Universität Bremen

Zentrum für Technomathematik

Principles of Neural Network Architecture Design: Invertibility and Domain Knowledge

Dissertation

- zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften -
(Dr. rer. nat.)

vorgelegt von

Jens Behrmann

Bremen, 30. Oktober 2019

Promotionskomitee:

1. Gutachter: Prof. Dr. Peter Maaß
2. Gutachterin: Jun.-Prof. Dr. Asja Fischer

Contents

Chapter 1

Introduction and Motivation

"All models are wrong, but some models are useful." - George Box.

We as humans build an understanding of our environment through models. Ranging from physical laws of the universe to microscopic biological interactions, abstract models guide our way of thinking about ourselves and the surroundings we live in. But what is the fundamental origin of those models? How do we build them or test if they actually reflect the nature they try to model? The answer is simple: through observations and interactions with our world. In other words, all models and approaches are at some point data-based.

Among all models, mathematical constructions play a central role. While qualitative descriptions provide intuition about behavior or reasons, mathematical models allow to go beyond intuition by quantifying processes. Yet, the origin of most mathematical constructions like physical or economical laws is (at least partially) based on an observation or an experiment. After describing them in mathematical terms, they can be transferred to other domains or generalized in various ways. The key to this generalization is often their dependence on parameters, where different values of those parameters enable the astonishing flexibility of mathematical constructions.

However, four main drivers are changing our ways to obtain new models. First, there is an ever increasing demand for simulating complex processes. Among the most demanding quests is certainly to model or even create general intelligence. Second, our ability to observe the world is drastically increasing by the development of new sensors like medical devices or everyday tools like smartphones. Third, we are not only able to measure this data, but can also store and access it when needed. Fourth, growing computational resources permit to process these observations.

All these changes fuel the construction of ever increasingly complex models. The key underlying mechanisms, however, remain. Starting with known mathematical constructions obtained through observations and quantification, we transfer the models to new domains by adding parameters to these models. Once data is available in this new domain, parameters are identified to obtain a specification of the parametrized model. While these traditional mechanisms still form the basis, their balance is shifting. Since more complex behavior is being modeled, our knowledge and thus our starting models for identification

need to be more flexible. Hence, more parameters are added and the attention of the modeling process is shifting towards the identification of the parameters. Since we often do not know how to model the problem at hand, thinking about this process as *learning* from observations is many times fitting.

When thinking about these constructions as functions which receive an input and return a desired output, neural networks are natural candidates since they are parametrized and most importantly flexible functions. Mathematically speaking, they are *universal approximators* (Cybenko, 1989). However, this property is often not the main reason for their usage, since other models like kernel approaches share this property (Micchelli et al., 2006). Most interestingly, they allow the processing of inputs in several steps or in hierarchies, which enables to incorporate knowledge about the target domain in a natural manner. For example, the visual processing of our brain is often modeled using these abstract mechanisms.

Thus, neural networks fundamentally follow the classical approach of using knowledge obtained from other tasks and adding flexibility through parametrization. Most prominently, specific neural network architectures, called convolutional neural networks (CNNs), were designed from knowledge about signal processing and human vision (LeCun et al., 2015). Even in other domains like language understanding, neural networks allow to incorporate structures, which we think as human helpful for models targeting not only an understanding of language but also the generation of creative text or speech.

Since this domain knowledge, besides flexibility, is one of the main drivers of neural network architecture design, studying the question

> *What kind of architecture should I use/ design in a new environment?*

forms one of the two main pillars of this thesis. In particular, we will discuss approaches to design suitable architecture for the processing of mass spectra and test their applicability to tumor classification tasks. Based on given similarities and differences to image data, we modify standard CNNs to better respect the nature of spectra arising from imaging mass spectrometry measurements.

While domain knowledge plays a pivotal role for neural networks, their flexibility is the answer to the growing demand of complex models and the availability of data. However, this flexibility brings many challenges. For once, the learning process is hard to understand. Yet, this is usually not surprising since it is often designed to adapt to behavior, that we do not know how to model ourselves. After training, using the neural network is straightforward and enables access to models with high complexity. Yet, even understanding basic mechanisms of these models is often beyond our current abilities. This not only hinders their usage in high-stakes applications, but also slows down the development of better models.

One way to circumvent these problems is by enforcing certain guarantees, which hold no matter which parameters the model learns. For example, having the guarantee to reverse the processing of the network could allow to answer the question:

> *If the output is Y, what is the corresponding input X?*

Having access to an answer to the question above could for example tell us, which inputs certain worst-case outputs induce. More positively, it also permits access to an input that results in a desired output behavior.

In mathematical terms, these guarantees are met when the model is invertible. This property can be very powerful not only to provide an answer to the questions above, but also to reason about inputs in a probabilistic manner. As we will see in this thesis, it will even allow us to generate new data. Thus the study of invertible neural networks will form the second main pillar. For this, we first dive into an analysis to which degree standard network architectures are invertible. Afterwards, we introduce new structures to obtain guaranteed invertibility. Having this property, we then propose an algorithm to use these networks for generative modeling. Finally, we discuss how invertible networks can help to understand and control learned representations of data.

As emphasized in this introduction, mathematical modeling is based on two major components: constructing a parametrized model using domain knowledge and learning/ identifying the parameters from data/ observations. First, we provide an overview of the articles that serve as the basis of this thesis. Afterwards, we introduce core concepts of neural network models and the learning process in the next chapter. In this foundation chapter, we reverse the ordering above (fitting to the major theme of this thesis) and start with the statistical fundamentals of learning. Then we discuss models based on neural network architectures. At the end of this chapter, we state the goals of this thesis more precisely.

1.1 Published Articles and Contribution of Author

Each chapter in this thesis is based on parts of an article by the author and collaborators, as indicated in the beginning of each chapter. In general, some aspects of those article are not presented in this thesis and only referenced. In most cases, those parts were omitted because they were orthogonal to the main focus of this thesis. In some cases, the research of the omitted parts was not conducted by the author and thus is only reviewed or referenced in this thesis.

This section briefly states the role of the author in each published article (below the article), listed by the appearance of the article in this thesis:

> **Jens Behrmann**, Sören Dittmer, Pascal Fernsel, Peter Maass: *Invariance and inverse stability under ReLU*, 2019, (submitted to IEEE Transactions on Neural Networks and Learning Systems)

Jens Behrmann contributed to all aspects of the article, in equal contribution with Sören Dittmer. This article serves as the basis for chapter 3.

> **Jens Behrmann**, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, Jörn-Henrik Jacobsen: *Invertible residual networks*, 2019, (International Conference on Machine Learning (ICML))

Jens Behrmann contributed to all aspects of the article, in equal contribution with Will Grathwohl and Jörn-Henrik Jacobsen. This article serves as the basis for chapter 4.

> Ricky T. Q. Chen, **Jens Behrmann**, David Duvenaud, Jörn-Henrik Jacobsen: *Residual flows: invertible generative modeling*, 2019, (under submission; early version presented at: ICML workshop on Invertible Networks and Normalizing Flows)

Jens Behrmann derived the generalization of i-ResNets to ℓ_p-norms, which is the basis for the presentation of the i-ResNet in chapter 4. Furthermore, Jens Behrmann derived and implemented the Neumann gradient estimator and contributed to the proofs of the unbiased estimators. The experiments and the improvements for generative modeling based on i-ResNets are only referenced in chapter 5.

