universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Numerical Methods for the Injectivity of ReLU-Layers

verfasst von | submitted by

Hannah Eckert BSc

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2024

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 645
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Data Science
Betreut von | Supervisor:	Mag. Dr. Peter Balazs Privatdoz.

Acknowledgements

Firstly, I want to express my deep gratitude to my supervisor Prof. Peter Balazs. Apart from supporting me in the work for this thesis, Peter integrated me at the Acoustics Research Institute and supported my participation in two conferences. Thank you for opening these doors for me! Furthermore, I want to thank my co-supervisor Daniel Haider for the countless hours he spent with me working on this thesis. Thank you for being the best mentor and friend.

Lastly, I want to thank my family for supporting me throughout my entire studies and my partner Waldemar for listening to me talk about ReLU-layers for a whole year.

Abstract

Injectivity is a desirable property for neural network layers as it ensures invertibility. Invertibility is advantageous because it opens possibilities for explainability, by allowing to trace back the decision-making process. This capability can be crucial for gaining an understanding of the entire neural network. A common type of neural network layer is the ReLU-layer. ReLU-layers are neural network layers where the activation function used is the Rectified Linear Unit (ReLU), defined as $\mathrm{ReLU}(s) = \max(0, s)$. The ReLU-function is applied component-wise to the affine linear function $Wx - \alpha$ with weight matrix $W \in \mathbb{R}^{m \times n}$, bias vector $\alpha \in \mathbb{R}^m$ and data point $x \in \mathbb{R}^n$. The injectivity of ReLU-layers has been investigated theoretically [HBE23] [MBB+23] [PKL+22] but the practical verification of this property has not been fully solved so far. In a recent paper [HEB24] a frame-theoretic condition to ensure injectivity of ReLU-layers on some open or convex set was introduced. The proposed condition is formulated in terms of an upper bound α^*, such that any ReLU-layer with bias $\alpha \leq \alpha^*$ (component-wise) is injective. Since, the exact calculation of α^* is computationally not feasible in high dimensions, the first goal of this work is to provide a probabilistic algorithm that approximates α^*. The algorithm is based on a Monte Carlo sampling approach and ensures "point-wise" invertibility for each uniformly drawn sample on the closed ball of $\mathbb{R}^n$. As a further step to make the algorithm more efficient, we perform data-driven sampling instead of sampling uniformly. We propose two approaches to implement this: 1) Augmenting the train data by adding Gaussian noise and using it as random samples for the Monte Carlo sampling and 2) Using kernel density estimation. The ultimate goal of the data-driven sampling is that the estimated upper bound has a higher probability of ensuring injectivity for samples that lie close (in the Euclidean distance) to the samples in the train data. The second goal of this work is to use the derived algorithm to study the injectivity behavior of a ReLU-layer during the training of a deep neural network in different settings. As a last step, we enforce injectivity throughout the training process and assess the effects on performance and stability, evaluating the costs associated with making ReLU-layers invertible.

Kurzfassung

Injektivität ist eine erstrebenswerte Eigenschaft für neuronale Netzwerkschichten, da sie Invertierbarkeit sicherstellt, was das neuronale Netz interpretierbarer macht, weil sie es ermöglicht, den Entscheidungsprozess nachzuvollziehen. ReLU-Schichten sind weit verbreitete Bausteine neuronaler Netzwerke und verwenden die die ReLU-Funktion als Aktivierungsfunktion. Die ReLU-Funktion ist als $\mathrm{ReLU}(s) = \max(0, s)$ definiert, wobei sie komponentenweise auf die lineare Funktion $Wx - \alpha$ angewendet wird und $W \in \mathbb{R}^{m \times n}$ die Gewichtsmatrix, $\alpha \in \mathbb{R}^m$ der Bias-Vektor und $x \in \mathbb{R}^n$ der Datenpunkt ist. Bis heute ist die praktische Verifikation der Injektivität von ReLU-Schichten nicht vollständig gelöst. In einer kürzlich erschienenen Arbeit haben meine Betreuer [HBE23] eine frametheoretische Bedingung zur Sicherstellung der Injektivität von ReLU-Schichten auf offenen oder konvexen Mengen eingeführt. Die Bedingung ist eine obere Schranke α^* für den Bias-Vektor α, sodass die zugehörige ReLU-Schicht injektiv ist. Allerdings ist die exakte Berechnung der vorgeschlagenen oberen Schranke α^* in hohen Dimensionen rechnerisch nicht durchführbar. Das erste Ziel dieser Arbeit ist es, einen probabilistischen Algorithmus bereitzustellen, der die obere Schranke α^* approximiert. Die Idee basiert auf einem Monte-Carlo-Sampling-Ansatz und der Sicherstellung der punktweisen Injektivität für jede gezogene Stichprobe. Als nächster Schritt zur Verbesserung des Algorithmus werden wir datengesteuertes Sampling anstelle von uniformen Sampling durchführen. Wir schlagen zwei Ansätze zur Implementierung vor: 1) Erweiterung der Trainingsdaten durch Hinzufügen von Gaußschem Rauschen und Verwendung der erweiterten Trainingsdaten als Zufallsstichproben für das Monte-Carlo-Sampling und 2) Verwendung von Kerndichteschätzung. Das ultimative Ziel ist, dass die geschätzte obere Schranke eine höhere Wahrscheinlichkeit hat, Injektivität für Stichproben sicherzustellen, die nahe (in der euklidischen Distanz) an Stichproben in den Trainingsdaten liegen. Das zweite Ziel dieser Arbeit ist es, den eingeführten Algorithmus zur approximativen numerischen Verifikation der Injektivität von ReLU-Schichten zu verwenden um das Injektivitätsverhalten einer ReLU-Schicht während des Trainings eines neuronalen Netzwerks in verschiedenen Szenarien untersuchen. Im letzten Schritt erzwingen wir die Injektivität während des gesamten Trainingsprozesses und bewerten anschließend die Auswirkungen auf Leistung und Robustheit.

Contents

1. Motivation

Over the last decade, neural networks have become an integral part of modern machine learning. Although they are powerful and versatile tools several obstacles remain, particularly regarding their lack of interpretability [FXLW20] [SMV+19]. The black box nature of neural networks presents significant challenges in understanding how these systems arrive at their specific decisions, raising concerns about transparency, trust, and accountability in critical applications [GBW+19] [HKK+18]. Addressing these issues is crucial for the broader adoption and ethical deployment of neural network technologies.

One promising approach to enhance interpretability is the development of invertible neural networks (INNs), which allow the reconstruction of inputs from outputs. This property enables INNs to function as both discriminative and generative models within a single set of parameters. That is because the inverse of an invertible discriminative neural network can be used to generate data samples from samples of the output space. The most prominent example of using INNs for data generation is the concept of normalized flows [PNR+21]. There, an INN is used to transform a complicated data distribution into a Gaussian distribution by a set of invertible transformations. The inverse of these transformations can then be used to generate data samples from the Gaussian distribution. The generative ability of an INN is suitable to interpret the model's decision-making process by examining the samples it generates based on its predictions. More specifically, we may systematically perturbate single features of the intermediate representations or the output, mapping them back to the data space, and interpreting the reconstructed input corresponding to the perturbed features. This enables a better understanding of the results. Therefore, invertible neural networks can enhance trust and reliability in neural networks, making them more suitable for sensitive and high-stakes environments. Another benefit is that the memory requirements for the computationally expensive backpropagation can be reduced because INNs don't need to store the results of intermediate activations during training since these values can be computed from the output [GRUG17].

The invertibility of a neural network is naturally linked to the invertibility of its fundamental units, the individual layers. Note that the invertibility of all individual layers of the network is a sufficient but not necessary condition for the invertibility of the whole network. The k-th layer $f^{(k)} : \mathbb{R}^{n^{(k)}} \to \mathbb{R}^{m^{(k)}}$ of a neural network is defined by its nonlinear activation function $\sigma^{(k)} : \mathbb{R}^{m^{(k)}} \to \mathbb{R}^{m^{(k)}}$ its weight matrix $W^{(k)} \in \mathbb{R}^{m^{(k)} \times n^{(k)}}$ and its bias vector $\alpha^{(k)} \in \mathbb{R}^{m^{(k)}}$ as

$$f^{(k)} : x \mapsto \sigma^{(k)}(W^{(k)}x - \alpha^{(k)}).$$

A deep neural network in its simplest form is the composition $f^{(l)}(..f^{(1)}(x))$ of $l \in \mathbb{N}$ such layers. A layer of this form is usually not invertible. Likewise, neural networks are generally not invertible.

1. Motivation

One approach to the invertibility of neural network layers is through their injectivity, as injectivity ensures invertibility on their range. A function f is injective on some set K, if for all $x, x' \in K$ it holds that if $f(x) = f(x')$ then $x = x'$. If a function is not injective, it can't be inverted because multiple inputs could lead to the same output, making it impossible to determine which input corresponds to a given output. On the other hand, every injective function is invertible on its range, i.e. there exists a unique inverse for each output.

In this work, we focus on the injectivity of ReLU-layers, which are neural network layers with the widely used ReLU activation function [NH10], defined by:

$$\begin{aligned} \mathrm{ReLU} : \mathbb{R}^n &\to \mathbb{R}^m \\ x &\mapsto \begin{pmatrix} \max\{x_1, 0\} \\ \vdots \\ \max\{x_m, 0\} \end{pmatrix}. \end{aligned}$$

The injectivity of ReLU-layers has been investigated theoretically [HBE23] [MBB+23] [PKL+22] but the practical verification of this property has not been fully solved so far. In a newly published study, Froese et al. [FGS24] demonstrated that the computational complexity of determining the injectivity of a ReLU-layer is in the class coNP, meaning that a 'no' answer can only be verified in polynomial time. To address this issue, we adopt an approximate approach based on a recently proposed frame theoretic condition for the injectivity of ReLU-layers [HEB24]. Explicitly checking the condition proposed in [HEB24] requires calculating the convex hull of the row vectors of the weight matrix W. This work is dedicated to developing a sampling-based algorithm to approximately check this condition for injectivity to make it feasible for high dimensional scenarios. We also show how to use this algorithm to promote the injectivity of a ReLU-layer and computationally analyze its impact on the performance of the neural network, in common baseline classification tasks.

2. Literature Review

The field of invertible neural networks has been considerably researched and studied [PNR+21] [DKB14] [GRUG17]. An early version of an invertible neural network was introduced by Deco et al [DB95]. Their ideas were used ten years later by Baird et al. [BSI05] who proposed a bijective neural network architecture for learning probability density functions from example vectors, which could also be used for data generation. This concept builds the basis of normalized flows, which is the most renowned example of an invertible neural network architecture today. Originally, normalized flows were introduced by Tabak et al. [TVE10]. Since then it has been generalized [TT13] and reworked many times [RM15] [DSDB16] [KD18] [PNR+21]. The basic concept of normalized flows is to transform a complex distribution into a Gaussian distribution through a sequence of invertible transformations. But also besides normalized flows, the field of invertible neural networks has been explored from many different perspectives. Another well-known approach was introduced by Ardizzone et al. [AKW+18], who developed an invertible neural network architecture for approximating physical processes. Several alternative architectures are available for use in a variety of contexts where invertibility is required [GRUG17] [JSO18] [BGC+19] [KW19].

As mentioned in Chapter 1, the invertibility of a neural network is closely linked to the injectivity of the individual layers of the network. Different types of layers, distinguished by their activation functions, can exhibit varying behaviors in terms of injectivity. Among various activation functions, the Rectified Linear Unit (ReLU) [NH10] is widely used due to its simplicity and effectiveness. However, the injectivity of ReLU-layers has only recently been addressed in the literature. One of the first to address this topic was Puthawala et. al. [PKL+22], who, among other settings, considered layers, with a weight matrices $W \in \mathbb{R}^{m\times n}$ with i.i.d. random Gaussian entries, a ReLU activation function, and no bias vector. They assumed the asymptotical setting where $n, m \to \infty$. They established a sufficient but not necessary condition for the redundancy $r = \frac{m}{n}$, showing that for $r > 9$ the ReLU-layer is asymptotically injective. On the other hand, they found that for $r < 3.3$ the ReLU-layer is asymptotically not injective. Therefore, the injectivity behavior is unknown for

$$3.3 \leq r \leq 9.$$

These bounds were improved to

$$5.32 \leq r \leq 7.65$$

by Maillard et. al. [MBB+23], using methods from statistical physics. Numerically, they were able to show that a ReLU-layer with random weight matrix with Gaussian i.i.d.

entries is asymptotically injective if $r < \hat{r}$ and asymptotically not injective if $r > \hat{r}$ with

$$\hat{r} \in (6.6979, 6.6982).$$

However, a formal proof is pending.

Vallin, et al. [VLL23] considered a ReLU-layer, where $n \geq m$ holds for the input and output dimension n, m of the layer. They provided a geometric interpretation of such a ReLU-layer as a projection onto a polyhedral cone, followed by an affine transformation. This perspective was used to formulate explicit expressions for the images and preimages of the layer, where the preimage of a set K under the ReLU-layer is the set of all elements in the domain $\mathbb{R}^n$ that $\mathrm{ReLU}_{W,\alpha}$ maps into K. It is possible to use the preimages of a ReLU-layer to derive an equivalent formulation of its injectivity. Specifically, a ReLU-layer is injective if, for all sets containing only one element, their preimages under the layer also contain only one element. However, as they only considered cases where for the redundancy of the layer $r \leq 1$ holds, i.e. for layers that are never injective, their work does not immediately imply anything new about the injectivity of ReLU-layers.

The most recent work on the topic is a study on the computational complexity of deciding injectivity [FGS24]. They proved coNP-completeness of deciding the injectivity of a ReLU-layer, where coNP means that a 'no' answer can only be verified in polynomial time and coNP-complete means that every other problem in coNP can be reduced to this problem using a polynomial time reduction. This demonstrates the need for an approximate approach to determining injectivity, as done in this work.

The primary foundation of this work is Haider et al.'s study [HBE23] of the injectivity of ReLU-layers. Their approach is based on frame theory [DS52] [DGM86] [KC+08], which is a subfield of harmonic analysis that deals with stable and invertible representations of functions. The motivation for this approach comes from the fact that the injectivity of non-linear functions applied on linear mappings has already been investigated from the perspective of frame theory before[AFG+24] [BCE06]. An example of such a setting can be found in the work of Alharbi et al. [AFG+24] on the injectivity of the clipping function. The clipping function truncates any measurement that falls below or exceeds a specified saturation threshold. The problem of recovering a signal from a linear transformation and subsequent clipping is referred to as saturation recovery. The authors employ frame theory to identify the circumstances under which saturation recovery is possible and to develop a reconstruction algorithm. Their work is inspired by Balan et al. [BCE06], who used a similar approach to address the problem of phase retrieval, i.e. reconstructing signals from the absolute values of frame coefficients.

Haider et al. [HBE23] used frame theory to derive an injectivity condition on the bias vector α of a ReLU-layer. The introduced condition involves an upper bound α^*, so that a ReLU-layer with bias vector α is injective if $\alpha \leq \alpha^*$ (component-wise) holds [HBE23]. We discuss their condition in detail in Section 3. However, the exact verification of their condition requires the calculation of the convex hull of the row vectors of the weight matrix W. This is a computationally infeasible task in high dimensions. In their follow-up work, the authors propose a method for obtaining α^* differently, allowing it to be approached from a numerical side [HEB24]. This work builds on their approach

and introduces a computationally feasible algorithm, which is capable of varifying the proposed condition in an approximate fashion also in high dimensions. Furthermore, the algorithm is used to better understand the injectivity behavior of ReLU-layers in practice and to enforce it during neural network training.

