity via neuron alignment. *Advances in Neural Information Processing Systems*, 33, 2020. 4

Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3D shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 1

Samuel K. Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git re-basin: Merging models modulo permutation symmetries, 2022, arXiv:2209.04836. 4, 5, 6, 11, 13

Emilien Dupont, Hyunjik Kim, S. M. Ali Eslami, Danilo Jimenez Rezende, and Dan Rosenbaum. From data to functa: Your data point is a function and you can treat it like one. In *39th International Conference on Machine Learning (ICML)*, 2022. 3

Rahim Entezari, Hanie Sedghi, Olga Saukh, and Behnam Neyshabur. The role of permutation invariance in linear mode connectivity of neural networks. In *International Conference on Learning Representations*, 2022. 4, 5, 10

Shi-Min Hu, Zheng-Ning Liu, Meng-Hao Guo, Junxiong Cai, Jiahui Huang, Tai-Jiang Mu, and Ralph R. Martin. Subdivision-based mesh convolution networks. *ACM Trans. Graph.*, 41(3):25:1–25:16, 2022. 2

Chang Liu, Chenfei Lou, Runzhong Wang, Alan Yuhan Xi, Li Shen, and Junchi Yan. Deep neural network fusion via graph matching with applications to model ensemble and federated learning. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pages 13857–13869, 17–23 Jul 2022. 4

Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. Neural fields in visual computing and beyond. *Computer Graphics Forum*, 2022. 1

Luca De Luigi, Adriano Cardace, Riccardo Spezialetti, Pierluigi Zama Ramirez, Samuele Salti, and Luigi Di Stefano. Deep learning on implicit neural representations of shapes. In *International Conference on Learning Representations (ICLR)*, 2023. 2, 3, 15, 24, 27, 28

Appendix A

Implementation and hardware

General settings Our experiments were mainly implemented with the PyTorch library and performed on a machine with Intel Core i7-7700K CPU and a single NVIDIA GeForce RTX 2080 Ti GPU. The linear assignment problems were solved via the SciPy library, and the point cloud visualizations were made with Open3D (Zhou et al., 2018).

Activation matching The forward pass required by ACTIVATION-MATCHING was computed on a batch of 125K points (i.e. the most we could fit in our GPU): 62500 close to the surface, 50K at a medium-far distance from the surface, 6250 far from the surface, and 6250 scattered uniformly in the volume.

Timings Both ACTIVATION-MATCHING and WEIGHT-MATCHING run in a few seconds (on our GPU and CPU, respectively). HYPER-MATCHING runs on our GPU in ~2 minutes.