> Jörn-Henrik Jacobsen, **Jens Behrmann**, Richard Zemel, Matthias Bethge: *Excessive invariance causes adversarial vulnerability*, 2019, (International Conference on Learning Representations (ICLR))

Jens Behrmann contributed to the conceptual idea of the reverse view on adversarial examples, to the analysis of the proposed independence cross-entropy loss and to the design of some experiments. The other contributions of this article are only referenced or reviewed in the related work section of chapter 6.

> Jörn-Henrik Jacobsen, **Jens Behrmann**, Nicholas Carlini, Florian Tramer, Nicolas Papernot: *Exploiting excessive invariance caused by norm-Bounded adversarial robustness*, 2019, (under submission; early version presented at: ICLR workshop on Safe Machine Learning: Specification, Robustness and Assurance)

Jens Behrmann contributed to the main concept of the article, to the definition of both modes of adversarial examples and to the theoretical analysis based on the synthetic spheres dataset. This article and in particular the synthetic datasets serves as an example in chapter 6, while other contributions of this article are only reviewed.

> **Jens Behrmann**, Christian Etmann, Tobias Boskamp, Rita Casadonte, Jörg Kriegsmann, Peter Maass: *Deep learning for tumor classification in imaging mass spectrometry*, 2018, Bioinformatics, volume 34, issue 7, pages 1215 - 1223

Jens Behrmann contributed to all aspects of the article, in equal contribution with Christian Etmann. This article serves as the basis for chapter 7.

Remark:
At several places in this thesis, above articles are only cited. The symbol *** highlights those articles, in order to distinguish these articles by the author of the thesis from other articles. For example: (Behrmann et al., 2018a***) or (Jacobsen et al., 2019b***).

Chapter 2

Mathematical Foundations of Deep Learning

2.1 Statistical Learning

2.1.1 Goals of Learning

In order to introduce the fundamental goals of learning, we will first lay the setting from a probabilistic viewpoint. In general, we will consider the probability space $(\Omega, \mathcal{A}, \mathbb{P})$ with following elements:

- Ω denotes the sample space: all possible outcomes of the random experiment.
- $\mathcal{A} \subseteq \mathcal{P}(\Omega)$ (power set) denotes the σ-algebra of Ω, called the event space: the space of potential results of the experiment
- $\mathbb{P} : \mathcal{A} \to [0, 1]$ denotes the probability measure on the event space.

Furthermore, let $V : \Omega \to \Omega'$ denote a random variable, where Ω' is called the target space of the random variable. If $\Omega' = \mathbb{R}$, we call V a real random variable. If $\Omega' = [N]$, where $[N] := \{1, \ldots, N\}$, we call V a (finite) discrete random variable. A realization of a random variable corresponds to $V(\omega) = v$, with $\omega \in \Omega$.

The distribution of random variable V with respect to the probability space $(\Omega, \mathcal{A}, \mathbb{P})$ is $\mathbb{P}_V = \mathbb{P} \circ V^{-1}$. For notational convenience, we will use the notation $V \sim \mathcal{D} := \mathbb{P}_V$ to denote that random variable V has distribution $\mathcal{D}$. Furthermore, if the probability measure $\mathbb{P}$ is absolutely continuous with respect to the measure λ used for the target space Ω' of random variable V, we can uniquely identify the distribution via a probability density function (pdf) $p : \Omega' \to \mathbb{R}_+$ as

$$\mathbb{P}[V \in A] = \int_A p(v)\, d\lambda(v),$$

where $\int_{\Omega'} p(v)\, d\lambda(v) = 1$ and $A \in \mathcal{A}$.

When considering real random variables, we use the Lebesgue measure for λ and simplify the notation via $d\lambda(v) = dv$. For discrete random variables, the measure λ refers to the counting measure and we write

$$\int_A p(v)\, d\lambda(v) = \sum_{v \in A} p(v).$$

In this case, we call p a probability mass function (pmf) to emphasize the distinction between real and discrete random variables. Furthermore, for discriminative models we will often consider the conditional pdf/pmf $p(y \mid X = x) := \frac{p(y,x)}{p(x)}$, where we condition the random variable Y on the realization x of random variable X. To shorten notation, we will sometimes use $p(y \mid x) := p(y \mid X = x)$.

To simplify subsequent discussions, we will restrict ourselves to the most often encountered case of real random variables with target space $\mathbb{R}^d$ under the Lebesgue measure and discrete random variables over $[N]$. For a generalization to other domains and measures, as well as for a measure-theoretic treatment of probability theory we refer to (Billingsley, 1995).

Based on the previously introduced notation, we formulate the goal of learning via a risk function, see for example (Shalev-Shwartz and Ben-David, 2014, sec. 3.2.2).

Definition 2.1 (Risk function)**.** *Consider a set of models $\mathcal{F}$ and let ℓ be any function with $\ell : \mathcal{F} \times \Omega \to \mathbb{R}_+$ (called loss function from now on). Further, let $\mathcal{D}$ be a distribution over the sample space Ω and $F \in \mathcal{F}$. Then, we define the risk function L as*

$$L_{\mathcal{D}}(F) = \mathbb{E}_{v \sim \mathcal{D}}[\ell(F, v)].$$

Note, that the above definition incorporates:

- Prediction tasks with $\Omega = \mathcal{X} \times \mathcal{Y}$, where a categorical domain $\mathcal{Y}$ refers to classification and continuous domains $\mathcal{Y} = \mathbb{R}^d$ to regression.
- Unsupervised tasks with $\Omega = \mathcal{X}$, e.g. density estimation of real random variables over $\mathbb{R}^d$.

However, in practical scenarios we do not have access to the data distribution $\mathcal{D}$, but only to a sample $\mathcal{T} = \{v^{(i)}\}_{i=1}^N$, where $v_i \in \Omega$. Thus, we need to consider an empirical risk, which is defined as follows, see (Shalev-Shwartz and Ben-David, 2014, sec. 3.2.2).

Definition 2.2 (Empirical risk)**.** *Let $\mathcal{T} = \{v^{(i)}\}_{i=1}^N$ be a training set, where $v^{(i)}$ are realizations of random variables $V^{(i)}$ that are independent and identically distributed (short: i.i.d.). Further. let ℓ, F be defined as in Definition 2.1. Then, we define the empirical risk L as*

$$L_{\mathcal{T}}(F) = \frac{1}{N} \sum_{i=1}^{N} \ell(F, v^{(i)}).$$

While estimating the true risk via the empirical risk is a key idea of learning from samples, the set of models under consideration $\mathcal{F}$ is another crucial ingredient of learning. Since we choose an appropiate $\mathcal{F}$, we call this set *hypothesis class*. Combining these ideas leads to the following formalization, see (Shalev-Shwartz and Ben-David, 2014, sec. 2.3):

Definition 2.3 (Empirical risk minimization with inductive bias)**.** *Let $\mathcal{T} = \{v^{(i)}\}_{i=1}^{N}$ be a training set of realizations of i.i.d. random variables. Further, let model F be from the hypothesis class $\mathcal{F}$ and let $\ell : \mathcal{F} \times \Omega \to \mathbb{R}_+$ define a loss function. Then, empirical risk minimization (ERM) with inductive bias is formulated as*

$$F^* \in \operatorname*{argmin}_{F \in \mathcal{F}} L_{\mathcal{T}}(F) = \operatorname*{argmin}_{F \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} \ell(F, v^{(i)}).$$

This restricted search over a hypothesis class refers to an inductive bias since this choice is made before seeing training examples. Ideally this choice reflects some prior knowledge about the learning task at hand. While this seems restrictive, an ERM learner without any inductive bias will lead to severe overfitting. This results in a tradeoff, called *bias-complexity tradeoff*. On the one hand, choosing rich hypothesis classes allows to achieve a small empirical risk. On the other hand, rich classes can result in an increase of the estimation error, see (Shalev-Shwartz and Ben-David, 2014, sec. 5.2) for a more detailed discussion.