3. Theoretical Injectivity of ReLU-layers

In this chapter, we introduce frame theory [DS52] [DGM86] [KC+08], a theoretical backbone for neural network layers. We will constrain ourselves to neural network layers with ReLU activation function, which we call *ReLU-layers* and formulate a mathematical condition for the injectivity of ReLU-layers, which will be used for the numerical verification of this property.

3.1. Introduction to Frame Theory

Definition 3.1.1 (Frames)**.** We say that a matrix $W \in \mathbb{R}^{m\times n}$ with row vectors $w_i \in \mathbb{R}^n$ for $i = 1, .., m$ induces a frame for $\mathbb{R}^n$ if there are constants $0 < A \leq B < \infty$ such that the inequality

$$A\|x\|^2 \leq \sum_{i=1}^{m} |\langle x, w_i\rangle|^2 \leq B\|x\|^2 \tag{3.1}$$

holds for all $x \in \mathbb{R}^n$. The vectors w_i are called the *frame elements* and the set of vectors $\{w_i\}_{i=1,..,m}$ is called a *frame*.

In $\mathbb{R}^n$, a frame is equivalent to a spanning set [CK12], hence $m \geq n$ has to hold. What the perspective through frames offers here is that the constants A, B determine the numerical stability of the representation of x in terms of the inner products $(\langle x, w_i\rangle)_{i=1,...,m}$. We will later use them to study the stability of ReLU-layers.

Definition 3.1.2 (Analysis Operator)**.** Applying the matrix $W \in \mathbb{R}^{m\times n}$ to a vector $x \in \mathbb{R}^n$ results in the *frame coefficients*

$$Wx = (\langle x, w_i\rangle)_{i=1,..,m}.$$

In this context, we refer to W as the *analysis operator*.

The following lemma states that the frame property holds with probability one for random tall matrices, i.e. random matrices where the dimension of the column vectors is greater than that of the row vectors. This will prove to be crucial for studying upsampling ReLU-layers, i.e. ReLU-layers with tall weight matrices, later in this thesis.

Lemma 3.1.1 ([BD09])*.* For a matrix $W \in \mathbb{R}^{m\times n}$ that maps into a higher dimensional space ($m \geq n$) with entries that are i.i.d. (independent and identically distributed) samples from an absolutely continuous probability distribution the frame property holds true with probability one.

The final frame-theoretic property that must be introduced to apply frame theory to ReLU-layers is the connection between the frame property and the injectivity of the analysis operator.

Lemma 3.1.2. The analysis operator $W : \mathbb{R}^n \to \mathbb{R}^m$ is injective if and only if the matrix $W \in \mathbb{R}^{m\times n}$ induces a frame.

In the next section, we will see frame theory in the context of ReLU-layers, and interpret the above lemma as the frame-inducing property of a weight matrix $W \in \mathbb{R}^{m\times n}$ being characterizing for the injectivity of the corresponding linear layer.

3.2. Frame Theory and ReLU-layers

To build the bridge from frame theory to neural network layers we shall assume that our ReLU-layers are *upsampling layers*, i.e. layers where $m \geq n$ holds for the input dimension n and the output dimension m. We further assume i.i.d. randomly initialized weights. Using Lemma 3.1.1 we know that for the weight matrix of such layers, the frame property holds true with probability one. With this in mind, we can define ReLU-layers in the spirit of the analysis operator.

Definition 3.2.1 (ReLU-layer)**.** Given a weight matrix $W \in \mathbb{R}^{m\times n}$ with row vectors $w_i \in \mathbb{R}^n$ for $i = 1, .., m$ and a bias vector $\alpha \in \mathbb{R}^m$ we call the function defined by:

$$\mathrm{ReLU}_{W,\alpha} : \mathbb{R}^n \to \mathbb{R}^m$$
$$x \mapsto \begin{pmatrix} \max\{x_1, 0\} \\ \vdots \\ \max\{x_m, 0\} \end{pmatrix}$$

a *ReLU-layer* with *weight matrix* W and *bias vector* α.

We may express the upsampling property in terms of the redundancy of the layer as follows.

Definition 3.2.2 (Redundancy)**.** Given a ReLU-layer $\mathrm{ReLU}_{W,\alpha} : \mathbb{R}^n \to \mathbb{R}^m$, we define its *redundancy* r as

$$r = \frac{m}{n}.$$

For upsampling layers, $r \geq 1$ holds.

To get an intuitive understanding of the injectivity of a ReLU-layer observe that we can rewrite the layer component-wise as

$$\mathrm{ReLU}_{W,\alpha}(x)_i = \begin{cases} \langle x, w_i\rangle - \alpha_i & \text{if } \langle x, w_i\rangle \geq \alpha_i \\ 0 & \text{if } \langle x, w_i\rangle < \alpha_i. \end{cases}$$

Since the analysis operator is injective if W induces a frame it appears natural that the injectivity of $\mathrm{ReLU}_{W,\alpha}$ depends on the set of entries of the input vector x_i that are not projected to zero by the ReLU function, i.e., for which $\langle x, w_i\rangle \geq \alpha_i$ and in particular if these set form a frame [HBE23]. This motivates the following definitions.

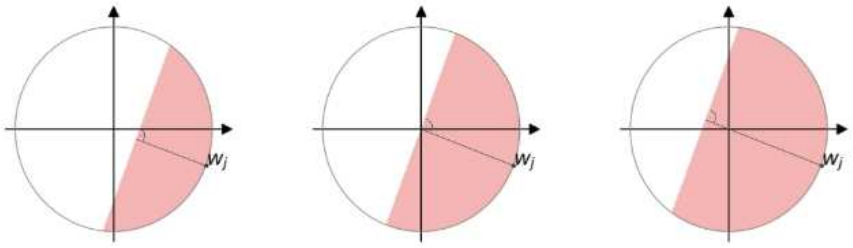

Figure 3.1.: The illustration shows the half-spaces defined by the affine hyperplanes $\langle x, w_j \rangle = \alpha_j$. The red area represents the set of points $x \in \mathbb{B}^2$ for which the shown frame element w_j is active for $\alpha_j = 0.25$ (left), $\alpha_j = 0$ (mid), and $\alpha_j = -0.25$ (right).

Definition 3.2.3 (Active Frame Elements)**.** We call w_i *active* for x and α if $\langle x, w_i \rangle \geq \alpha_i$. Further, we define the index set associated to all active frame elements for x and α as

$$I_x^\alpha := \{i \in I : \langle x, w_i \rangle \geq \alpha_i\}.$$

To gain a deeper understanding of the concept of active frame elements, it is helpful to consider a geometric interpretation. From a geometric perspective, the set of points for which a frame element w_i is active can be conceptualized as a half-space of $\mathbb{R}^n$, defined by the separating affine hyperplane $\langle x, w_i \rangle = \alpha_i$. For $\alpha_i = 0$ the hyperplane passes through the origin. For $\alpha_i \leq 0$, the origin lies in the set of points for which w_i is active, and conversely, for $\alpha_i > 0$, the origin lies in the set of points for which w_i is not active. Figure 3.1 illustrates these halfspaces restricted to the two-dimensional ball $\mathbb{B}^2 = \{x \in \mathbb{R}^2 : ||x|| \leq 1\}$.

The notion of active frame elements leads us to our next property, which will be essential for characterizing injectivity. We consider frames which satisfy that for all $x \in K$ the active frame elements form a frame.

Definition 3.2.4 (α-Rectifiying Frames)**.** We call a frame-inducing matrix $W \in \mathbb{R}^{m \times n}$ *α-rectifiying* on $K \subseteq \mathbb{R}^n$ if for all $x \in K$ the active frame elements for x and α form a frame. In particular, we call W α-rectifying for x if it is α-rectifying on the set $\{x\}$.

The geometric interpretation of the α-rectifying property is that each point $x \in K$ lies in at least n active half-spaces $\{x : w_j \text{ is active for } x \text{ and } \alpha\}$. Figure 3.2 illustrates how the set for which a frame-inducing matrix W is α-rectifying naturally increases for smaller values of α.

The definition of the α-rectifying property allows us to derive a frame theoretic condition for the injectivity of a ReLU-layer on $K \subseteq \mathbb{R}^n$:

Theorem 3.2.1 ([HEB24]). If $W \in \mathbb{R}^{m \times n}$ is α-rectifying on $K \subseteq \mathbb{R}^n$ and K is open or convex then $\text{ReLU}_{W,\beta}$ is invertible on K if $\beta \leq \alpha$ holds in a point-wise sense.

This theorem is the main result upon which the algorithm introduced in Chapter 4 is built. To provide a deeper understanding of the relationship between the α-rectifying

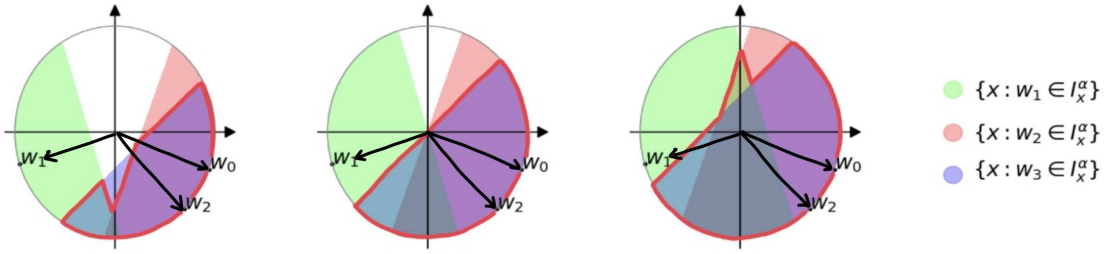

Figure 3.2.: The illustration shows the maximal set $K \subseteq \mathbb{B}^2$ (outlined in red) for which the frame inducing matrix $W = (w_1, w_2, w_3)^T$ is α-rectifying for $\alpha = (0.25, 0.25, 0.25)^T$ (left), $\alpha_j = \vec{0}$ (middle), and $\alpha_j = (-0.25, -0.25, -0.25)^T$ (right). The set for which W is α-rectifying increases for smaller values of α.

property and injectivity provided by the theorem, we offer a brief overview of the proof presented by Haider et al. [HEB24] by presenting the case $\beta = \alpha$. For the other case, we refer to the original work.

Let $W \in \mathbb{R}^{m \times n}$ be a frame-inducing matrix that is α-rectifiying on some open or convex set $K \subseteq \mathbb{R}^n$. Let further

$$\mathrm{ReLU}_{W,\alpha}(x) = \mathrm{ReLU}_{W,\alpha}(y) \tag{3.2}$$

for some $x, y \in K$. Equation 3.2 implies that

$$\langle x, w_i \rangle = \langle y, w_i \rangle \tag{3.3}$$

holds for all $i \in I_x^\alpha$. Since W is α-rectifying we know that the set of elements that are active for x form a frame. From Lemma 3.1.2, we can conclude the injectivity of the analysis operator of the set of active elements for x. It follows that $x = y$, and therefore injectivity of the ReLU-layer.

This allows us to approach the invertibility of ReLU-layers by verifying that the weight matrix W is α-rectifying on $K \subseteq \mathbb{R}^n$. This is still not trivial, so we will make an additional assumption that further simplifies the problem.

Definition 3.2.5 (Full Spark Frames [ACM12])**.** A frame-inducing matrix $W \in \mathbb{R}^{m \times n}$ is called *full spark* if each size n sub-collection of frame elements forms a frame for $\mathbb{R}^n$.

Lemma 3.1.1 directly implies that matrices with entries that are i.i.d. samples from an absolutely continuous probability distribution are full spark with probability one. This is particularly important for neural network applications, as weight matrices are commonly initialized in this manner. Although not proven, it is reasonable to assume that the full spark property is also preserved during gradient steps in training. Consequently, a full spark assumption is very natural in the context of neural network training and will therefore be made in our injectivity examinations. With this assumption the verification of the α-rectifying property can be reduced to a counting argument.

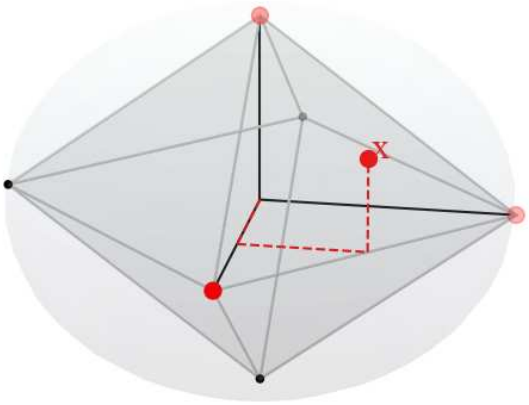

Figure 3.3.: The octahedron in the ball, which vertices form a frame. For each point on the ball x there are three active frame elements. These are given by the vertices of the octahedron where the angle between x and the vertice is smaller or equal 90° (red).

Lemma 3.2.2. Let $W \in \mathbb{R}^{m\times n}$ be frame-inducing and full spark and $K \subseteq \mathbb{R}^n$. W is α-rectifying on K if and only if the cardinality of the set of active frame elements satisfies

$$|I_x^\alpha| \geq n \tag{3.4}$$

for all $x \in K$.

Example 3.2.1. Consider the matrix that contains the vertices of an octahedron in $\mathbb{R}^3$ as row vectors, given by

$$W = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix}.$$

The vertices lie on the unit ball $\mathbb{B}^3$. We choose $\alpha = \vec{0}$. The vertices of the octahedron form a spanning set for $\mathbb{R}^n$ and thus form a frame. For each point $x \in \mathbb{B}^n$ there are three vertices $w_{i_1}, w_{i_2}, w_{i_3}$ of the octahedron where the angle between x and w_i is less than or equal to 90° and thus the inner product satisfies the inequality $\langle x, w_i\rangle \geq 0$ for $i = i_1, i_2, i_3$ (Figure 3.3). So for every $x \in \mathbb{B}^3$ and $\alpha = \vec{0}$ there are three active frame elements in the octahedron. It is evident that these active frame elements always form a spanning set of $\mathbb{R}^3$ and thus, a frame. We conclude that the octahedron is $\vec{0}$-rectifying on $K = \mathbb{B}^3$. we can use Theorem 3.2.1 to conclude that the associated ReLU-layer $\mathrm{ReLU}_{W,\vec{0}}$ is injective.

In the presented example it is relatively simple to show the α-rectifying property. However, in general, this is not the case. Deciding the injectivity of a ReLU-layer has even been recently proven to belong to the class coNP [FGS24]. Consequently, it is essential to

adopt an approximation approach to address this issue, which will be discussed in greater detail in the next chapter.

The results presented in this chapter enable us to reformulate the question of the injectivity of ReLU layers in frame-theoretic terms. The injectivity of a ReLU-layer $\mathrm{ReLU}_{W,\alpha}$ with weight matrix $W \in \mathbb{R}^{m\times n}$ and bias vector $\alpha \in \mathbb{R}^m$ on a open or convex set $K \subseteq \mathbb{R}^n$ is equivalent to the frame inducing matrix W being α-rectifying on K. To verify the injectivity of the ReLU-layer $\mathrm{ReLU}_{W,\alpha}$ on K, it is possible to solve the equivalent problem of verifying that the weight matrix W is α-rectifying on K.