This thesis explores various instances of inducing a bias into the learning process by studying neural network architecture design principles. Another crucial choice for ERM is the loss function, for which we introduce examples in the next section.

2.1.2 Maximum Likelihood Estimation

While the previous section studied learning using an abstract formalism, this section proceeds by making two fundamental assumptions:

- Hypothesis class $\mathcal{F}$ is restricted to parametric models
- Loss functions ℓ are derived based on generative assumptions on data.

In particular, we associate a model $F \in \mathcal{F}$ to a parameter $\theta \in \Theta$, where Θ denotes the parameter space. For example, an affine function $F : \mathbb{R} \to \mathbb{R}$, with $x \mapsto ax + b$, can be identified via its slope $a \in \mathbb{R}$ and bias $b \in \mathbb{R}$, hence $\theta = (a, b)$ and $\Theta = \mathbb{R}^2$.

By using these parametric models as a probability density function (pdf, in case of regression or density estimation) or probability mass function (pmf, in case of classification), we can formulate the likelihood function, see e.g. (Held and Bove, 2013):

Definition 2.4 (Likelihood function)**.** *Let $\theta \in \Theta$ denote a parameter and let $v \in \Omega'$. Further, let $p_\theta(v)$ denote a probability density function (pdf) or probability mass function (pmf). Then, we call $\ell : \Theta \times \Omega' \to \mathbb{R}_+$ the likelihood loss, where the mapping $v \mapsto p_\theta(v)$ is the pdf/pmf and $\theta \mapsto p_\theta(v)$ is the likelihood function.*

This definition leads us to the powerful idea of maximum likelihood estimation, see e.g. (Held and Bove, 2013):

Definition 2.5 (Maximum likelihood estimation)**.** *Let $v \in \Omega'$ be fixed, $\theta \in \Theta$ and ℓ denote the likelihood loss from Definition 2.4. Then, θ_{MLE} is called a maximum likelihood estimate (MLE) if*

$$\theta_{MLE} \in \operatorname*{argmax}_{\theta \in \Theta} \ell(\theta, v) = \operatorname*{argmin}_{\theta \in \Theta}(-\log \ell(\theta, v)).$$

Remark 2.6. *Above reformulation using the logarithm is valid since the logarithm is strictly monotone. Furthermore, for multiple i.i.d. realizations $v^{(i)}$, $i \in [N]$, the logarithm turns a product of likelihoods into a sum of likelihoods and thus simplifies the derivative. Additionally, this negative log-likelihood allows to switch from maximization to minimization, which is more common in numerical optimization.*

Furthermore, MLE is a special case of empirical risk minimization (ERM) with the likelihood loss, see (Shalev-Shwartz and Ben-David, 2014, sec. 24.1.2), since

$$\theta_{MLE} = \theta_{ERM} \in \operatorname*{argmin}_{\theta \in \Theta} L_{\mathcal{T}}(\theta) = \operatorname*{argmin}_{\theta \in \Theta} \frac{1}{N} \sum_{i=1}^{N} -\log \ell(\theta, v^{(i)}) = \operatorname*{argmin}_{\theta \in \Theta} \frac{1}{N} \sum_{i=1}^{N} -\log p_\theta(v^{(i)}).$$

However, note that MLE optimizes in parameter space Θ, wheres ERM was defined (Def. 2.3) via optimizing over functions in $\mathcal{F}$.

In addition to considering parametric models as the hypothesis class, assumptions on the data generation process allow to derive particular loss functions and models. For this, consider a supervised learning task with $\Omega = \mathcal{X} \times \mathcal{Y}$. If we formulate the goal of learning in a *discriminative* manner, we need to find a parametric conditional probability $p_\theta(y|X = x)$, which explains the training data well. For example, least-squares with linear regression can be derived as a MLE via the generative assumption of linear dependence under additive Gaussian noise, i.e.

$$y(x) = w_*^T x + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma^2),$$

where $\mathcal{N}$ denotes the normal distribution with variance σ^2, $w_* \in \mathbb{R}^d$ the true weights, $x \in \mathbb{R}^d$ the inputs and $y \in \mathbb{R}$ the targets, see (Murphy, 2013, sec. 7.2).

While the above regression example is an often encountered problem, in this thesis we focus on classification as well as on maximum likelihood based unsupervised learning (which will be discussed in chapter 5). Consider the following model for classification with C classes, see (Murphy, 2013, sec. 8.3.7):

Definition 2.7 (Multi-class logistic regression)**.** *Let $x \in \mathbb{R}^d$ denote the inputs and let labels $y \in \{0, 1\}^C$ (one-hot encoded) be categorically distributed as*

$$Cat(y \mid \mu) = \prod_{k=1}^{C} \mu_k^{\mathbb{I}[y_k = 1]},$$

where $\mathbb{I}[\cdot]$ denotes the indicator function and μ_k is the probability that y is from class k. By assuming linear dependence of the label y on the inputs x after a softmax function,

multi-class logistic regression can be formulated via

$$p_\theta(Y = \boldsymbol{k} \mid X = x) = \text{softmax}(Wx)_k = \frac{\exp\left(W_k^T x\right)}{\sum_{k'=1}^{C} \exp\left(W_{k'}^T x\right)},$$

where $Y = \boldsymbol{k}$ denotes that the target is from class k, $W \in \mathbb{R}^{C \times d}$ and W_k denotes the k-th row of W. In this case, the parameter θ is described by the matrix W.

Remark 2.8. *The negative log-likelihood with multi-class logistic regression is often called* ***categorical cross-entropy*** *in the deep learning literature. In this thesis we will adopt both terms and primarily use cross-entropy when studying loss functions from a information-theoretic perspective.*

More generally, the exponential family allows generative assumptions beyond additive Gaussian noise and categorical distributions. Corresponding maximum likelihood approaches can be found in (Murphy, 2013, sec. 9.3). Before we relax above assumptions of linear dependence between inputs x and targets y using deep neural networks, we briefly review some core concepts from information theory.

2.1.3 Information Theory

Information theory is tightly related to machine learning, such that many algorithms can be developed and studied via information-theoretic considerations (see e.g. in section 6.4.2 of this thesis). A pivotal concept is the relative entropy/ Kullback-Leibler divergence, which can be interpreted as the dissimilarity of a probability distribution q to a reference distribution p, see (Cover and Thomas, 2006, sec. 2.3):

Definition 2.9 (Kullback-Leibler divergence)**.** *Let p, q denote two probability density functions on Ω', where the support of p is contained in q ($supp(p) \subset supp(q)$). Then, the Kullback-Leibler (KL) divergence is defined as*

$$D_{KL}(p(v) \parallel q(v)) = \int_{\Omega'} p(v) \log \frac{p(v)}{q(v)} dv.$$

Further, we write in short $D_{KL}(p \parallel q)$.