4. Monte Carlo Bias Estimation

The main goal of this thesis is to find a bias $\alpha^*(K)$ for which a given weight matrix $W \in \mathbb{R}^{m\times n}$ is $\alpha^*(K)$-rectifying for all x in the open or convex set K. On the sphere, the calculation of such a bias $\alpha^*(K)$ is possible [HBE23]. However, this approach becomes computationally infeasible in high dimensions. That is because the calculation involves determining the convex hull of the row vectors of W, which is expensive from a computational standpoint. To circumvent this issue, another construction was presented in [HEB24]. This construction yields a method called Monte Carlo Bias Estimation (**MCBE**) that uses a finite subset $X \subset K$ to calculate the bias $\alpha^*(X)$ so that W is $\alpha^*(X)$-rectifying for all $x \in X$. $\alpha^*(X)$ can then be used as a approximation of $\alpha^*(K)$, making it suitable to verify injectivity in practical applications. The results of the numerical experiments in Chapter 7 will demonstrate that **MCBE** achieves invertibility on a high percentage of unseen samples $x \notin X$. Since the quality of the upper bound depends on how dense the data points $x \in X$ lie in the considered set K, we will discuss an error approximation in Section 4.4.

4.1. Motivation and Overview

For a given set of points $X \subset \mathbb{R}^n$ in the output of **MCBE** is a bias vector $\alpha^*(X)$ such that W is α-rectifying on X for all $\alpha \leq \alpha^*(X)$. Here, "$\leq$" is meant in an entry-wise sense, i.e., $\alpha_i \leq \alpha_i^*$ for all $i = 1, \ldots, m$. To obtain , we first compute point-wise maximal biases $\alpha(x_i)$ that ensure that W is $\alpha(x_i)$-rectifying for x_i.

Definition 4.1.1 (Point-wise Maximal Bias)**.** Let $W \in \mathbb{R}^{m\times n}$ and $x \in \mathbb{R}^n$. The *point-wise maximal bias* $\alpha(x)$ is defined as:

$$\alpha(x)_j := \begin{cases} \langle x, w_j \rangle & \text{for } j \in J_x \\ \infty & \text{else ,} \end{cases}$$

where J_x is the index set associated with the n largest entries of the frame coefficients.

Remark. Strictly speaking, the index set J_x is not uniquely defined for all $x \in \mathbb{R}^n$. For example let $x = \vec{0}$ and $m > n$. Then $Wx = \vec{0}$ and therefore J_x is not unique. However, in such a case **MCBE** works with any choice of J_x.

The following theorem states that with this definition of the point-wise maximal bias, $W \in \mathbb{R}^{m\times n}$ is α-rectifying for x if $\alpha \leq \alpha(x)$.

Theorem 4.1.1 (Point-wise Maximal Bias). Let $W \in \mathbb{R}^{m\times n}$ be full spark, and let $\alpha(x)$ for $x \in \mathbb{R}^n$ be the point-wise maximal bias. Then W is α-rectifying for x if

$$\alpha \leq \alpha(x) \tag{4.1}$$

in a pointwise sense.

Proof. Let $\alpha \leq \alpha(x)$, then in particular,

$$\alpha_j \leq \alpha(x)_j = \langle x, w_j \rangle \text{ for all } j \in J_x. \tag{4.2}$$

Since J_x contains n elements, Equation 4.2 implies that n frame elements are active. Therefore W is α-rectifying for x by the full spark assumption (see Definition 3.2.5). □

This allows us to check the α-rectifying property in a point-wise manner by calculating the point-wise maximal bias and verifying Equation 4.1. This is demonstrated in the following example.

Example 4.1.1 (Pointwise Maximal Bias). Let us recall the octahedron frame W as defined in Example 3.2.1. Let further the point x be defined as

$$x = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

The frame coefficients for x are

$$Wx = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \\ 1 \\ -1 \end{pmatrix}.$$

The index set associated with the $n = 3$ largest entries of the analysis operator is given as

$$J_x = \{1, 3, 5\},$$

and therefore the point-wise maximal bias for x is

$$\alpha(x) = \begin{pmatrix} \frac{1}{\sqrt{3}} \\ \infty \\ \frac{1}{\sqrt{3}} \\ \infty \\ \frac{1}{\sqrt{3}} \\ \infty \end{pmatrix}.$$

Using Theorem 4.1.1 we conclude that for any α for which $\alpha \leq \alpha(x)$ holds, the octahedron frame is α-rectifying for x.

For a given dataset $X \subseteq \mathbb{R}^n$ **MCBE** uses all point-wise maximal biases $\alpha(x_i) : x_i \in X$ to calculate the **MCBE**-bias $\alpha^*(X)$.

Definition 4.1.2 (**MCBE**-bias)**.** For $X \subseteq \mathbb{R}^n$ the ***MCBE**-bias* $\alpha^*(X)$ is given as

$$\alpha^*(X)_j = \min_{x_i \in X} \alpha(x_i)_j. \tag{4.3}$$

With this definition we obtaion that W is α-rectifying on X as long as $\alpha \leq \alpha^*(X)$.

Theorem 4.1.2. Let $W \in \mathbb{R}^{m \times n}$ be full spark, and let $\alpha^*(X)$ for $X \subseteq \mathbb{R}^n$ be the **MCBE** bias. Then W is α-rectifying on X if $\alpha \leq \alpha^*(X)$.

Proof. Let $\alpha \leq \alpha^*(X)$ then $\alpha \leq \alpha(x_i)$ for all $x_i \in X$. Consequently, W is α-rectifying on $\{x_i\}$ for all $x_i \in X$ and therefore also on X. □

The **MCBE**-bias $\alpha^*(X)$ is easy to calculate for finite sets $X \subseteq \mathbb{R}^n$. However, since Equation 4.3 minimizes a non-trivial, non-linear function over X through point-wise evaluation, it cannot be easily adapted for infinite sets. Our approach to address this issue is to approximate the **MCBE** bias $\alpha^*(K)$ for an infinite set $K \subseteq \mathbb{R}^n$ by $\alpha^*(X)$, where $X \subseteq \mathbb{R}^n$ contains finitely many samples from K. This method proves to be effective in practice, as we will demonstrate in our numerical experiments in Section 4.3. We formalize the resulting property of K in the following definition.

Definition 4.1.3 (**MCBE**-injectivity)**.** We say that a ReLU-layer with bias vector $\alpha \in \mathbb{R}^m$ is **MCBE**-injective on $K \in \mathbb{R}^n$ if

$$\alpha \leq \alpha^*(X)$$

holds in a point wise sense, where $\alpha^*(X)$ is the **MCBE**-bias calculated on a finite sampling set $X \subset K$.

Note that the meaning of **MCBE**-injectivity heavily relies on the selected sampling set X. In Section 4.3, we will conceptually connect **MCBE**-injectivity with injectivity on K.

We will demonstrate the simple calculation of the **MCBE**-bias for finite datasets through an example.

Example 4.1.2. Recall the octahedron frame W from Example 3.2.1. We consider the dataset $X = \{x_1, x_2, x_3\}$ consisting of the points

$$x_1 = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, x_2 = \frac{1}{\sqrt{3}} \begin{pmatrix} -1 \\ 1 \\ -1 \end{pmatrix}, x_3 = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}.$$

Calculating the point-wise biases as shown in Example 4.1.1 yields

$$\alpha(x_1) = \begin{pmatrix} \frac{1}{\sqrt{3}} \\ \infty \\ \frac{1}{\sqrt{3}} \\ \infty \\ \frac{1}{\sqrt{3}} \\ \infty \end{pmatrix}, \alpha(x_2) = \begin{pmatrix} \infty \\ \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \\ \infty \\ \infty \\ \frac{1}{\sqrt{3}} \end{pmatrix}, \alpha(x_3) = \begin{pmatrix} \frac{1}{\sqrt{3}} \\ \infty \\ \infty \\ \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \\ \infty \end{pmatrix}.$$

If we now take the point-wise minimum to get the **MCBE**-bias $\alpha^*(X)$, we get

$$\alpha^*(X) = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}.$$

Using Theorem 4.1.1 we can conclude that for any α for which $\alpha_i \leq \frac{1}{\sqrt{3}}$ holds for all $i = 1, .., m$, the octahedron frame is α-rectifying on the dataset X.

The introduced **MCBE** bias $\alpha^*(K)$ for a given set $K \subseteq \mathbb{R}^n$, combined with the concept of estimating it using a sampling set $X \subseteq K$, reduces the problem of checking the injectivity of a ReLU-layer on K to verifying whether the condition $\alpha \leq \alpha^*(X)$ holds point-wise for X. This criterion provides a practical and efficient way to ensure the injectivity of ReLU-layers with high probability. However, due to the approximate nature of the approach, we can never guarantee injectivity for infinite sets. We will introduce an error approximation for this method in Section 4.4.

4.2. Implementation

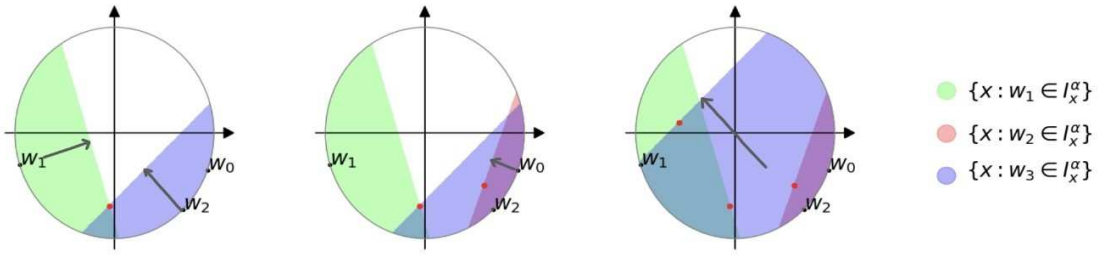

Figure 4.1.: Illustrations of the first three iterations of the Monte Carlo Bias Estimation on the two-dimensional ball $\mathbb{B}^2$ (left: iteration 1, middle: iteration 2, right: iteration 3). The red points represent the uniformly drawn samples, and the grey arrows represent the movement of the half spaces during the respective iterations.

We may implement the Monte Carlo Bias Estimation iteratively by initializing $\alpha^*(\{\})_j = \infty$ for $j = 1, .., m$ and updating $\alpha^*(\{x_1, .., x_{i-1}\})$ in the i-th iteration by uniformly drawing the sample x_i, calculating the set J_{x_i} and setting

$$\alpha^*(\{x_1, .., x_i\})_j = \min\{\alpha^*(\{x_1, .., x_{i-1}\})_j, \langle x_i, w_j \rangle\} \tag{4.4}$$

for $j \in J_{x_i}$ (Algorithm 1).

Geometrically, we can understand the i-th iteration as sampling a point $x_i \in K$ and then making the minimal update on $\alpha^*(\{x_1, .., x_{i-1}\})$ so that x_i in at least n of the

] **Input:** weight matrix W, number of iterations for which α has to stay the same for convergence s
Result: $\alpha^*(x_1, .., x_i)$
Initialize $\alpha^*(\{\})_j = \infty$ for $j = 1, .., m$;
$i = 0$;
while $(||\alpha^*(\{x_1, .., x_i\}) - \alpha^*(\{x_1, .., x_{i-s}\})|| > 0) \vee (i < s)$ **do**
  sample x_i from K;
  calculate J_{x_i};
  for $j \in J_{x_i}$ **do**
    $\alpha^*(\{x_1, .., x_i\})_j = \min\{\alpha^*(\{x_1, .., x_{i-1}\})_j, \langle x_i, w_j \rangle\}$;
  end
  $i = i + 1$
end

Algorithm 1: Monte Carlo Bias Estimation

half-spaces defined by $Wx = \alpha^*(\{x_1, .., x_i\})$. The first three iterations of this process are visualized on the two-dimensional ball in Figure 4.1.

The samples for **MCBE** are drawn uniformly at random for each iteration. However, we have included the option to initialize the algorithm by setting the frame elements $w_1, .., w_m$ as the first m samples if $w_i \in K$ for all $i = 1, .., m$. This is because we have found empirically that in some cases the samples drawn around the frame elements provide critical updates for $\alpha^*(X)$. These critical points are visualized in Figure 4.2, where blue points represent high values in $\alpha(x_i)$ and red points represent low values in $\alpha(x_i)$. Since $\alpha^*(X)$ is calculated as the point-wise minimum of $\alpha(x_i)$ for $x_i \in X$, the points where $\alpha(x_i)$ has low values provide critical updates for the **MCBE** algorithm. The points are color-coded in a way that represents the minimal value of the set J_{x_i}, i.e., the set of values that potentially provide point-wise updates for $\alpha^*(X)$. The examples in the top row of Figure 4.2 show how critical points are clustered around the frame elements (vertices) and the line segments (edges) between them. On the other hand, the example on the right in the bottom row shows how critical points also occur in areas without frame elements.

To determine the number of iterations for the algorithm, it is possible to ask the user to set a fixed number of iterations or to formulate a stopping criterion. In our implementation, we have incorporated both approaches, allowing the user to either specify a fixed iteration count or rely on a stopping criterion to determine when the algorithm should terminate dynamically. Our proposed stopping criterion is to set a number of iterations s for which the **MCBE** bias $\alpha^*(X)$ must remain without update to consider convergence. After convergence, the ReLU-layer with the given weight matrix W is **MCBE**-injective if $\alpha \leq \alpha^*(X)$.

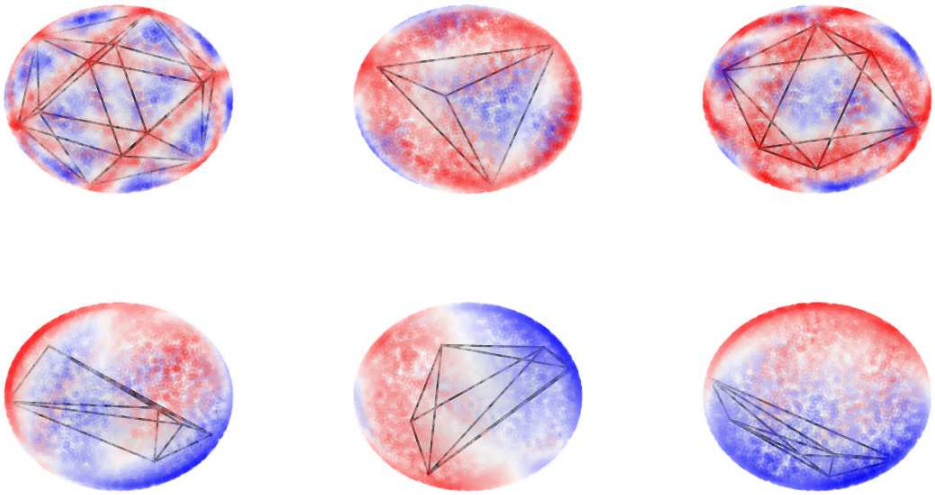

Figure 4.2.: Illustration of critical samples of the **MCBE** algorithm for different frames (icosahedron, tetrahedron, and octahedron in the top row and random frames in the bottom row). The sampling points are drawn uniformly on the unit sphere. Blue points represent high values in $\alpha(x_i)$, i.e., points where $\alpha^*(X)$ is unlikely to be updated, and red points represent low values in $\alpha(x_i)$, i.e., critical points where $\alpha(x_i)$ is more likely to be updated. The figure illustrates that critical updates are located around the frame elements (vertices). In the examples shown, this is particularly true for the non-random frames (top row). The random frame shown in the bottom right illustrates that this is not always the case.

4.3. Numerical Demonstration of Convergence

One way to conceptually connect the introduced **MCBE**-injectivity with injectivity on K is by observing the convergence of the **MCBE** algorithm. Recall that if the algorithm has stopped, we would only expect very small or no updates in the iterations following. If $\alpha^*(\{x_1, .., x_i\}) = \alpha^*(\{x_1, .., x_{i-1}\})$, we can interpret this as W being already $\alpha^*(\{x_1, .., x_{i-1}\})$-rectifying for the x_i. Consequently, if updates are rare or nonexistent on samples uniformly drawn from K, it suggests that W is $\alpha^*(X)$-rectifying for $x \in K$ with high probability. If we assume, theoretically, perfect convergence — meaning that updates are never made — then W is $\alpha^*(X)$-rectifying for all $x \in K$, which implies injectivity of the associated ReLU-layer on K.

We demonstrate the convergence behavior of **MCBE** on matrices with Gaussian i.i.d entries. We sample the m row vectors of the matrix W, the sample points needed for the **MCBE** iterations as well as a test set uniformly on the unit ball $\mathbb{B}^n$. For every iteration of the **MCBE** algorithm, we measure the proportion of samples x from the test set, for which W is $\alpha^*(X)$-rectifying for x. This proportion represents the number of test samples

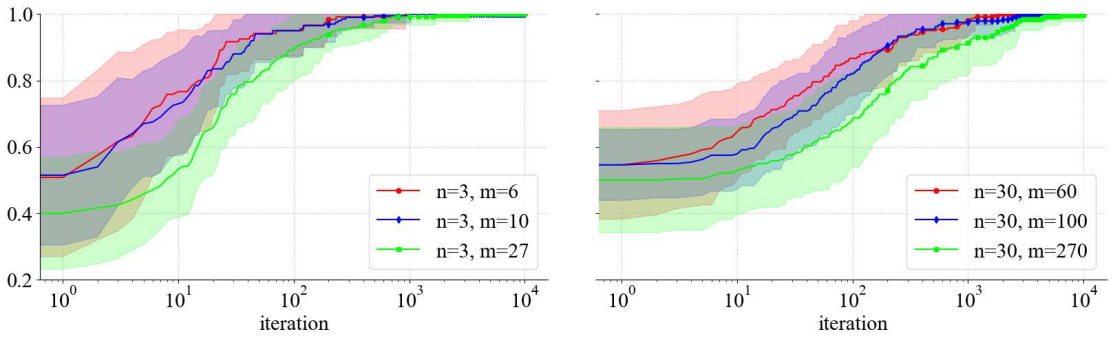

Figure 4.3.: Convergence behavior of the **MCBE** algorithm on a weight matrix with Gaussian i.i.d. entries. The plots show the proportion of unseen test samples per iteration for which **MCBE** would not make an update. Left: $n = 3$. Right: $n = 30$, both for redundancies 2, 3.3, and 9 respectively. The plots show the empirical mean and variance over 20 independent trials of the experiment. All tested examples demonstrate reliable convergence. However, it can be observed that convergence occurs generally faster in lower dimensions.

for which the algorithm would make no update. An equivalent perspective is that this proportion represents the number of test samples for which perfect reconstruction from the output $\mathrm{ReLU}_{W,\alpha^*(X)}$ is possible. We will focus on this perspective in Chapter 6. Figure 4.3 shows the empirical mean and variance over 20 independent trials of this procedure for different dimensions n and redundancies $r = \frac{m}{n}$. We observe that the **MCBE** algorithm generally converges faster in lower dimensions and with lower redundancies. Notably, all tested examples demonstrated reliable convergence.

4.4. Error Approximation

Since **MCBE** is a sampling-based algorithm, we are interested in an error approximation, i.e. in the difference between the **MCBE**-bias $\alpha^*(X)$ and the true maximal bias α_{true}, for which W is α_{true}-rectifying on K. The error depends on the approximation points $x_1, .., x_N \in K$, more specifically on their covering radius.

Definition 4.4.1 (Euclidean covering radius)**.** The *Euclidean covering radius* of some set K and points $x_1, .., x_N \in K$ is defined as

$$\rho(x_1, .., x_N; K) := \sup_{x \in K} \inf_{1 \leq i \leq N} ||x - x_i||.$$

For uniformly sampled points on the sphere, the Euclidean covering radius can be estimated asymptotically.

Theorem 4.4.1 (Euclidian covering radius [RS16])**.** Asymptotically in N, the Euclidian covering radius $\rho(x_1, .., x_N; K)$ for $x_1, .., x_N$ uniformly sampled on a sphere $\mathbb{S}_r^n = \{x \in$

$\mathbb{R}^n : ||x|| = r\}$ is given by

$$\rho(x_1, .., x_N; K) \asymp k(\frac{\log(N)}{N})^{\frac{1}{n}}$$

for some constant $k \in \mathbb{R}^n$. For the unit sphere $\mathbb{S}_1^n$, this coefficient is given by

$$k_{\mathbb{S}} = (\frac{(n+1)v_{n+1}}{v_n})^{\frac{1}{n}},$$

where v_n is the volume of the n-dimensional unit ball.

As we will see in the following theorem, the Euclidean covering radius is an upper bound on the difference between the **MCBE** bias α^* and the true maximal bias α_{true}.

Theorem 4.4.2 (Error Approximation **MCBE**). Let $W \in \mathbb{R}^{m\times n}$ be a full rank (and therefore frame-inducing) matrix, and let $K \subseteq \mathbb{R}^n$. Consider a set of points $X = \{x_1, .., x_N\} \subseteq K$ and let $\alpha^*(X)$ denote the **MCBE**-bias calculated from this points. Let further α_{true} be the true maximal bias for which W is α-rectifying on K. Let the error ϵ be defined as:

$$\epsilon = ||\alpha^*(X) - \alpha_{\text{true}}||^2$$

Then, the error ϵ is bounded from above by $\rho(x_1, .., x_N; K)$, i.e.,

$$\epsilon \leq m(\beta\rho(x_1, .., x_N; K))^2,$$

where

$$\beta = \max_{i=1,..,m}\{||w_i||\}.$$

Proof. Let $x \in K$. Then, there exists some $x_i \in X$ such that

$$||x - x_i|| \leq \rho(x_1, .., x_N; K).$$

Because of the **MCBE** algorithm, W is $\alpha^*(X)$-rectifying for x_i, i.e.

$$|I_{x_i}^{\alpha^*}| \geq n.$$

For all $j \in I_{x_i}^{\alpha^*(X)}$ it holds that

$$\alpha_j^* \leq \langle x_i, w_j\rangle = \langle x_i - x + x, w_j\rangle = \langle x_i - x, w_j\rangle + \langle x, w_j\rangle$$
$$\leq ||x - x_i|| \cdot ||w_j|| + \langle x, w_j\rangle \leq \rho(x_1, .., x_N; K)\beta + \langle x, w_j\rangle$$

This can be rearranged to

$$\langle x, w_j\rangle \geq \alpha_j^* - \beta\rho(x_1, .., x_N; K).$$

Hence, W is $(\alpha^* - \beta\rho(x_1, .., x_N; K))$-rectifying for x. This is true for all $x \in K$ therefore W is $(\alpha^* - \rho(x_1, .., x_N; K))$-rectifying on K. Let α_{true} be the true (unknown) maximal bias. Then,

$$(\alpha_{\text{true}})_j \geq \alpha^*(X)_j - \beta\rho(x_1, .., x_N; K)$$

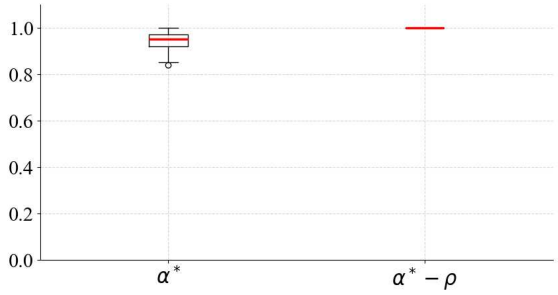

Figure 4.4.: Comparison of the proportion of test samples x_i so that W is α-rectifying on x_i before (left) and after (right) error correction. The comparison is based on 100 random matrices $W \in \mathbb{R}^{m \times n}$ with row vectors $w_1, .., w_m \in \mathbb{S}^n$. For each sampled matrix W we calculate the **MCBE** bias $\alpha^*(\{x_1, .., x_1 00\})$ using 100 sample points and the error corrected **MCBE** bias $\alpha^*(\{x_1, .., x_1 00\}) - \rho(x_1, .., x_{100}; \mathbb{S}^n)$. We then determine the percentage of 100 sampled points $x_1^{\text{test}}, .., x_{100}^{\text{test}} \in \mathbb{S}^n$, for which W is $\alpha^*(\{x_1, .., x_1 00\})$-rectifying and the percentage for which it is $\alpha^*(\{x_1, .., x_1 00\}) - \rho(x_1, .., x_{100}; \mathbb{S}^n)$-rectifying. The distribution of these percentages is compared using boxplots. The figure illustrates that W is $\alpha^*(\{x_1, .., x_1 00\}) - \rho(x_1, .., x_{100}; \mathbb{S}^n)$-rectifying for all test samples across all trials, suggesting that subtracting the covering radius ensures injectivity, even in a non-asymptotic setting.

holds for all $j = 1, .., m$. This can be rearranged to

$$\rho(x_1, .., x_N; K) \geq \alpha^*(X)_j - (\alpha_{\text{true}})_j.$$

Applying the norm it follows that

$$\epsilon \leq m(\beta \rho(x_1, .., x_N; K))^2.$$

□

It follows from Theorem 4.4.2 that subtracting the covering radius from the **MCBE**-bias allows us to guarantee injectivity asymptotically for sets where the covering radius is known, i.e. spheres. Figure 4.4 illustrates that also in a non-asymptotic setting, subtracting the covering radius ensures injectivity on the sphere. As the covering radius converges to zero for infinitely many sample points, theoretically, an arbitrarily small error can be achieved. This method of error correction works well in low dimensions, but in high dimensions, the asymptotical covering radius shows very slow convergence (Figure 4.5). The problem that arises with subtracting a large covering radius ρ is that

$$\alpha^*(X)_i - \rho(X, K) \leq \langle x, w_i \rangle$$

for all $i = 1, .., m$ with high probability. Then

$$\text{ReLU}_{W, \alpha_i^* - \rho}(x) = Wx - (\alpha_i^* - \rho)$$

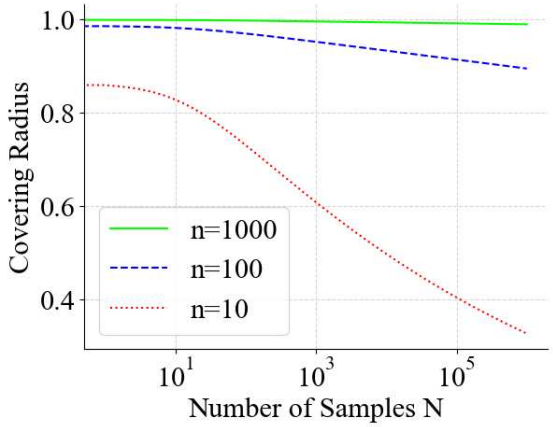

Figure 4.5.: Illustration of the slow convergence of the asymptotic covering radius $\rho(x_1, .., x_N; \mathbb{S}^n)$ as a function of the number of samples N for different dimensions n.

hence the ReLU-layer loses its non-linearity property in x. The non-linearity property however is essential for the accuracy of the neural network [Cyb89]. It is therefore evident that subtracting the covering radius is not suitable for high dimensions.

If one does not subtract the covering radius but rather interprets it as an error estimate for the **MCBE** algorithm, the convergence behavior of the covering radius implies that the number of samples required to achieve a satisfactory estimation of the true maximal bias explodes in high dimensions. In the following chapter, a potential solution to this problem will be proposed, namely a data-driven approach to Monte Carlo bias estimation.

5. Data Driven Monte Carlo Bias Estimation

In Chapter 4, we have seen that in high dimensions, a large number of samples is required for **MCBE** to converge. To circumvent this issue, we introduce a data driven version of the **MCBE**-algorithm, called data driven Monte Carlo Bias Estimation (**DDMCBE**). It circumvents the slow convergence of **MCBE** in high dimensions by drawing data points according to an approximation of the distribution of a given data set. We may conclude the α-rectifying property of W on all data that follow the same distribution. This shall provide a sufficient condition for injectivity of $\mathrm{ReLU}_{W,\alpha^*(X)}$ with high probability for all data points that are close (in terms of Euclidean distance) to the initial ones. To achieve this, we introduce two different approaches: 1) to blow up the train data via randomized smoothing and 2) kernel density estimation (Figure 5.1).

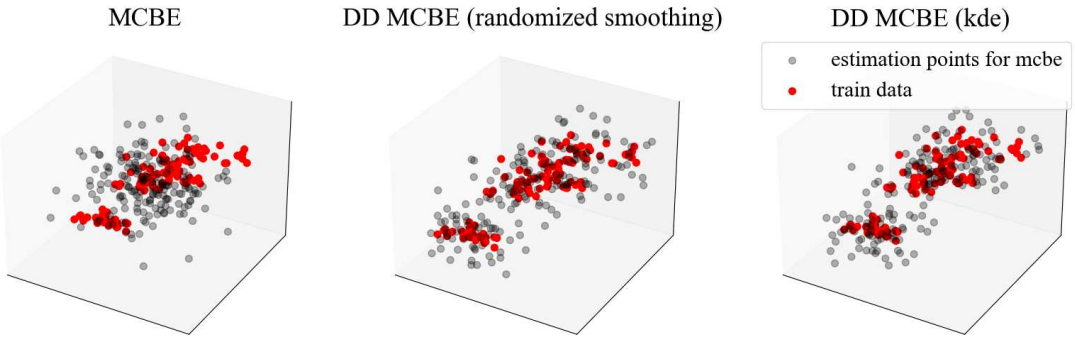

Figure 5.1.: Train data of the Iris benchmark dataset [Fis88] (red) and samples (grey) created by uniform sampling (left), randomized smoothing (mid), and kernel density estimation (right).

5.1. Randomized Smoothing

Randomized smoothing is a technique that can be used to increase the robustness of neural networks. Gaussian noise is added to the train data to smoothen the decision boundaries of the neural network [LAG+19] [CRK19]. In this section, we show how to make use of the same principle of adding noise to the sampling set for the Monte Carlo bias estimation. This yields a larger and denser set of samples than the train data, which

still follows a similar distribution. This shall reduce the number of samples required for good results in comparison to Monte Carlo bias estimation that uses uniform sampling.