The KL-divergence is not a metric since it is not symmetric and does not satisfy the triangle inequality. But it is always non-negative and zero if and only if $p = q$, see (Cover and Thomas, 2006, Thm. 2.6.3):

Lemma 2.10 (Information Inequality)**.** *Let p, q be defined as in Definition 2.9. Then,*

$$D_{KL}(p \parallel q) \geq 0,$$

where $D_{KL}(p \parallel q) = 0$ if and only if $p = q$ almost everywhere.

Proof. Following (Cover and Thomas, 2006), we have

$$\begin{aligned} -D_{KL}(p \parallel q) &= \int_{\Omega'} p(v) \log \frac{q(v)}{p(v)} dv = \mathbb{E}_V \left[\log \frac{q(v)}{p(v)} \right] \leq \log \mathbb{E}_V \left[\frac{q(v)}{p(v)} \right] \\ &= \log \int_{\Omega'} p(v) \frac{q(v)}{p(v)} dv = \log 1 = 0, \end{aligned}$$

where the inequality follows from Jensen's inequality. Furthermore, the integral simplifies because $\int_{\Omega'} q(v)dv = 1$ due to q being a probability density function. Equality holds, iff we have equality in Jensen's inequality, which occurs iff $p = q$ almost everywhere. □

Based on the Kullback-Leibler divergence, we can further define mutual information (MI) as a key quantity to measure dependence between random variables, see (Cover and Thomas, 2006, sec. 2.3 and 2.5). MI will be a crucial property in information-theoretic considerations in section 6.4.2.

Definition 2.11 (Mutual information (MI))**.** *Let X, Y be random variables on spaces $\mathcal{X}$, $\mathcal{Y}$ with joint probability density $p(x, y)$ and marginal densities $p(x)$, $p(y)$. Then the mutual information based on the Kullback-Leibler divergence (see Definition 2.9) is defined as*

$$I(X;Y) = D_{KL}(p(x,y) \parallel p(x)p(y)) = \int_{\mathcal{X}\times\mathcal{Y}} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} dx\, dy\,.$$

For a discrete random variable y, the integral turns into a sum over all possible events $\mathcal{Y}$. The conditional mutual information with random variable Z is defined as

$$I(X;Y|Z) = \int_{\mathcal{X}\times\mathcal{Y}\times\mathcal{Z}} p(x,y,z) \log \frac{p(x,y|z)}{p(x|z)p(y|z)} dx\, dy\, dz\,.$$

Besides mutual information, we introduce the notion of entropy, see (Cover and Thomas, 2006, sec. 2.2 and sec. 9.1) first to understand properties of mutual information and second to understand the connection between the Kullback-Leibler divergence and negative log-likelihood.

Definition 2.12 (Entropy, joint entropy, conditional entropy)**.** *Let Y be a discrete random variable on $\mathcal{Y}$ and X be a continuous random variable on $\mathcal{X}$ with marginal densities $p(y), p(x)$. Then the entropy $H(Y)$ and differential entropy $h(X)$ are defined as*

$$\begin{aligned} H(Y) &= -\sum_{\mathcal{Y}} p(y) \log p(y) \\ h(X) &= -\int_{\mathcal{X}} p(x) \log p(x) dx. \end{aligned}$$

The joint entropy for continuous random variables X, Y with joint density $p(x, y)$ is defined as

$$h(X,Y) = -\int_{\mathcal{X}\times\mathcal{Y}} p(x,y) \log p(x,y) dx\, dy\,.$$

Furthermore, the conditional entropy is defined as

$$h(Y|X) = \int_{\mathcal{X}} p(x) h(Y|X=x) dx = -\int_{\mathcal{X}} p(x) \int_{\mathcal{Y}} p(y|x) \log p(y|x) dy\, dx\,.$$

After introducing above information-theoretic quantities, we state some properties of mutual information and entropy, see (Cover and Thomas, 2006, Thm. 2.4.1, Thm. 2.5.2 and Thm. 2.8.1).

Lemma 2.13 (Properties of mutual information (MI) and entropy)**.** *Let X, Y, Z be continuous random variables and $D : \mathcal{X} \to \mathcal{Z}$ be a deterministic transform. Then following properties hold:*

(i) MI and entropy: $I(X;Y) = h(Y) - h(Y|X)$

(ii) Chain rule of MI: $I(X;Y,Z) = I(X;Y) + I(X;Z|Y)$

(iii) Data processing inequality of MI: $I(X;Y) \geq I(D(X);Y)$.

Proof. See (Cover and Thomas, 2006) for the proofs, in particular we refer for (i) to Theorem 2.4.1, for (ii) to Theorem 2.5.2. and for (iii) to Theorem 2.8.1. □

Furthermore, there is a tight connection between the Kullback-Leibler divergence and maximum likelihood estimation (MLE), see (Shalev-Shwartz and Ben-David, 2014, sec. 24.1.2). This equivalence allows us to reason statistically via MLE or in an information-theoretic manner via the Kullback-Leibler divergence.

Lemma 2.14. *Let p denote the pdf of a random variable $V \sim \mathcal{D}$ and let p_θ denote the likelihood function of a parametric model. Then, minimizing the negative log-likelihood is equivalent to minimizing the Kullback-Leibler divergence between p and p_θ, hence*

$$\theta_{MLE} \in \operatorname*{argmin}_{\theta \in \Theta} \mathbb{E}_{V \sim \mathcal{D}} \left[-\log p_\theta(v) \right] = \operatorname*{argmin}_{\theta \in \Theta} D_{KL}(p \parallel p_\theta).$$

Proof. Following the derivation in (Shalev-Shwartz and Ben-David, 2014, sec. 24.1.2), it holds

$$\begin{aligned}
\mathbb{E}_{V \sim \mathcal{D}} \left[-\log p_\theta(v) \right] &= \int_{\Omega'} -p(v) \log p_\theta(v) dv \\
&= \int_{\Omega'} p(v) \log \frac{1}{p_\theta(v)} dv \\
&= \int_{\Omega'} p(v) \log \frac{1}{p_\theta(v)} + p(v) \log p(v) + p(v) \log \frac{1}{p(v)} dv \\
&= \int_{\Omega'} p(v) \log \frac{p(v)}{p_\theta(v)} dv + \int_{\Omega'} p(v) \log \frac{1}{p(v)} dv \\
&= D_{KL}(p \parallel p_\theta) + h(V),
\end{aligned}$$

where $h(V)$ denotes the (differential) entropy (Definition 2.12). Since $h(V)$ is independent of θ, the claimed equivalence holds. □

We return to information-theoretic considerations in the overview of deep generative models in chapter 5. Further, we leverage several of the presented properties to derive an alternative loss function in section 6.4.2. For an extensive treatment of information theory we refer to (Cover and Thomas, 2006) and for an advanced mathematical introduction to (Polyanskiy and Wu, 2015). After reviewing some fundamental goals of learning, we introduce deep neural networks in the subsequent section.

2.2 Deep Neural Networks

Deep learning had a major impact on many fields such as computer vision, natural language processing, speech recognition, reinforcement learning and artificial intelligence in general, see e.g. (LeCun et al., 2015). A cornerstone of deep learning is to compose the processing from raw inputs to targets via a cascade of differentiable transforms. This in turn enables learning in an end-to-end fashion via gradient-based approaches. Often, this concept is referred to as representation learning and is implemented by deep neural networks.

In this section, we introduce the main mathematical concepts used in the subsequent chapters. While there are plenty of resources that build up an intuition of deep neural networks, we focus on technical aspects and aim towards a brief introduction. For a broader discussion on deep learning, we refer to (Goodfellow et al., 2016).