Definition 5.1.1 (Randomized Smoothing [LAG+19])**.** Let $X^{(\text{train})} \in \mathbb{R}^{M \times n}$, $\sigma \in \mathbb{R}$ and let $U(X^{(\text{train})})$ denote the uniform distribution over the row vectors of $X^{(\text{train})}$. Performing Randomized Smoothing, the i-th sample is generated as

$$x_i := x + \epsilon \text{ with } \epsilon \ \sim \mathcal{N}(0, \sigma \mathbb{I}_n), x \ \sim U(X^{(\text{train})}).$$

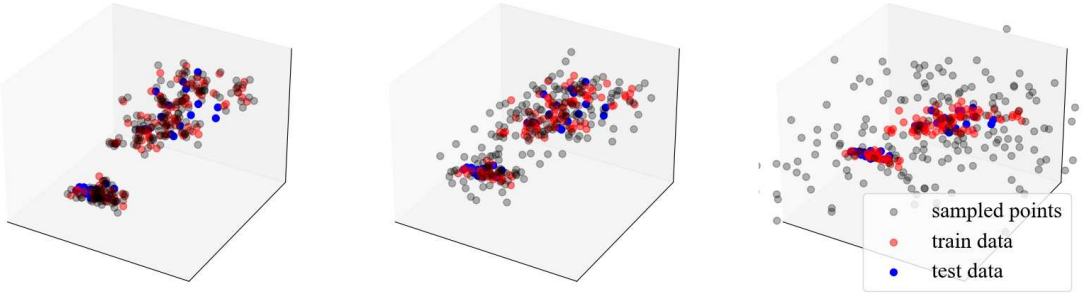

Figure 5.2.: Illustration of the importance of choosing an appropriate standard deviation σ for randomized smoothing. We compare points sampled with $\sigma = 0.1$ (left), $\sigma = 0.5$ (mid) and $\sigma = 2$ (right).

Input: weight matrix W, number of iterations for which α has to stay the same for convergence s, train data $X^{(\text{train})}$, standard deviation σ
Result: $\alpha^*(x_1, .., x_i)$
Initialize $\alpha^*(\{\})_j = \infty$ for $j = 1, .., m$;
$i = 0$;
while $(||\alpha^*(\{x_1, .., x_i\}) - \alpha^*(\{x_1, .., x_{i-s}\})|| > 0) \vee (i < s)$ **do**
  sample x from $U(X^{(\text{train})})$;
  sample ϵ from $\mathcal{N}(0, \sigma \mathbb{I}_n)$;
  set $x_i = x + \epsilon$;
  calculate J_{x_i};
  for $j \in J_{x_i}$ **do**
    $\alpha^*(\{x_1, .., x_i\})_j = \min\{\alpha^*(\{x_1, .., x_{i-1}\})_j, \langle x_i, w_j \rangle\}$;
  end
  $i = i + 1$
end

Algorithm 2: Data-Driven Monte Carlo Bias Estimation with Randomized Smoothing

In the original work [LAG+19] randomized smoothing was acting on the classifier, not on the train data. However, by applying this technique to the train data, we obtain a

method that generates samples for the **MCBE** algorithm customized to the data set (Algorithm 2).

To achieve good results, it is essential to select the standard deviation σ appropriately. It should be chosen so that the samples generated by randomized smoothing follow a distribution similar to that of the training set. If σ is too small, the samples will be insufficiently generalized, leading to poor performance on the test set (Figure 5.2, left). Conversely, if σ is too large, the samples will be overly dispersed, resulting in a similar situation as uniform sampling, where the number of samples needed to cover the space sufficiently explodes (Figure 5.2, right). When selecting σ, it is advisable to consider the empirical standard deviation of the training set as a reference point.

5.2. Kernel Density Estimation

In our second approach to data driven Monte Carlo bias estimation, we employ kernel density estimation, a well-established statistical method for estimating the density function of a given dataset. The estimated distribution can then be used to generate samples for the **MCBE** algorithm.

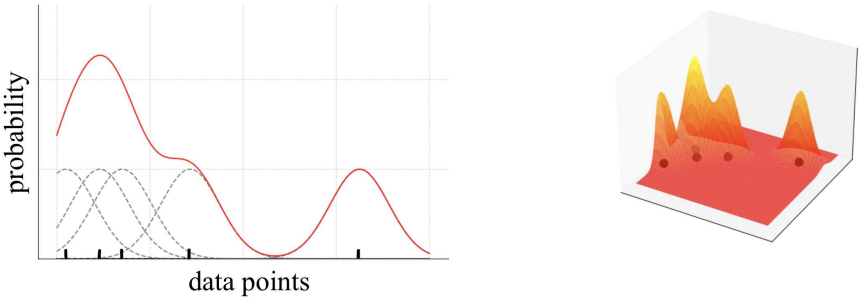

Figure 5.3.: Illustration of kernel density estimation with Gaussian kernels on one-dimensional data (left) and two-dimensional data (right). In the one-dimensional case, the grey dashed lines represent the individual kernels, and the red line represents the kernel density estimation, i.e., the sum of the individual kernels. In the two-dimensional case, the density estimate is represented as a smooth surface, which is the sum of two-dimensional Gaussian kernels centered around the data points.

Definition 5.2.1 (Kernel Density Estimate [Sim12])**.** Let $x_1^{(\text{train})}, .., x_M^{(\text{train})} \in \mathbb{R}^n$ be samples drawn from a common unknown distribution, $H \in \mathbb{R}^{n \times n}$ a symmetric and positive definite matrix and $K : \mathbb{R}^n \to \mathbb{R}$ a symmetric multivariate density function. Then the *Kernel Density Estimate* $\widehat{f}_H$ is defined as

$$\widehat{f}_H := \frac{1}{M} \sum_{i=1}^{M} K_H(x - x_i^{(\text{train})})$$

where

$$K_H(x) := |H|^{-\frac{1}{2}} K(H^{-\frac{1}{2}} x).$$

H is called the *bandwidth* and K is called the *kernel function*.

Kernel density estimation can be intuitively understood as taking the sum of individual kernels centered around each data point in the train set (Figure 5.3). The kernel is a smooth, symmetric function (often a Gaussian), and the bandwidth is a parameter that controls the width of the kernel, affecting the smoothness of the resulting density estimate. Similar to the randomized smoothing approach, we can use the kernel density estimate $\widehat{f}_H$ of the density of the train data to draw samples that follow approximately the same distribution as the train data. These samples can be used instead of the uniformly sampled points in **MCBE**, resulting in Algorithm 3). It is widely assumed that the selection of the kernel function, K, is not a significant factor for the accuracy of the estimate; however, the bandwidth, H, is crucial [WJ94]. In our experiments, we use the density function of a multivariate normal distribution as the kernel function K and Scott's rule, which we will define below, to estimate the optimal bandwidth.

Input: weight matrix W, number of iterations for which α has to stay the same for convergence s, train data $X^{(\text{train})}$, bandiwth H
Result: $\alpha^*(x_1, .., x_i)$
Initialize $\alpha^*(\{\})_j = \infty$ for $j = 1, .., m$;
$i = 0$;
calculate $\widehat{f}_H$ from train data;
while $(||\alpha^*(\{x_1, .., x_i\}) - \alpha^*(\{x_1, .., x_{i-s}\})|| > 0) \vee (i < s)$ **do**
  sample $x_i \sim \widehat{f}_H$;
  calculate J_{x_i};
  for $j \in J_{x_i}$ **do**
    $\alpha^*(\{x_1, .., x_i\})_j = \min\{\alpha^*(\{x_1, .., x_{i-1}\})_j, \langle x_i, w_j \rangle\}$;
  end
  $i = i + 1$
end

Algorithm 3: Data Driven Monte Carlo Bias Estimation with Kernel Density Estimation

Definition 5.2.2 (Scott's Rule [Sco79])**.** Given the i.i.d. samples $x_1^{(\text{train})}, .., x_M^{(\text{train})} \in \mathbb{R}^n$ with standard deviation $\sigma \in \mathbb{R}^n$. *Scott's rule* for optimal bandwidth selection is defined as selecting the bandwidth matrix $H \in \mathbb{R}^n$ in the following way:

$$(H)_{i,j} := \begin{cases} M^{\frac{-1}{n+4}} \sigma_i & \text{for } i = j \\ 0 & \text{else} \end{cases}$$

where M is the number of samples in the train data and σ_i is the i-th entry of σ. In practice, σ can be estimated from the data.

Remark. Conceptually, sampling from the kernel density estimate and randomized smoothing are closely related. It even seems that the two concepts are equivalent, given that they depend on the choice of the standard deviation σ in randomized smoothing and the bandwidth H in the kernel density estimate. This potential equivalence is speculative and requires further analysis and verification to be established rigorously. This lies beyond the scope of this thesis.

5.3. Comparison

To compare the performances of our bias estimation methods we trained neural networks on three widely used benchmark datasets (Iris [Fis88], MNIST [LCB10], CIFAR10 [Kri09]). We chose architectures where the first layer is an upsampling layer and has a ReLU activation function and used our different bias estimation methods to verify their injectivity. The neural network architectures are provided in the appendix. To quantify the performance of the different methods, we assessed point-wise invertibility, which refers to the percentage of test samples that can be reconstructed without loss from the output of $\mathrm{ReLU}_{W,\alpha^*(X)}$ with weights matrix W and the **MCBE**-bias $\alpha^*(X)$. To conclude point-wise invertibility in a sample x we checked if the weight matrix W is $\alpha^*(X)$-rectifying in x. Assuming W to be full spark, we know we can do that by verifying the cardinality of the set of active frame elements $|I_x^{\alpha^*(X)}| \geq n$. If this holds for x, we call x an invertible test samle. For the true maximal bias α_{true}, the percentage of invertible test samples would be 100%. A good estimation of α_{true} would be any bias $\alpha^*(X)$ that results in a percentage of invertible test samples close to 100%. We found that **DDMCBE** outperformed **MCBE** in all test cases (Figure 5.4). As expected, **MCBE** performed particularly poorly in high-dimensional scenarios. Which of the two proposed methods for **DDMCBE** performed better varied across datasets. To evaluate the computational efficiency of the different methods, we measured the mean execution times for **DDMCBE** with kernel density estimation, **DDMCBE** with randomized smoothing, and **MCBE**. As shown in Table 5.1, randomized smoothing outperformed kernel density estimation regarding running time, indicatings that randomized smoothing is a more efficient choice while providing comparable accuracy. Additionally, randomized smoothing and **MCBE** performed comparably in terms of running time but randomized smoothing outperformed **MCBE** in terms of accuracy. Hence randomized smoothing was the optimal method in the tested case, as it effectively balanced efficiency and accuracy.

MCBE	**DDMCBE** (RS)	**DDMCBE** (KDE)
0.060 s.	0.063 s.	0.111 s.

Table 5.1.: Mean execution times (in seconds) for different versions of the algorithm: **MCBE**, **DDMCBE** with randomized smoothing (RS), and **DDMCBE** with kernel density estimation (KDE). The algorithms were run using 1000 points on three-dimensional data. Execution times are averaged over 100 repetitions for each algorithm version.

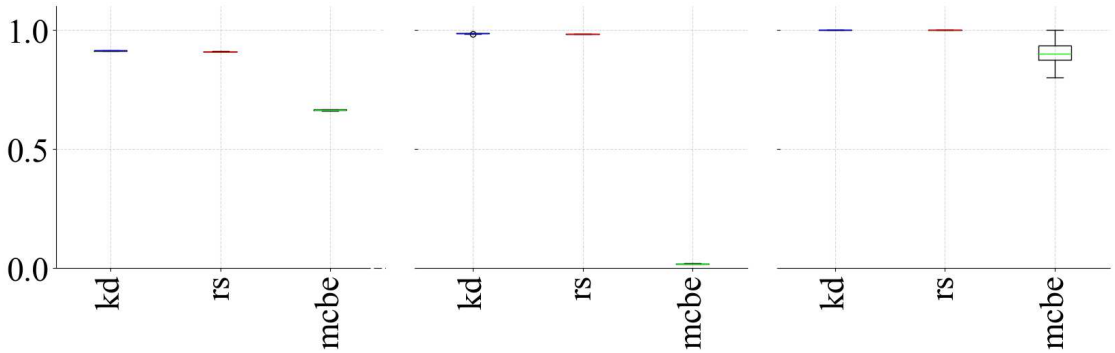

Figure 5.4.: Comparison of the proportion of point-wise invertible test samples (to be specified in Definition 6.1.1) after applying $\mathrm{ReLU}_{W,\alpha^*(X)} : \mathbb{R}^n \mapsto \mathbb{R}^m$ with weights matrix W and the **MCBE** bias $\alpha^*(X)$, where $\alpha^*(X)$ is calculated by **DDMCBE** with kernel density estimation (kd), **DDMCBE** with randomized smoothing (rs) and **MCBE** (mcbe). Comparisons are made for ReLU-layers trained on the CIFAR-10 dataset (left, $n = 3072$), the MNIST dataset (mid, $n = 784$) and the Iris dataset (right, $n = 3$). Each boxplot is computed on 10 repetitions, with (**DD**)**MCBE** evaluated with 100 points for Iris, and 500.000 points for MNIST and CIFAR-10.

6. Using DDMCBE to Monitor Injectivity of ReLU-layers

In Chapter 5, we introduced the **DDMCBE** algorithm, which allows us to study the injectivity of ReLU-layers in practice. This opens the possibility of directly examining the circumstances under which ReLU-layers are injective. In this chapter, we will explore this possibility to gain a deeper understanding of the conditions under which injectivity naturally occurs during the learning process of a ReLU-layer. This can be approached from multiple perspectives. First, we will introduce some methods for formally quantifying injectivity. We will then approach the topic of naturally occurring injectivity from the perspective of the redundancy of the layer. After that, we will revisit the topic from a different perspective by examining how naturally occurring injectivity evolves during the training process.

6.1. Methods and Approaches

In order to study the injectivity of ReLU-layers, we quantify the probability of this property. We propose two different approaches to achieve this.

The first approach is quantifying injectivity based on the proportion of test samples for which the weight matrix is α-rectifying. We have previously applied this approach in Chapter 4 and Chapter 5, without a formal definition. We will now revisit this and introduce the formal definition.

Definition 6.1.1 (Proportion of Point-wise Invertibility)**.** Let $\mathrm{ReLU}_{W,\alpha}$ be a ReLU-layer with bias vector $\alpha \in \mathbb{R}^m$ and weight matrix $W \in \mathbb{R}^{m\times n}$ and let $X_N = \{x_1, \ldots, x_N\}$. We define the *proportion of point-wise invertibility* $p_I(W, \alpha, X_N)$ as the percentage of points $x \in X_N$ such that W is α-rectifying on x. This can be interpreted as the percentage of points $x \in X_N$ for which reconstruction without loss is possible.

Assuming that W is full spark, we obtain $p_I(W, \alpha, X_N)$ by calculating the percentage of points $x \in X_N$ such that $|I_x^\alpha| \geq n$ holds. This allows for a straightforward implementation.

Our second approach to quantify injectivity is looking at the distance between α and the true maximal bias α_{true}. Given that α_{true} cannot be calculated, the **MCBE** bias $\alpha^*(X)$ is employed as an alternative. The distance between α and $\alpha^*(X)$ is comuted through a method that is related to the Hamming distance. The Hamming distance is a measure of the number of positions at which the corresponding elements of two binary vectors differ [WW95]. We will not use the Hamming distance directly; instead, we will apply its concept by calculating the percentage of entries where two given vectors do not exhibit a desired relationship.

Definition 6.1.2 (Hamming Proportion)**.** Let $\mathrm{ReLU}_{W,\alpha}$ be a ReLU-layer with bias vector $\alpha \in \mathbb{R}^m$ and weight matrix $W \in \mathbb{R}^{m \times n}$ and let $X_N = \{x_1, \ldots, x_N\}$. We define the *Hamming Proportion* $p_H(W, \alpha, X_N)$ as the percentage of the bias values α_i such that $\alpha_i \leq \alpha^*(X)_i$, where $\alpha^*(X)$ is the **MCBE**-bias.

We can interpret the Hamming proportion as the percentage of bias values that do not have to be changed to make the layer **MCBE**-injective.

Note that the two proposed measures to quantify injectivity represent distinct approaches. Consequently, it is anticipated that there will be discrepancy between the measures.