2.2.1 Feedforward Neural Networks

Neural networks are usually defined as a composition of parametrized affine and non-linear activation functions. In this thesis, we focus on feedforward neural networks as defined below:

Definition 2.15 (Basic feedforward neural network)**.** *Let L denote the number of layers, then we call $F_\theta : \mathbb{R}^d \to \mathbb{R}^{d'}$ a (basic) feedforward neural network with L layers if*

$$F_\theta = f_\theta^L \circ \cdots \circ f_\theta^1,$$

where

$$f_\theta^i(x^i) = \phi^i(A^i x^i + b^i), \quad i \in [L]$$

denotes a layer. Further, $A^i \in \mathbb{R}^{m_i \times n_i}$ is called a linear layer, $b^i \in \mathbb{R}^m$ a bias and ϕ^i an elementwise non-linear function. This elementwise function $\phi^i : \mathbb{R}^{m_i} \to \mathbb{R}^{m_i}$ transforms each element separately as

$$\phi^i(x) = \begin{pmatrix} \phi^*(x_1) \\ \vdots \\ \phi^*(x_{m_i}) \end{pmatrix},$$

by using a non-linear activation function ϕ^*. Note, that the input/output dimension of F_θ fixes the dimension of $A^L \in \mathbb{R}^{d' \times n_L}$ and $A^1 \in \mathbb{R}^{m_1 \times d}$, while all inner dimension can be freely chosen but must match $m_i = n_{i+1}$. The vector $\theta \in \mathbb{R}^p$ will be called the parameters of the neural network and comprises all free entries of the linear layers and biases. Formally,

$$\theta = \left(\mathrm{op2par}(A^L), \mathrm{op2par}(b^L), \ldots, \mathrm{op2par}(A^1), \mathrm{op2par}(b^1)\right),$$

where $\mathrm{op2par}(A^i)$ *and* $\mathrm{op2par}(b^i)$ *is an operation-to-parameter mapping. The x^i are called activations of input x in layer i.*

Example 2.16 (Operation-to-parameter mapping)**.** *As there are many ways to parametrize the operations in a neural network (most prominently, dense and convolutional parametrization), we introduce the operation-to-parameter mapping* op2par *to formalize the relationship between parameters and operation. For example, consider the matrices*

$$A^1 = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \quad \text{and} \quad A^2 = \begin{pmatrix} a_1 & a_1 \\ a_2 & a_2 \end{pmatrix},$$

where the matrix A^2 has shared parameters.
Then, we have $\text{op2par}(A^1) = (a_{11}, a_{21}, a_{12}, a_{22})$ *and* $\text{op2par}(A^2) = (a_1, a_2)$*. Note that we also use* op2par *for the biases b^i since parameters can be shared for biases as well (e.g. in convolutional layers).*

Similarly to the above abstraction, one can think of a parameter-to-operation mapping which builds the operations like a matrix multiplication via a given parameter vector. In general, these considerations are an abstraction to describe the connection between parameters and operations in the neural network. The *architecture* of the neural network then comprises all hyper-parameters like the depth L, the width in each layer m_i and the implementation of the mappings between parameters and operations.

We further introduce the following:

Example 2.17 (Vectorization)**.** *Let $A \in \mathbb{R}^{m \times n}$ and* $\text{vec}(A)$ *denote the columnwise stacking of the matrix A into a vector $\mathbb{R}^{m \cdot n}$. For A^1 and A^2 from Example 2.16, it is*

$$\text{vec}(A^1) = (a_{11}, a_{21}, a_{12}, a_{22}) \quad \text{and} \quad \text{vec}(A^2) = (a_1, a_2, a_1, a_2).$$

Note, that $\text{op2par}(A^2) \neq \text{vec}(A^2)$ *since parameters are shared, while* $\text{op2par}(A^1) = \text{vec}(A^1)$*.*

Using above definitions and abstractions, we define:

Definition 2.18 (Multi-Layer perceptron)**.** *Let F_θ be as in Definition 2.15. If*

$$\text{op2par}(A^i) = \text{vec}(A^i), \quad \forall\, i \in [L],$$

we call F_θ a multi-layer perceptron (MLP). Furthermore, we call the matrices A^i fully-connected or dense.

The term fully-connected refers to viewing each output of a hidden layer f_θ^i as a neuron and considering the matrix A^i as the connection between neurons in layer i and $i-1$. For an intuition and connections of (artificial) neural networks to neuroscience, we refer to (Goodfellow et al., 2016).

The introduction of feedforward neural networks allows a generalization of linear regression (see section 2.1.2) and, more importantly for further considerations, a generalization of multi-class logistic regression (see Definition 2.7):

Definition 2.19 (Multi-class classification with neural networks)**.** *Let $x \in \mathbb{R}^d$ denote the inputs and let labels $y \in \{0,1\}^C$ (one-hot encoded) be categorically distributed as*

$$Cat(y \mid \mu) = \prod_{k=1}^{C} \mu_k^{\mathbb{I}[y_k=1]},$$

where $\mathbb{I}[\cdot]$ denotes the indicator function and μ_k is the probability that y is from class k. Multi-class classification with a feedforward neural network $F_\theta : \mathbb{R}^d \to \mathbb{R}^C$ can be formulated via

$$p_\theta(Y = \boldsymbol{k} \mid X = x) = \text{softmax}(F_\theta(x))_k = \frac{\exp(F_\theta(x)_k)}{\sum_{k'=1}^{C} \exp(F_\theta(x)_{k'})},$$

where $Y = \boldsymbol{k}$ denotes that the target is from class k.

As it turns out, above generalization using neural networks is very powerful from an approximation-theoretic point of view. It is regarded as an universal approximator of continuous functions on bounded domains when using at least one hidden layer, non-polynomial activation functions ϕ and the width is allowed to grow infinitely, see (Cybenko, 1989) for an early result or (Goodfellow et al., 2016) for an overview.

Besides the affine mappings, the non-linear activation function ϕ^* is an important property of a neural network. Following common activation functions will be discussed in this thesis:

- Rectified-linear unit (Glorot et al., 2011): $\text{ReLU}(x) = \max(x, 0)$
- Hyberbolic tangent: $\tanh(x)$
- Exponential linear unit (Clevert et al., 2016): $\alpha \geq 0$

$$\text{ELU}(x) = \begin{cases} x, & x > 0 \\ \alpha(e^x - 1), & \text{else} \end{cases}.$$

For an overview of most common activation functions, we refer to (Goodfellow et al., 2016). While this section covers basic feedforward neural networks, many variants are developed. One of the most prominent subtype of feedforward neural networks are convolutional neural networks, which will be introduced in the next section.

2.2.2 Convolutional Neural Networks

While MLPs are powerful models from an approximation-theoretic point of view, they do not incorporate any knowledge about the input structure. Especially images are very structured as neighboring pixels are often strongly correlated. A well-suited operation for such correlation structures are convolutions.

In this section, we take a rather technical approach to define convolutional layers on 2D spatial data with multiple channels. In particular, we follow the notation in (Sedghi et al.,

2019), which allows us to discuss the matrix structure underlying a convolutional layer in a precise manner. Moreover, we focus on the circular convolution which allows to transfer the convolution theorem from the continuous domain to discrete data. Further remarks about different variants of discrete convolution are given at the end of the section.

First, we start by considering only a single filter:

Definition 2.20 (2D circular discrete convolution)**.** *Let $X \in \mathbb{R}^{d\times d}$ be an input image and $\tilde{W} \in \mathbb{R}^{k\times k}$ be the convolutional filter. Further, let $W \in \mathbb{R}^{d\times d}$ be the zero-padded convolutional kernel, where $W_{ij} = \tilde{W}_{ij}$ if index $i \leq k$ and $j \leq k$ and $W_{ij} = 0$ else. Then, the circular discrete convolution corresponds to the operation*

$$Y_{ij} = \sum_{p=0}^{d-1}\sum_{q=0}^{d-1} X_{i+p \,\%\, d,\, j+q \,\%\, d} W_{p+1,q+1} \quad i,j \in [d],$$

where $\%$ denotes the modulo operation, with the exception that $i \,\%\, i = i$.

In order to write above convolution as a matrix-vector operation, we first need the definition of a circulant matrix:

Definition 2.21 (Circulant matrix)**.** *Let $a \in \mathbb{R}^d$ (row vector), then we call the matrix*

$$\operatorname{circ}(a) = \begin{pmatrix} a_1 & a_2 & \cdots & a_d \\ a_d & a_1 & \cdots & a_{d-1} \\ \vdots & \ddots & \ddots & \vdots \\ a_2 & a_3 & \cdots & a_1 \end{pmatrix}$$

a circulant matrix defined by row vector a. Thus, by definition it is $\operatorname{op2par}(\operatorname{circ}(a)) = a$.

Then, we can write the 2D circular discrete convolution as a matrix-vector product using a doubly block circulant matrix, see (Sedghi et al., 2019):

Lemma 2.22. *Let X, W be as in Definition 2.20. Further, let $\tilde{X} = \operatorname{vec}(X)$, $\tilde{Y} = \operatorname{vec}(Y)$ be the vectors obtained by stacking the columns of X, Y. Then,*

$$\tilde{Y} = A\tilde{X},$$

where $A \in \mathbb{R}^{d^2\times d^2}$ is a doubly block circulant matrix (see Definition 2.21)

$$A = \begin{pmatrix} \operatorname{circ}(W_{1,:}) & \operatorname{circ}(W_{2,:}) & \cdots & \operatorname{circ}(W_{d,:}) \\ \operatorname{circ}(W_{d,:}) & \operatorname{circ}(W_{1,:}) & \cdots & \operatorname{circ}(W_{d-1,:}) \\ \vdots & \ddots & \ddots & \vdots \\ \operatorname{circ}(W_{2,:}) & \operatorname{circ}(W_{2,:}) & \cdots & \operatorname{circ}(W_{1,:}) \end{pmatrix},$$

where $W_{i,:}$ denotes the i-th row of W. Thus, by definition it is $\operatorname{op2par}(A) = \operatorname{vec}(\tilde{W})$, *where* vec *denotes the vectorization of the 2D-matrix $\tilde{W}$ (see Example 2.17).*

Hence, a single filter on 2D spatial inputs already induces two hierarchies: circulant matrices due to circular convolution and block structure due to 2D inputs. Yet, in convolutional layers there is another dimension, the so-called channel dimension. Thus, we consider inputs $X \in \mathbb{R}^{m\times d\times d}$ with m channels. Furthermore, let n denote the number of output channels of the convolutional layer.

Definition 2.23 (Convolutional layer)**.** *Let $X \in \mathbb{R}^{m\times d\times d}$ and $W \in \mathbb{R}^{d\times d\times n\times m}$ denote the zero-padded 4D kernel tensor. Then, a convolutional layer implements the multi-channel convolution as*

$$Y_{cij} = \sum_{l=1}^{m}\sum_{p=0}^{d-1}\sum_{q=0}^{d-1} X_{l,i+p\,\%\,d,\,j+q\,\%\,d}\, W_{p+1,q+1,c,l} \qquad i,j \in [d],\ c \in [n],$$

where $\%$ denotes the modulo operation, with the exception that $i\,\%\,i = i$.

This operation can be written via matrix-vector products as follows, see (Sedghi et al., 2019):

Lemma 2.24 (Convolutional layer as matrix-vector product)**.** *Let X, W be as in Definition 2.23. Then, we can write a convolutional layer as a matrix-vector product*

$$\tilde{Y} = \tilde{A}\tilde{X},$$

where $\tilde{A} \in \mathbb{R}^{nd^2\times md^2}$ is a block matrix

$$\tilde{A} = \begin{pmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{n1} & \cdots & A_{nm} \end{pmatrix},$$

where each A_{ij} is a doubly circulant matrix from Lemma 2.22. Thus, by definition it is $\mathrm{op2par}(\tilde{A}) = \mathrm{vec}(\tilde{W})$*, where* vec *denotes the vectorization of the 4D-tensor $\tilde{W}$ (see Example 2.17).*

Hence, the multi-channel convolution of a convolutional layer introduces a third hierarchy into the operation. While above considerations are of rather technical nature, they are helpful to grasp several properties of convolutional layers:

- They allow to compute singular values efficiently in comparison to a naive computation on $\tilde{A}$, see (Sedghi et al., 2019).
- They serve to understand the construction of the matrix-vector product structure underlying convolutional layers. Hence, convolutional neural networks (CNNs), where (some) linear layers are implemented via convolutional layers, are sub-classes of feed-forward neural networks.
- Convolutional layers are highly structured by using parameter sharing, see also (Goodfellow et al., 2016). Hence, $\theta(\tilde{A}) \neq \mathrm{vec}(\tilde{A})$ as in MLPs. Instead, $\theta(\tilde{A}) = \mathrm{vec}(\tilde{W})$, where $\tilde{W}$ corresponds to the non-padded convolutional kernel.

In practice, however, convolutional layers are often modified in several ways. The most common modification from the circular convolution presented in this section, are different ways to handle image boundaries like *same/ valid convolution*. Furthermore, *strided convolutions* apply the convolutional kernel not at every spatial position as introduced above. The main idea of this operation is to downsample the spatial dimension. For an extensive discussion on different choices of convolutional layers, we refer to (Goodfellow et al., 2016).

While we introduced the most basic concepts of MLPs and CNNs, there are more building blocks in practice like:

- Pooling layers to downsample the spatial dimension, see (Goodfellow et al., 2016)
- Normalization layers like batch normalization (Ioffe and Szegedy, 2015) or spectral normalization (Miyato et al., 2018)
- Dropout layers, see (Srivastava et al., 2014).

To summarize, we have introduced deep neural networks as a generalization of linear models (Def. 2.19). Furthermore, the goal of learning was formulated using a risk function (Def. 2.1) which needs to be approximated by an empirical risk (Def. 2.2). Yet, there is still one missing piece: how to find an empirical risk minimizer (Def. 2.3)? This learning step will be discussed in the following section.

2.2.3 Learning Neural Networks

We begin by considering the empirical risk of hypothesis $F \in \mathcal{F}$, which was defined in Definition 2.3 as

$$L_{\mathcal{T}}(F) = \frac{1}{N} \sum_{i=1}^{N} \ell(F, v^{(i)}),$$

where $\mathcal{T} = \{v^{(i)}\}_{i=1}^{N}$ is an i.i.d. training set with $v^{(i)} \in \Omega'$ and a loss ℓ as defined in Definition 2.1. Since neural networks are parametric models with $\theta \in \Theta$, we consider ERM in parameter space Θ instead of minimization directly in the hypothesis space $\mathcal{F}$.