In Section 6.2 and Section 6.3, we will use the introduced measure to conduct experiments on the injectivity of ReLU-layers. For all experiments, we solve a classification task on the same datasets as in Section 5.3. We use architectures where the first layer has a ReLU activation function and conducted our numerical experiments on this layer. We refer to the appendix for detailed descriptions of the model architectures and training procedures.

6.2. Injectivity and Redundancy

Existing literature clearly shows a strong correlation between the redundancy of a ReLU-layer and its injectivity. In particular, it establishes a clear link between high redundancy and injectivity. This is consistent with the intuition that in higher-dimensional embedding spaces, more information is preserved, making it easier to differentiate between data points and maintain injectivity.

Puthawala et. al. [PKL+22] established a necessary but not sufficient condition for the asymptotical injectivity of ReLU-layers with random weight matrices with Gaussian i.i.d. entries and without bias, i.e., $\alpha = \vec{0}$, based on their redundancy r. For $r > 9$ the ReLU-layer is asymptotically injective and for $r < 3.3$ it is asymptotically not injective. Hence, the transition from injectivity to non-injectivity happens for

$$3.3 \leq r \leq 9.$$

Maillard et al. [MBB+23] showed numerically that in the same setting, a ReLU-layer is injective if $r > \hat{r}$ and not injective if $r < \hat{r}$ with

$$\hat{r} \in (6.6979, 6.6982).$$

We use the previously defined Hamming proportion to numerically reproduce these results in a non-asymptotic setting and to further investigate the relationship between redundancy and injectivity. We accomplish this by sampling random weight matrices $W \in \mathbb{R}^{m \times n}$ with Gaussian i.i.d. entries for dimensions $2 \leq n \leq 30$ and $n \leq m \leq 150$. For each sample, we calculate the Hamming proportion of the associated ReLU-layer and plot it to visualize the relationship between redundancy and injectivity (Figure 6.1 and Figure 6.2). We use the **MCBE**-bias obtained from the **DDMCBE** algorithm on synthetic data.

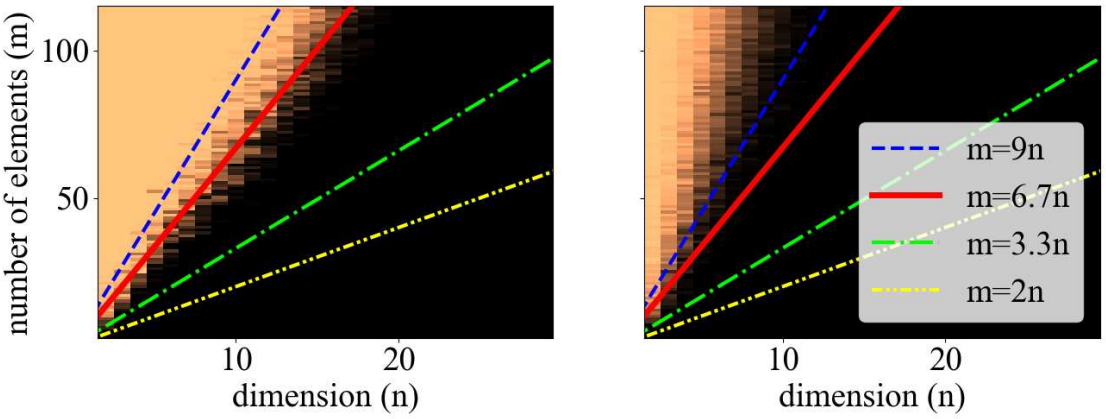

Figure 6.1.: Hamming proportion of ReLU-layers $\mathrm{ReLU}_{W,\alpha}$ with randomly sampled weight matrices $W \in \mathbb{R}^{m \times n}$ with Gaussian i.i.d. entries for dimensions $2 \leq n \leq 30$ and $n \leq m \leq 150$ and bias $\alpha = \vec{0}$ (left) and $\alpha = 0.\vec{2}5$ (right). Black tiles represent ReLU-layers with Hamming proportion $p_H(W, \alpha, X_N) = 0$, while copper tiles represent ReLU-layers with Hamming proportion $p_H(W, \alpha, X_N) = 1$, i.e. injective ReLU-layers. The lines represent different redundancies. The red line represents the theoretical injectivity boundary introduced by Maillard et al. [MBB+23]. The left image reproduces their results in a non-asymptotic setting. The right image shows that the injectivity boundary shifts to higher redundancy values as the bias values increase.

The left image of Figure 6.1 reproduces the results of Maillard et. al. [MBB+23], by investigating ReLU-layers with bias vector $\alpha = \vec{0}$ in a non-asymptotic setting. The image visualizes that the theoretical asymptotic threshold beyond which the ReLU-layers become injective appears even in a non-asymptotic setting. The right image of Figure 6.1 investigates the same setting but with a constant bias $\alpha = 0.\vec{2}5$, showing that the redundancy threshold at which the transition from injectivity to non-injectivity occurs increases with a higher bias. Figure 6.2 explores the same scenario but with a randomly sampled bias $\alpha \sim \mathcal{N}(0, \sigma^2)$, where $\sigma^2 = 0.1$ on the left image and $\sigma^2 = 1$ on the right image. We observe that for randomly sampled biases, the redundancy threshold for injectivity becomes less distinct as σ^2 increases.

Our experiments confirm that redundancy is one of the main deciding factors for the injectivity of a random ReLU-layer with no bias. We successfully reproduced the theoretical results of Maillard et al. [MBB+23] in a non-asymptotic setting. However, if the biases are also random, our experiments indicate that redundancy alone is not sufficient to predict injectivity. Overall we find that even though redundancy has a clear impact on the injectivity in the absence of a bias, the relationship between redundancy and injectivity becomes more complex and less predictable when a random bias is added.

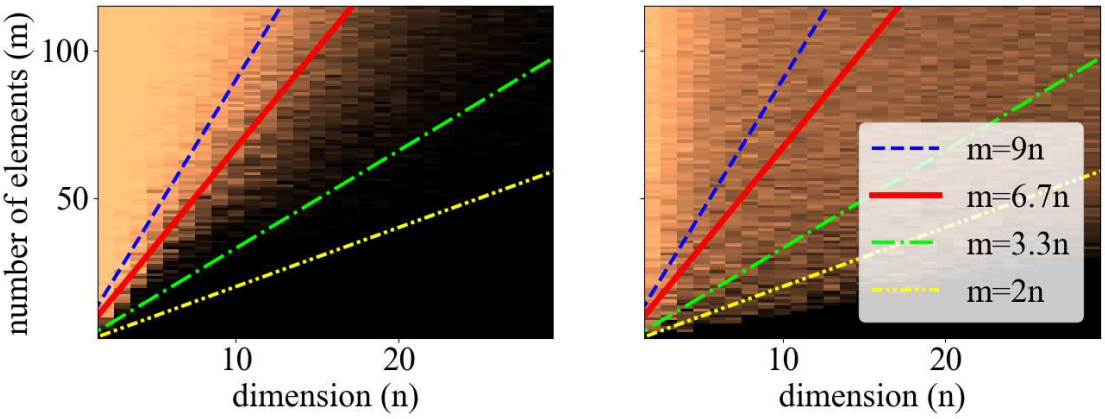

Figure 6.2.: Hamming proportion of ReLU-layers $\mathrm{ReLU}_{W,\alpha}$ with randomly sampled weight matrices $W \in \mathbb{R}^{m\times n}$ with Gaussian i.i.d. entries for dimensions $2 \leq n \leq 30$ and $n \leq m \leq 150$ and bias $\alpha \sim \mathcal{N}(0, 0.1)$ (left) and $\alpha \sim \mathcal{N}(0, 1)$ (right). Black tiles represent ReLU-layers with Hamming proportion $p_H(W, \alpha, X_N) = 0$, while copper tiles represent ReLU-layers with Hamming proportion $p_H(W, \alpha, X_N) = 1$, i.e. injective ReLU-layers. The lines represent different redundancies. The figure shows that, for randomly sampled biases, the redundancy threshold for injectivity becomes less distinct as the standard deviation increases.

6.3. Monitoring Injectivity During the Training Process

Our second experimental goal is to observe the evolution of the injectivity of ReLU-layers throughout the training process. In our experiments, we assess injectivity using both measures defined in Section 6.1 throughout every epoch of the training process. We do this for different datasets and levels of redundancy. As previously discussed, the two measures of injectivity represent distinct approaches, and therefore discrepancy between them is to be expected.

Figure 6.3 shows the injectivity during the training process, quantified by the proportion of pointwise invertibility. Figure 6.4 shows it quantified by the Hamming proportion. In both measures, there is a clear increase in injectivity as redundancy increases. This observation aligns with the results from the literature, which suggest that the injectivity of ReLU-layers with random weight matrices is asymptotically dependent on the level of redundancy being sufficiently high [PKL+22] [MBB+23]. As shown this is consistent with our findings in Section 6.2. Note that the jump from redundancy two to three has a significant effect in terms of invertibility, observed in both measures.

When quantifying injectivity based on the proportion of pointwise invertibility (Figure 6.3), we observe that the injectivity remains relatively consistent throughout the training process. This indicates that the training of the neural network does not significantly affect the layer's invertibility.

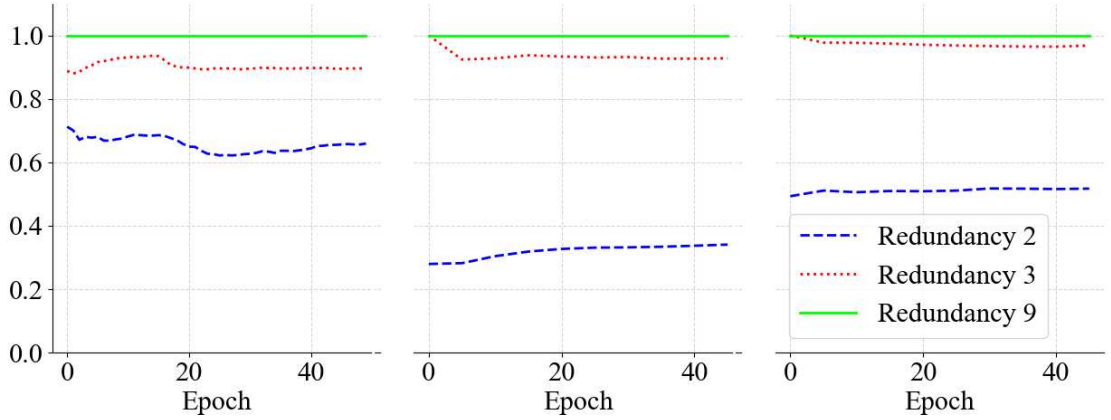

Figure 6.3.: Injectivity of the first ReLU-layer of a neural network during training quantified by the proportion of point-wise invertibility. Comparisons are made on ReLU-layers trained on the Iris dataset (left, $n = 3$), the MNIST dataset (middle, $n = 784$), and the CIFAR-10 dataset (right, $n = 3072$) and for different redundancies $\frac{m}{n}$. The figure shows how the proportion of point-wise invertibility remains relatively consistent throughout the training process across all datasets and redundancies.

When quantifying injectivity based on the Hamming proportion (Figure 6.4), the relationship between invertibility and the training process is less clear. Especially in the case of the low-dimensional Iris dataset (Figure 6.4, left) the Hamming proportion appears to exhibit a degree of randomness, which can be attributed to its low dimensionality. But also in the higher dimensional cases the training procedure does not have a consistent impact on the Hamming proportion across different datasets and levels of redundancy. This could suggest that the observed values where the Hamming proportion deviates from one are random. This remains a theory and is not proven. Overall it is clear that values where the Hamming proportion is not one are difficult to interpret. However, cases where the Hamming proportion is one have a clear interpretation, i.e., **MCBE**-injectivity. In contrast, all possible values of the proportion of pointwise invertibility are easy to interpret as they always directly reflect the proportion of test samples where perfect reconstruction is possible. Hence, in some cases, the Hamming proportion is not a suitable measure, where the proportion of pointwise invertibility is. However, because the Hamming proportion is computationally less expensive, it remains valuable in some cases.

In the light of our findings, it seems that the training process does not significantly and consistantly impact the injectivity of ReLU-ayers. It can be reasonably deduced that enforcing invertibility will not have a significant impact on the performance of the model, as we would otherwise expect injectivity to consistently increase or decrease during training. We will further test this in Chapter 7.

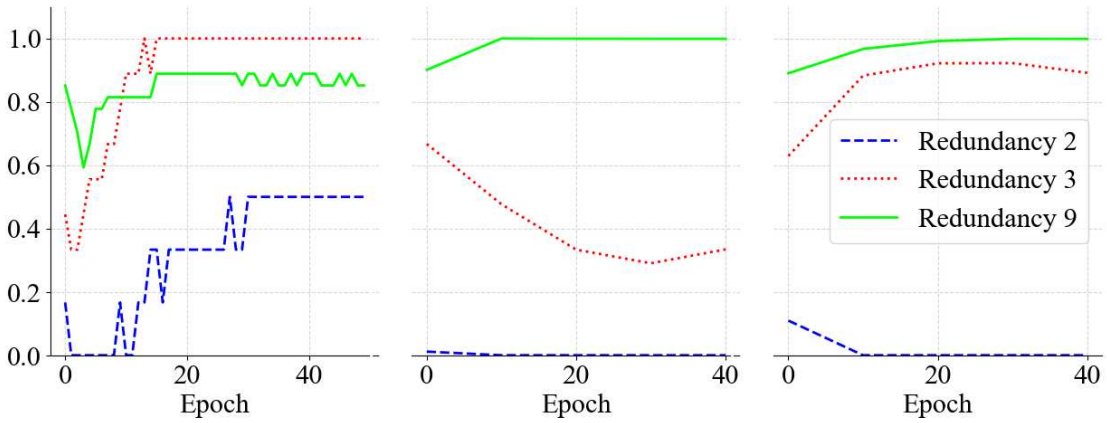

Figure 6.4.: Invertibility of the first ReLU-layer of a neural network during training quantified as the Hamming proportion. Comparisons are made on ReLU-layers trained on the Iris dataset (left, $n = 3$), the MNIST dataset (middle, $n = 784$), and the CIFAR-10 dataset (right, $n = 3072$) and for different redundancies $\frac{m}{n}$. The figure shows how the Hamming proportion exhibits inconsistent behavior throughout training across different datasets and redundancy levels.

7. Enforcing Injectivity of ReLU-Layers

Since the experiments in Section 6.2 did not reveal a strong relationship between the injectivity and the training process of the neural network, we now turn our attention to investigating the effect of injectivity on the model performance. We do that by using **DDMCBE** to enforce injectivity during the training and investigating whether this affects the overall performance of the model.

7.1. Methods and Approaches

To create **MCBE**-injective ReLU-layers, the training procedure is adapted by incorporating **DDMCBE**. We present two alternative approaches.

1. **Regularizing Non-Injectivity**: The first approach for incorporating **DDMCBE** into training is a regularization procedure where the ReLU-layer is gradually promoted to be injective during training. We implement this by modifying the learning objective, i.e., the loss function $\mathcal{L}$ as follows:

   $$\mathcal{L}^* = \mathcal{L} + \lambda \rho_{\text{inj}}(\alpha),$$

   where $\rho_{\text{inj}}(\alpha)$ is defined as

   $$\rho_{\text{inj}}(\alpha) = \left\| \begin{pmatrix} \max\{0, \alpha_1 - \alpha^*(X)_1\} \\ \vdots \\ \max\{0, \alpha_m - \alpha^*(X)_m\} \end{pmatrix} \right\|_2,$$

   with $\alpha^*(X)$ being the **MCBE**-bias calculated by **DDMCBE**. Note that the term $\rho_{\text{inj}}(\alpha)$ only affects those bias values where $\alpha_i > \alpha_i^*$ and has no effect, otherwise. In this sense, it penalizes non-injectivity. The parameter λ controls the impact of the regularization on the loss function.