Definition 2.25 (ERM with neural networks)**.** *Let $\mathcal{T}$ denote a training set and F_θ a neural network. Then, when assuming an argmin*

$$\theta^* \in \operatorname*{argmin}_{\theta \in \Theta} L_{\mathcal{T}}(F_\theta) = \operatorname*{argmin}_{\theta \in \Theta} \frac{1}{N} \sum_{i=1}^{N} \ell(F_\theta, v^{(i)})$$

exist, we define the parameter θ^* as the empirical risk minimizer for a neural network F_{θ^*} within the parameter space and model architecture.

The most common approach to finding such a parameter θ^* is via gradient-based methods. However, following often encountered situation makes computing the full-gradient $\nabla_\theta L_{\mathcal{T}}(\theta)$ prohibitive in practice:

- high-dimensional inputs like high-resolution images, spectral data, sound etc.
- large number of training points like 1.2 Mio. images in the ImageNet dataset (Deng et al., 2009)
- large neural networks like typical CNN-based image classifiers.

Thus, a stochastic approximation is usually preferred, see (Goodfellow et al., 2016):

Definition 2.26 (Stochastic gradient descent)**.** *Let $L_{\mathcal{T}}(F_\theta)$ denote the training loss, which is differentiable with respect to θ. Further, let $\hat{L}_{\mathcal{T}}(F_\theta)$ an unbiased estimate of the training loss, i.e.*

$$\mathbb{E}\left[\hat{L}_{\mathcal{T}}(F_\theta)\right] = L_{\mathcal{T}}(F_\theta). \tag{2.1}$$

Furthermore, let $\theta^{[0]} \in \Theta$ be an initial parameter vector. Then, stochastic gradient descent (SGD) is defined via the update rule

$$\theta^{[k+1]} = \theta^{[k]} - \lambda \, \nabla_\theta \hat{L}_{\mathcal{T}}^{[k]}(F_{\theta^{[k]}}) \quad k \in [K],$$

where K denotes the number of iterations, $\hat{L}_{\mathcal{T}}^{[k]}$ is an unbiased estimate depending on iteration k and $\lambda > 0$ is the learning rate.

Remark 2.27. *(Stochastic estimation of gradient) Note, that we deliberately did not specify the random variable over which the expectation in (2.1) is taken. Most commonly, the i.i.d. training set $\mathcal{T}$ is randomly divided into M disjoint mini-batches $\tilde{\mathcal{T}}_i$ such that*

$$\bigcup_{i=1}^{M} \tilde{\mathcal{T}}_i = \mathcal{T}.$$

Then, if $\mathcal{D}$ is a uniform distribution over $i \in [M]$, it holds

$$\mathbb{E}_{i\sim\mathcal{D}}\left[L_{\tilde{\mathcal{T}}_i}(F_\theta)\right] = L_{\mathcal{T}}(F_\theta).$$

This approach is also called mini-batch SGD. Once all mini-batches were used for a gradient update, we call this set of SGD-steps an epoch. Furthermore, in chapter 5 we will look into a loss function that requires estimation techniques to enable an efficient evaluation even for a fixed mini-batch.

While SGD is at the core of most optimization algorithms in deep learning, there are plenty of variants and extensions. Most popular approaches are for instance either SGD enhanced with momentum or adaptive algorithms like ADAM (Kingma and Ba, 2014). Those adaptive algorithms employ a learning rate λ separately for each entry of θ based on statistics of previous updates. For a discussion of optimization algorithms in deep learning, we refer to (Goodfellow et al., 2016).

Besides choosing an optimization algorithm and a corresponding stochastic estimation of the loss, being able to compute the gradient $\nabla_\theta L_{\tilde{\mathcal{T}}_i}(F_\theta)$ is crucial. An efficient algorithm for

computing this gradient is backpropagation, see (Goodfellow et al., 2016) for an extensive introduction. This algorithm can be analyzed via reverse-mode Automatic Differentiation (AD) on computational graphs, which forms the basis of modern deep learning frameworks like TensorFlow (Abadi et al., 2015) or PyTorch (Paszke et al., 2017).

Automatic differentiation is a set of techniques to numerically evaluate the exact gradient up to machine precision (Deisenroth et al., 2019). AD works by building a computational graph using intermediate values and the chain rule of differentiation to compute derivatives. More formally, see (Deisenroth et al., 2019, sec. 5.6.2):

Definition 2.28 (Reverse-mode Automatic Differentiation (AD))**.** *Let $f : \mathbb{R}^d \to \mathbb{R}$ and $x \in \mathbb{R}^d$ be the inputs, x_i for $i \in [d]$ denoting each individual entry. Further, let z_l for $l \in [D]$ denote intermediate features and let $y = f(x)$ denote the output variable. Then, assume that the computation graph can be expressed as*

$$\text{For } l \in [D], \text{ compute:} \qquad z_l = g_l(x_{Pa(z_l)}), \tag{2.2}$$

where $g_l(\cdot)$ are elementary functions like $\sin, \cos, \exp$ and $x_{Pa(z_l)}$ are the parent-nodes of the variable z_l in the computational graph. Given these functions, we can compute the derivative of the function step-by-step with the chain rule. Recall, that $\frac{\partial f}{\partial y} = 1$. For other variables x_i, we apply the chain rule

$$\tilde{x}_i := \frac{\partial f}{\partial x_i} = \sum_{\{z_L | x_i \in Pa(z_l)\}} \frac{\partial f}{\partial z_l} \frac{\partial z_l}{\partial x_i} = \sum_{\{z_L | x_i \in Pa(z_l)\}} \frac{\partial f}{\partial z_l} \frac{\partial g_l}{\partial x_i}, \tag{2.3}$$

where $Pa(z_l)$ is the set of parent nodes of z_l in the computational graph. The computation in (2.2) corresponds to the forward pass of a neural network f and (2.3) to the backward pass, hence the above is called reverse-mode AD.

Remark 2.29. *Besides reverse-mode AD, there is also forward-mode AD which is more suited to functions $f : \mathbb{R}^d \to \mathbb{R}^{d'}$ where $d' > d$. Since $d' \ll d$ in most ML-tasks, or even $d' = 1$ if we compute derivatives with respect to a loss function, reverse-mode is more common in machine learning (Baydin et al., 2018).*

Besides allowing to calculate $\nabla_\theta L_{\tilde{\mathcal{T}}_i}(F_\theta)$ in an efficient and exact manner, reverse-mode AD can compute vector-Jacobian products efficiently, see (Baydin et al., 2018): Let $f : \mathbb{R}^d \to \mathbb{R}^{d'}$ and let $J_f(x)$ denote the Jacobian of f at x. Then for $f(x) = (y_1, \dots, y_{d'})$ and column-vector $v \in \mathbb{R}^{d'}$, the vector-Jacobian product

$$v^T J_f(x) = \begin{pmatrix} v_1 & \dots & v_{d'} \end{pmatrix} \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \dots & \frac{\partial y_{d'}}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_l} & \dots & \frac{\partial y_{d'}}{\partial x_l} \end{pmatrix}$$

can be computed in a matrix-free/ implicit manner by setting $\tilde{x} = v$ as the initialization for the reverse-mode phase. This operations requires only a single reverse-mode AD call, whereas a direct computation via instantiating $J_f(x)$ would require d' reverse-mode AD calls. While these considerations may appear rather technical at this stage, this efficient computation of vector-Jacobian products will be crucial in chapter 5.

To summarize, learning neural networks builds upon two main pillars: First, stochastic gradient descent (SGD) allows efficient optimization of the weights. Second, gradients of even complex neural networks can be efficiently computed via backpropagation or more generally reverse-mode automatic differentiation.