2. **Manually Enforcing Injectivity**: The second approach for incorporating **DDMCBE** into training is to manually enforce injectivity by explicitly setting

   $$\alpha = \alpha^*(X)$$

   in each iteration of the training process, where $\alpha^*(X)$ is again the **MCBE**-bias estimated by **DDMCBE**.

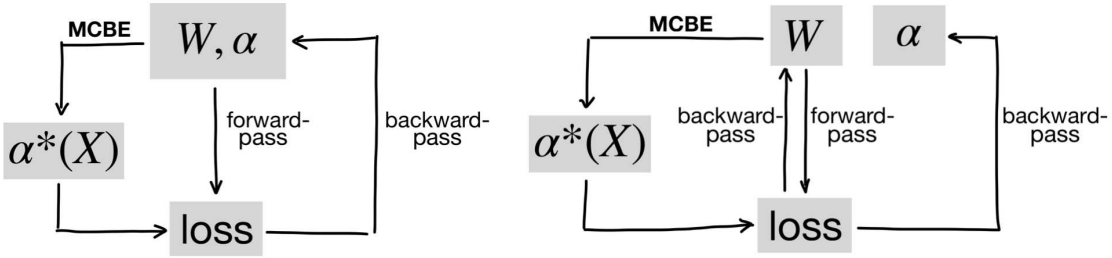

Figure 7.1.: The diagrams illustrate the dependencies of the variables of the two proposed approaches to enforce injectivity during training. By regularizing non-injectivity (left), the **MCBE**-bias $\alpha^*(X)$ is calculated using the weight matrix W. The loss is computed through a forward pass using W, α, and $\alpha^*(X)$. Then W and α are updated through the backward pass. By manually enforcing injectivity (right), the loss is computed only using W and $\alpha^*(X)$, and W and α are subsequently updated based on this loss through the backward pass.

Figure 7.1 illustrates the differences in the variable dependencies between the two approaches. In the first approach, W, α and $\alpha^*(X)$ are all represented in the loss function. The objective is to maximize accuracy while penalizing α for all indices i where $\alpha_i > \alpha^*(X)_i$. In the second approach, α is replaced by $\alpha^*(X)$ in the loss function. Consequently, α does not appear in the loss function, and the sole objective is to maximize accuracy. In both approaches, α^* is not updated through the backward pass but by performing **DDMCBE** on the updated weights. In our experiments, we recalculate $\alpha^*(X)$ at the beginning of each batch of the training data.

7.2. Enforce Injectivity During Training

In this section, we apply the previously outlined methodologies to modify the training process of neural networks in order to achieve **MCBE**-injectivity. For detailed descriptions of the model architectures and training procedures used in this section, we refer to the appendix.

Our first step is to evaluate the model performance when implementing the two proposed methods in comparison to the unregularized baseline. We do this with two goals in mind. The first goal is to determine whether numerical injectivity can be enforced during neural network training using our proposed methods without significantly compromising accuracy. The second goal is to identify which of the proposed methods should be preferred for future experiments, as conducting all upcoming experiments with both methods would be too computationally expensive.

We conduct the first evaluation by training neural networks on the Iris dataset [Fis88], implementing both proposed methods on the first layer of the network. We use this low-dimensional dataset to identify which of the methods works better before proceeding

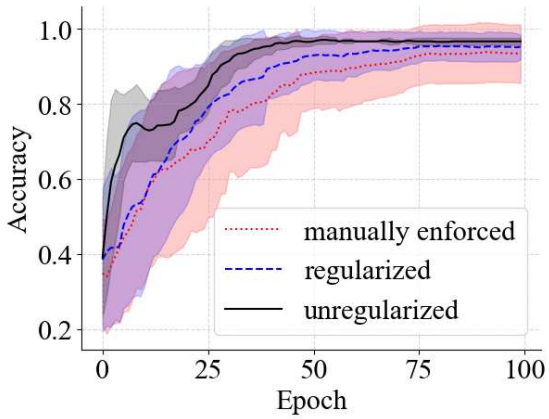

Figure 7.2.: The figure shows the comparison of the accuracy of neural networks in a classification taskusing the two proposed methods for enforcing injectivity on the first layer, as well as the baseline without enforced injectivity. The comparisons are based on the mean of 25 repetitions of training on the Iris dataset. The redundancy of the first layers is $\frac{m}{n} = 2$. For both injectivity enforcement methods, injectivity has only a minimal impact on accuracy.

to high-dimensional cases. We compare both methods and the baseline by evaluating the proportion of pointwise invertibility and the Hamming proportion (Table 7.1). We observe that both methods significantly increase both injectivity measures compared to the baseline. The Hamming proportion of the network with manually enforced injectivity is one by design, as it compares the model bias to the estimated maximal bias, which are identical in this scenario. However, in terms of the proportion of point-wise invertibility, the regularization method outperforms manual enforcement. Figure 7.2 provides a comparison of the accuracies of the classification task during the training process for the three approaches. For both injectivity enforcement methods, our experiment found that enforcing injectivity had only a minimal effect on the accuracy. The regularization method demonstrated slightly better accuracy compared to the manual enforcement method. Given the overall findings from this first experiment, we conclude that the regularization method should be preferred, as it outperformed the manual enforcement approach in terms of both, accuracy and proportion of point-wise invertibilty.

The next step is to use the regularization method to investigate the relationship between injectivity and accuracy of neural networks in higher-dimensional settings. As previously discussed, we will no longer use both methods simultaneously due to the considerable computational demands in high-dimensional scenarios. We again solve a classification task on the MNIST dataset [LCB10], and the CIFAR10 dataset [Kri09]. For the CIFAR10 dataset, achieving optimal performance with a ReLU-layer as the first layer proved to be challenging. Therefore, we applied regularization to the first upsampling ReLU-layer in the architecture, which is the seventh layer. Further research is needed to understand the impact of the layer's placement on its injectivity. For additional details on the model

	Regularized	Manually Enforced	Unregularized
$p_I(W, \alpha, X_N)$	0.98	0.9	0.67
$p_H(W, \alpha, X_N)$	0.7	1	0.16

Table 7.1.: The table shows a comparison of the injectivity measures for the first layer of the considered neural networks using the two proposed methods for enforcing injectivity, as well as the baseline without enforced injectivity. The comparisons are based on the mean of 25 repetitions of training on the Iris dataset. We measure injectivity using both the proportion of pointwise invertibility $p_I(W, \alpha, X_N)$ and the Hamming proportion $p_H(W, \alpha, X_N)$. We calculate both measures using the test set X_N. The redundancy of the first layers is $\frac{m}{n} = 2$. Both methods result in a significant increase in injectivity according to both measures compared to the baseline.

architectures, we refer to the appendix.

	MNIST	CIFAR10
Regularized	0.73	1
Unregularized	0	0

Table 7.2.: The table shows the proportion of point-wise injectivity of an **MCBE**-invertible ReLU-layer, compared to a non-regularized one on the test set of MNIST and CIFAR10. The redundancy of the layers is $\frac{m}{n} = 2$. For both datasets, the regularization increases the proportion of pointwise injectivity significantly.

Figure 7.3 and Table 7.2 summarize our findings in the high-dimensional settings. Our results suggest that enforcing injectivity may, in some cases, lead to a strong trade-off with model performance (Figure 7.3). In the case of MNIST, this trade-off is of minimal consequence; however, for CIFAR10, the impact on model performance is significant. Table 7.2 shows the proportion of point-wise invertibility of the respective layers with and without regularization. In the case of CIFAR10, regularization leads to a higher proportion of point-wise invertibility on the test set compared to MNIST, suggesting a trade-off between accuracy and proportion of point-wise invertibility. In other words: the neural network trained on the CIFAR10 dataset achieves a very high value in our injectivity measure, but lower accuracy, while the neural network trained on the MNIST dataset achieves a lower value in our injectivity measure, but higher accuracy. This leads us to hypothesize that there exists indeed a trade-off between these two quantities. This trade-off can potentially be balanced by adjusting the constant factor λ, which scales the regularization term in the loss function.

Summarizing, we conclude that injectivity can be achieved through regularization even in high-dimensional scenarios; however, this seems to come with a trade-off in accuracy.

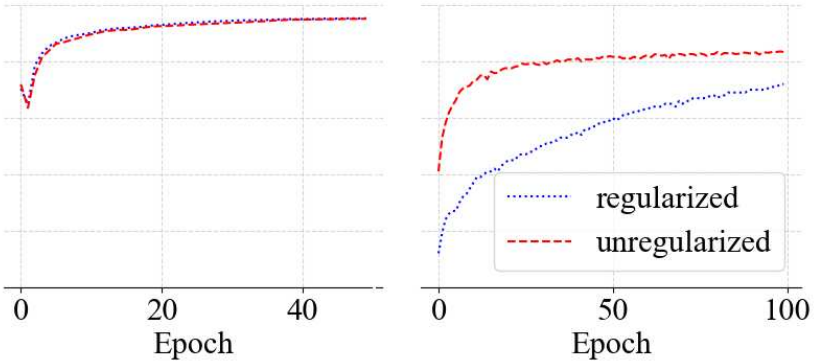

Figure 7.3.: The figure shows the classification accuracy of a neural network with an **MCBE**-invertible layer as the first upsampling ReLU-layer of the model architecture, compared to a non-regularized baseline throughout the training process. The neural networks are trained on the MNIST (left) and CIFAR10 (right) datasets. The accuracies that the models achieved in classification tasks are measured on the test set. The redundancy of the layers is $\frac{m}{n} = 2$. For the MNIST dataset (left), the trade-off between injectivity and model performance is negligible. For the CIFAR10 dataset (right), the trade-off is significant.

This was shown by a significant increase in the proportion of point-wise invertibility. In Section 7.3, we will connect these findings with the error associated with the reconstruction of the input of a ReLU-layer output.

7.3. Reconstruction

The input $x \in \mathbb{R}^n$ of a ReLU-layer can be reconstructed from its output $z = \mathrm{ReLU}_{W,\alpha}(x) \in \mathbb{R}^m$ by calculating the least-squares solution x^* of

$$W^+(x)x = \alpha^+(x) + z^+(x),$$

where $W^+(x)$ is the matrix, whose row vectors are the row vectors w_i of W that satisfy $(\mathrm{ReLU}_{W,\alpha}(x))_i > 0$, i.e., whose indices are in I_x^α, and $\alpha^+(x)$ and $z^+(x)$ are defined analogously [HBE23]. For any point $x \in \mathbb{R}^n$, for which the ReLU-layer is α-rectifying the reconstruction is exact.

We shall compare the reconstruction error for samples of the test set with and without regularization. We use the same experimental setup as in Section 7.2 and compare the mean reconstruction errors of the test sets after training, defined as the Euclidean distance between the input x and the reconstructed input x^*. The results of this procedure are presented in Table 7.3. Although perfect reconstruction was not achieved in our examples, the regularization reduced the reconstruction error significantly across all datasets. Figure

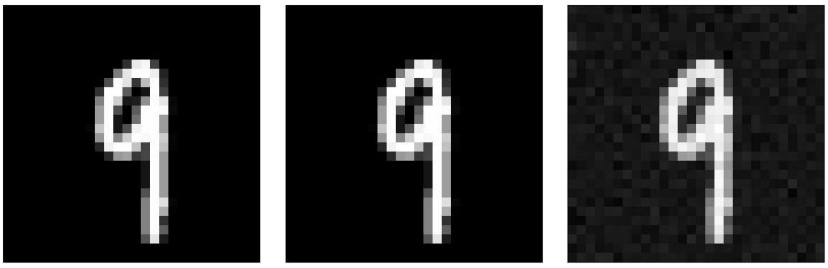

Figure 7.4.: The figure shows the original (left) and the reconstructed MNIST images from the output of trained ReLU-layers. Mid: Regularized. Right: Unregularized. The redundancy of the layers is $\frac{m}{n} = 2$.

	Iris	**MNIST**	**CIFAR10**
Regularized	0.84	0.79	0.006
Unregularized	2.02	1.63	57.87

Table 7.3.: The table shows the reconstruction error of a ReLU-layer, where injectivity was enforced, compared to a non-regularized baseline, computed on the test sets. The redundancy of the layers is $\frac{m}{n} = 2$. Across datasets, the regularized layers demonstrate lower reconstruction errors than the non-regularized layers.

7.4 provides a visual comparison of a reconstructed MNIST image using a regularized ReLU-layer versus an unregularized one.

The reduction in the reconstruction error aligns with the anticipated outcome of enforcing injectivity. However, the regularization procedure is not without drawbacks, namely a reduction in accuracy. In Section 7.4, we will examine another potential drawback: its impact on network stability.

7.4. Stability Analysis

Neural network stability influences the robustness and reliability of its predictions, particularly in terms of how small changes in the input or the parameters affect the output. A common measure for assessing the stability of a neural network layer is its upper Lipschitz bound, which quantifies the maximum extent to which the layer can amplify the distances between input vectors. In the context of invertibility, the lower Lipschitz bound of the layer becomes crucial, too, as it reflects the upper Lipschitz bound of the inverse. If the inverse does not exist, the lower Lipschitz bound is zero. Accordingly, ReLU-layers are *bi-Lipschitz stable* if both the upper and lower Lipschitz bounds exist.

Definition 7.4.1 (Bi-Lipschitz Stable)**.** A ReLU-layer $\mathrm{ReLU}_{W,\alpha}$ is *bi-Lipschitz stable* if

there are $A, B > 0$ such that

$$A||x - x'|| \leq ||\operatorname{ReLU}_{W,\alpha}(x) - \operatorname{ReLU}_{W,\alpha}(x')|| \leq B||x - x'||$$

holds for all $x, x' \in \mathbb{R}^n$.

The following theorem connects the notion of bi-Lipschitz stability to injectivity.

Theorem 7.4.1 (ReLU Bi-Lipschitz Bounds). Let $W \in \mathbb{R}^{m \times n}$ and $\alpha \in \mathbb{R}^m$. For any $x \in K \subseteq \mathbb{R}^n$ let W^+ be defined as the matrix whose row vectors are the active frame elements of W (as previously defined in Section 4.4). Let us further denote the smallest singular value of the matrix V as $\lambda_-(V)$ and the largest singular value as $\lambda_+(V)$. Then the upper and lower Lipschitz bounds of $\operatorname{ReLU}_{W,\alpha}$ on K are given as

$$A_0 = \frac{1}{2} \min_{x \in K} \lambda_-(W^+(x)),$$

$$B_0 = \max_{x \in K} \lambda_+(W^+(x)).$$

Moreover, $\operatorname{ReLU}_{W,\alpha}$ is injective on K if and only if $A_0 > 0$.

Note that $\lambda_-(W^+(x))$ and $\lambda_+(W^+(x))$ correspond to the lower and upper frame bounds of the sub-frame of active elements for x and α.

To gain insight into how enforcing injectivity affects the stability, we define sets containing the lowest and highest singular values of $W^+(x)$ for each point in some set K.

Definition 7.4.2. We define the sets $\mathcal{A}, \mathcal{B}$ for $K \subseteq \mathbb{R}^n$ as

$$\mathcal{A} = \{\lambda_-(W^+(x)) \text{ for } x \in K\}$$

and

$$\mathcal{B} = \{\lambda_+(W^+(x)) \text{ for } x \in K\}.$$

These sets contain the bounds A_0 and B_0 for the set K as minimum and maximum, respectively.

Low values in the set $\mathcal{B}$ indicate good stability behavior of the corresponding ReLU-layer in the usual sense, while high values in the set $\mathcal{A}$ indicate good stability behavior of the inverse mapping. Zero values in the set $\mathcal{A}$ correspond to points, where the ReLU-layer is not invertible.

To illustrate the effect of enforcing injectivity on stability we calculate the sets $\mathcal{A}$ and $\mathcal{B}$ on the test set after training. The results are shown in Figure 7.5. Across the datasets, we found, that enforcing injectivity increases the values in both $\mathcal{A}$ and $\mathcal{B}$. The increased values in $\mathcal{B}$ indicate a reduction in the stability of the ReLU-layer. The increase of values in $\mathcal{A}$ from zero indicates the injectivity of the layer. Hence, it seems that enforcing injectivity also involves a trade-off with the stability of the layer, in addition to the trade-off with accuracy. This could potentially be improved by incorporating a regularizer that promotes stability during neural network training, in a manner similar to how we addressed injectivity. Recently, Nenov et al. [RNon] introduced such a regularizer that is provably differentiable while promoting stability. However, further research is needed to fully understand the interplay between injectivity and stability.

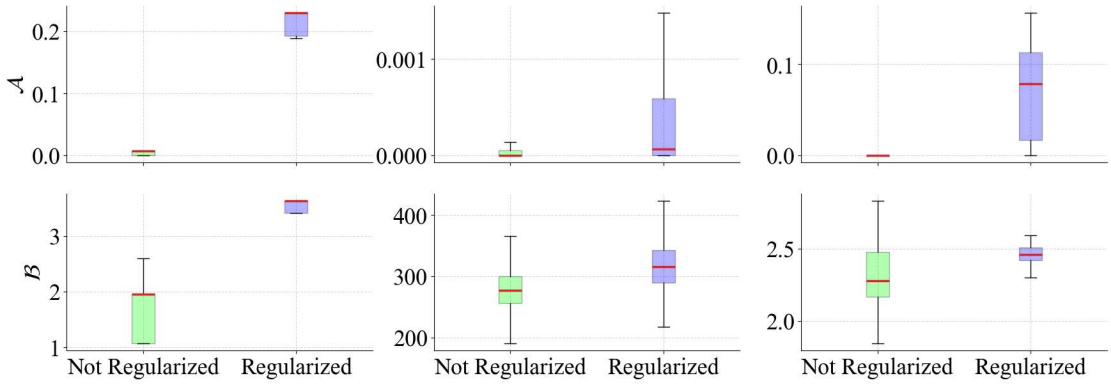

Figure 7.5.: Sets $\mathcal{A}, \mathcal{B}$, defined such that high values in the set $\mathcal{A}$ indicate stability of the inverse mapping and low values in the set $\mathcal{B}$ indicate stability of the ReLU-layer. Sets are shown for ReLU-layer trained on the Iris dataset (left), the MNIST dataset (middle), and the CIFAR10 dataset (right). The increased values in $\mathcal{B}$, observed across datasets in the regularized case, indicate a reduction in the stability of the ReLU-layers. The increase of values in $\mathcal{A}$, also observed across datasets in the regularized case, from zero indicates the injectivity of the layers.

7.5. Discussion

Our experiments demonstrate that **MCBE**-injectivity can be enforced in two ways: manually replacing the learned bias with the **MCBE**-bias, or penalizing non-injectivity in the loss function using the **MCBE**-bias. A comparison of the two methods on the low-dimensional Iris dataset [Fis88] yielded the conclusion that penalizing non-injectivity is the preferred method. Applying it led to a significant decrease in the reconstruction error from the laye output on the test sets. However, we found that there seems to be an inherent trade-off between injectivity on one hand and accuracy and stability on the other. This trade-off could potentially be balanced by adjusting the factor λ, which scales the regularization term in the loss function. Further research is needed to identify if the observed behavior is specific to the considered examples or more general.

8. Conclusion

This thesis employs a frame-theoretic perspective to introduce a Monte Carlo-type method for numerically verifying a sufficient condition for the injectivity of a ReLU-layer in a neural network. The method is fine-tuned via data-driven sampling to ensure applicability in high-dimensional cases. For any given weight matrix and dataset X the introduced algorithm estimates a bias $\alpha^*(X)$ which is maximal with the property that any ReLU-layer with bias α is **MCBE**-injective if $\alpha \leq \alpha^*(X)$. The introduced **MCBE**-bias $\alpha^*(X)$, alongside an alternative way to quantify injectivity, is employed to study the information flow in ReLU-layers by comparing α with $\alpha^*(X)$. We use this methodology to investigate the relationships between injectivity and redundancy, and injectivity and the training process. This allows us to confirm known results from the literature about injectivity and redundancy [PKL+22], and additionally suggests that injectivity is not strongly correlated with the training process. Based on these results, we propose an algorithm to enforce injectivity by penalizing bias values that exceed the values of $\alpha^*(X)$ during training. The investigation of the effects of enforcing injectivity reveals a trade-off between injectivity on the one hand and accuracy and stability on the other. Further research is needed to identify if the observed behavior is specific to the considered examples or more general.

The ability to create **MCBE**-injective ReLU layers opens new possibilities for exploring their interpretability. Specifically, this could involve studying the feature space by perturbing the output of the layer and analyzing the corresponding changes in the reconstructed input. Exciting open questions that remain include the investigation of the geometry of the feature space of ReLU-layers and its impact on the injectivity of subsequent layers, as well as verifying the full-spark assumption during gradient updates.

In summary, this thesis contributes a valuable method for the numerical verification of the injectivity of ReLU-layers and opens avenues for deeper exploration of neural network interpretability. The findings and methods introduced lay the groundwork for future research focused on enhancing our understanding of neural network dynamics.

Bibliography

[ACM12] Boris Alexeev, Jameson Cahill, and Dustin G Mixon. Full spark frames. *Journal of Fourier Analysis and Applications*, 18:1167–1194, 2012.

[AFG+24] Wedad Alharbi, Daniel Freeman, Dorsa Ghoreishi, Brody Johnson, and N Lovasoa Randrianarivony. Declipping and the recovery of vectors from saturated measurements. *arXiv preprint arXiv:2402.03237*, 2024.

[AKW+18] Lynton Ardizzone, Jakob Kruse, Sebastian Wirkert, Daniel Rahner, Eric W Pellegrini, Ralf S Klessen, Lena Maier-Hein, Carsten Rother, and Ullrich Köthe. Analyzing inverse problems with invertible neural networks. *arXiv preprint arXiv:1808.04730*, 2018.

[BCE06] Radu Balan, Pete Casazza, and Dan Edidin. On signal reconstruction without phase. *Applied and Computational Harmonic Analysis*, 20(3):345–356, 2006.

[BD09] Thomas Blumensath and Mike E Davies. Sampling theorems for signals from the union of finite-dimensional linear subspaces. *IEEE Transactions on Information Theory*, 55(4):1872–1882, 2009.

[BGC+19] Jens Behrmann, Will Grathwohl, Ricky TQ Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. In *International conference on machine learning*, pages 573–582. PMLR, 2019.

[BSI05] Leemon Baird, David Smalenberger, and Shawn Ingkiriwang. One-step neural network inversion with pdf learning and emulation. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 966–971. IEEE, 2005.

[CK12] Peter G. Casazza and Gitta Kutyniok. *Finite frames: Theory and applications*. Springer, 2012.

[CRK19] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *international conference on machine learning*, pages 1310–1320. PMLR, 2019.

[Cyb89] George V. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.

[DB95] Gustavo Deco and Wilfried Brauer. Nonlinear higher-order statistical decorrelation by volume-conserving neural architectures. *Neural Networks*, 8(4):525–535, 1995.

[DGM86] Ingrid Daubechies, Alex Grossmann, and Yves Meyer. Painless nonorthogonal expansions. *Journal of Mathematical Physics*, 27(5):1271–1283, 1986.

[DKB14] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[DS52] Richard J Duffin and Albert C Schaeffer. A class of nonharmonic fourier series. *Transactions of the American Mathematical Society*, 72(2):341–366, 1952.

[DSDB16] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[FGS24] Vincent Froese, Moritz Grillo, and Martin Skutella. Complexity of deciding injectivity and surjectivity of relu neural networks. *arXiv preprint arXiv:2405.19805*, 2024.

[Fis88] Ronald Fisher. Iris. UCI Machine Learning Repository, 1988. DOI: https://doi.org/10.24432/C56C76.

[FXLW20] Fenglei Fan, Jinjun Xiong, Mengzhou Li, and Ge Wang. On interpretability of artificial neural networks: A survey. *IEEE Transactions on Radiation and Plasma Medical Sciences*, 5:741–760, 2020.

[GBW+19] Raymond Geis, Adrian P Brady, Carol C Wu, Jack Spencer, Erik Ranschaert, Jacob L Jaremko, Steve G Langer, Andrea Borondy Kitts, Judy Birch, William F Shields, et al. Ethics of artificial intelligence in radiology: summary of the joint european and north american multisociety statement. *Radiology*, 293(2):436–440, 2019.

[GRUG17] Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. *Advances in neural information processing systems*, 30, 2017.

[HBE23] Daniel Haider, Peter Balazs, and Martin Ehler. Convex geometry of ReLU-layers, injectivity on the ball and local reconstruction. In *Proceedings of the International Conference on Machine Learning (ICML), Honolulu, Hawaii*, 2023.

[HEB24] Daniel Haider, Martin Ehler, and Peter Balazs. Injectivity of ReLU-layers: Perspectives from frame theory. *arXiv*, abs/2406.15856, 2024.

[HKK+18] Xiaowei Huang, Daniel Kroening, Marta Kwiatkowska, Wenjie Ruan, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. Safety and trustworthiness of deep neural networks: A survey. *arXiv preprint arXiv:1812.08342*, page 151, 2018.

[JSO18] Jörn-Henrik Jacobsen, Arnold Smeulders, and Edouard Oyallon. i-revnet: Deep invertible networks. In *ICLR 2018-International Conference on Learning Representations*, 2018.

[KC+08] Jelena Kovačević, Amina Chebira, et al. An introduction to frames. *Foundations and Trends in Signal Processing*, 2(1):1–94, 2008.

[KD18] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems*, 31, 2018.

[Kin14] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[KW19] Rainer Kelz and Gerhard Widmer. Towards interpretable polyphonic transcription with invertible neural networks. *arXiv preprint arXiv:1909.01622*, 2019.

[LAG+19] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE symposium on security and privacy (SP)*, pages 656–672. IEEE, 2019.

[LCB10] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[MBB+23] Antoine Maillard, Afonso S Bandeira, David Belius, Ivan Dokmanić, and Shuta Nakajima. Injectivity of relu networks: perspectives from statistical physics. *arXiv preprint arXiv:2302.14112*, 2023.

[NH10] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[PKL+22] Michael Puthawala, Konik Kothari, Matti Lassas, Ivan DokmaniÄ‡, and Maarten de Hoop. Globally injective relu networks. *Journal of Machine Learning Research*, 23(105):1–55, 2022.

[PNR+21] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64, 2021.

[RM15] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.

[RNon] Peter Balazs Rossen Nenov, Daniel Haider. (almost) smooth sailing: Towards numerical stabilityof neural networkstrough differentiable regularization of the condition number. In *Proceedings of the International Conference on Machine Learning (ICML)*, in preperation.

[RS16] A. Reznikov and E. B. Saff. The covering radius of randomly distributed points on a manifold. *International Mathematics Research Notices*, 2016(19):6065–6094, 2016.

[Sco79] David W Scott. On optimal and data-based histograms. *Biometrika*, 66(3):605–610, 1979.

[Sim12] Jeffrey S Simonoff. *Smoothing methods in statistics*. Springer Science & Business Media, 2012.

[SMV+19] Wojciech Samek, Grégoire Montavon, Andrea Vedaldi, Lars Kai Hansen, and Klaus-Robert Müller, editors. *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer Cham, 2019.

[TT13] Esteban G Tabak and Cristina V Turner. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, 2013.

[TVE10] Esteban G Tabak and Eric Vanden-Eijnden. Density estimation by dual ascent of the log-likelihood. *Commun. Math. Sci. 8 (1) 217 - 233, March 2010*, 2010.

[VLL23] Jonatan Vallin, Karl Larsson, and Mats G Larson. The geometric structure of fully-connected relu-layers. *arXiv preprint arXiv:2310.03482*, 2023.

[WJ94] Matt P Wand and M Chris Jones. *Kernel smoothing*. CRC press, 1994.

[WW95] Bill Waggener and William N Waggener. *Pulse code modulation techniques*. Springer Science & Business Media, 1995.

A. Appendix

A.1. Neural Network Architectures and Training Details

All datasets were normalized before further processing. In all cases an Adam optimizer [Kin14] with a learning rate of 0.01 was used.

A.1.1. Neural Network Architecture for Iris and redundancy $\frac{m}{n} \in \{2, 3\}$ of the first layer

For the Iris data and redundancy $\frac{m}{n} \in \{2, 3\}$, we used two fully connected layers with ReLU activation function followed by an output layer with Softmax activation function.

1. FC1: m units, ReLU activation
2. FC2: 27 units, ReLU activation
3. Output Layer: 3 units, Softmax activation

A.1.2. Neural Network Architecture for MNIST

For the MNIST data, we used one fully connected layers with ReLU activation function followed by an output layer with Softmax activation function.

1. FC1: m units, ReLU activation
2. Output Layer: 10 units, Softmax activation

A.1.3. Neural Network Architecture for CIFAR10

For the CIFAR10 data, we used the following architecture.

1. CONV1: 2D convolutional layer, input channels: 3, output channels: 8, kernel size: 3×3.
2. CONV2: 2D convolutional layer, input channels: 8, output channels: 32, kernel size: 3×3, stride: 1, padding: 1, ReLU activation.
3. MAX POOL1: Max pooling layer, kernel size: 2×2, stride: 2.
4. CONV3: 2D convolutional layer, input channels: 32, output channels: 64, kernel size: 3×3, stride: 1, padding: 1, ReLU activation.

5. DROPOUT1: Dropout layer, probability: 0.5.
6. CONV4: 2D convolutional layer, input channels: 64, output channels: 128, kernel size: 3×3, stride: 1, padding: 1, ReLU activation.
7. MAX POOL2: Max pooling layer, kernel size: 2×2, stride: 2.
8. CONV5: 2D convolutional layer, input channels: 128, output channels: 256, kernel size: 3×3, stride: 1, ReLU activation.
9. DROPOUT2: Dropout layer, probability: 0.5.
10. FC1: Fully connected layer, input features: 6×6×256, output features: 256, ReLU activation.
11. FC2: Fully connected layer, input features: 256, output features: 512, ReLU activation.
12. FC3: Fully connected layer, input features: 512, output features: 128, ReLU activation.
13. FC4: Fully connected layer, input features: 128, output features: 64, ReLU activation.
14. Output Layer: Fully connected layer, input features: 64, output features: 10, logits output.

We performed the regularization discussed in Chapter 7 on the layer FC2.