So far we have introduced the main ideas of learning and deep neural networks, which serve as the foundation of this thesis. The next section aims to bridge these fundamental concepts with the research contributions in the subsequent chapters.

2.2.4 Goals of Thesis and Overview

The universal function approximation theorem (Cybenko, 1989) claims that even basic feedforward neural networks (Definition 2.15) are able to represent any continuous function on a bounded domain. Yet, modern deep neural network architectures like Residual Networks (He et al., 2016) or Universal Transformers (Dehghani et al., 2019) move further and further away from this vanilla architecture.

Following this trend, the goal of this thesis is to study non-standard designs of neural network architectures in a principled manner. In particular, we will focus on two design principles:

- Design of invertible neural networks
- Design of neural networks architectures based on domain knowledge.

In mathematical terms, this thesis focuses on certain hypothesis spaces $\mathcal{F}$, which serve as an inductive bias for empirical risk minimization

$$F^* \in \operatorname*{argmin}_{F \in \mathcal{F}} L_{\mathcal{D}}(F).$$

In the first part, we study neural networks F which build a hypothesis space of (partly) invertible functions. We begin by analyzing the invertibility of standard architectures in chapter 3. In particular, preimages of ReLU-layers and inverse stability are investigated from a theoretical and empirical perspective. This stability analysis then lead us to the idea of invertible residual networks in chapter 4. Based on invertible residual networks, chapter 5 discusses the application to maximum likelihood based generative modeling. The main part on invertible networks then ends with a reverse view on adversarial examples in chapter 6. In that chapter, a modification of common loss functions $L_{\mathcal{D}}(F)$ based on information-theoretic considerations is proposed.

After an in-depth analysis of invertible neural networks, chapter 7 introduces a convolutional neural network architecture which reflects the domain knowledge of mass spectra obtained from Imaging Mass Spectrometry measurements. This architecture is then applied to tumor classification and compared to other approaches. Finally, a conclusion summarizes the thesis and discusses future work.

Chapter 3

Invariance and Inverse Stability under ReLU

"There is nothing more practical than a good theory" - Kurt Lewin.

This chapter analyzes standard feedforward networks through the lens of invertibility. In particular, we link invariance to the preimage of activations of ReLU-layers and robustness to inverse stability. These insights will motivate the inclusion of additional structure in subsequent sections, which enables control over the preimage and inverse stability of neural networks.

This section is mainly based on the following article:

> **Jens Behrmann**, Sören Dittmer, Pascal Fernsel, Peter Maass: *Invariance and inverse stability under ReLU*, 2019, (submitted to IEEE Transactions on Neural Networks and Learning Systems)

3.1 Introduction and Motivation

Invariance and stability/robustness are two of the most important properties characterizing the behavior of a neural network. Due to growing requirements like robustness to adversarial examples (Szegedy et al., 2014) and the increasing use of deep learning in safety-critical applications, there has been a surge in interest in these properties. They are key mechanisms for dealing with uninformative properties of the input (Achille and Soatto, 2018; Mallat, 2016) and are studied from the information-theoretic perspective in form of the loss of information about the input (Tishby and Zaslavsky, 2015; Saxe et al., 2018).

Invariance and stability are also tightly linked to robustness against adversarial attacks (Cisse et al., 2017; Tsuzuku et al., 2018; Simon-Gabriel et al., 2018), generalization (Sokolić et al., 2017; Gouk et al., 2018) and even the training of Generative Adversarial Networks (Miyato et al., 2018). In general, stability is studied via two basic properties:

- Locally via analyzing a norm of the Jacobian, see e.g. (Sokolić et al., 2017; Simon-Gabriel et al., 2018).
- Globally via bounding Lipschitz constants, see e.g. (Cisse et al., 2017; Miyato et al., 2018; Tsuzuku et al., 2018).

From a high-level perspective, each of these approaches studies an upper bound on stability. Both the Lipschitz constant and Jacobian norm quantify the highest possible change under a perturbation with a given magnitude. We, unlike the approaches above, aim to broaden our understanding by analyzing the lowest possible change under a perturbation.

More formally, we study which perturbations Δx that do not (or only little) affect the outcome of a network F. Note that we dropped the dependence of F on parameter θ, since we are not concerned with learning in this chapter. Our analysis considers a given input data point x and investigates the Δx's, such that

$$\begin{aligned} & F(x) = F(x + \Delta x) && \text{(invariant)} \\ \text{or} \quad & \|F(x) - F(x + \Delta x)\| \leq \varepsilon && \text{(stable)}, \end{aligned}$$

where a small $\varepsilon > 0$ is given. In particular, if F is invariant to perturbations Δx, then x and $x + \Delta x$ lie in the preimage of the output $z = F(x)$, i.e. F is not uniquely invertible. Robustness towards large perturbations induces an instable inverse mapping as small changes in the output can be due to large changes in the input.

While these properties may be influenced by many factors, we simplify the setting by:

(i) Focusing on vanilla feedforward networks with ReLU-activations (see Def. 2.15)

(ii) Only providing conditions, when certain properties are fulfilled. Thus neglecting the influence of the data distribution and learning algorithm.

In particular, we characterize the preimage of ReLU-layers as a single point (singleton), finite (bounded) or infinite (unbounded). Further, we study the stability of the linearization of rectifier networks via its singular values.

To illustrate these locally changing properties and to demonstrate their tight connection, we visualize the behavior on a synthetic problem in Figure 3.1. As ReLU-layers are piecewise linear, the local behavior is constant on convex polytopes (Raghu et al., 2017). Further, the regions with infinite/finite preimages correspond to regions with condition number of one or zero, while singleton preimages link to condition numbers larger than one. Thus, both properties are tightly connected and investigating one property alone would yield only an incomplete picture.

3.2 Related Work

While analyzing invariance and robustness properties is a major topic in theoretical treatments of deep networks (Mallat, 2016), studying it via the inverse is less common. Several

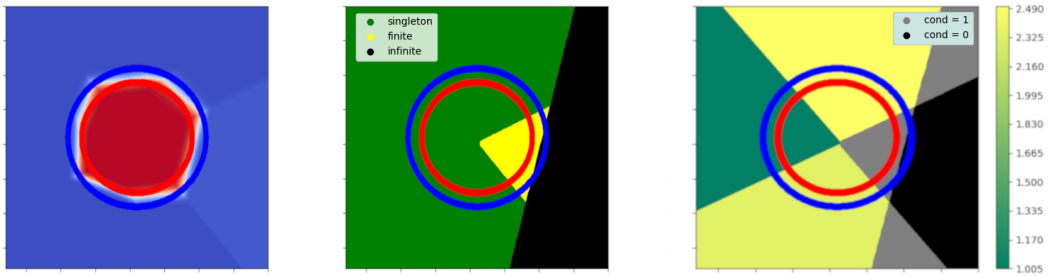


Figure 3.1: Left: Prediction of a small ReLU-network (one hidden layer with 3 neurons) trained to distinguish samples from two circles. **Middle:** Characterization of the preimage of first layer activations into unbounded (infinite), compact (finite) or unique (a single point). **Right:** Condition of the linearization of the first layer at each point.

works like (Mahendran and Vedaldi, 2015), (Mahendran and Vedaldi, 2016) or (Dosovitskiy and Brox, 2016) focus on reconstructing inputs from features of convolutional neural networks (CNNs) to visualize the information content of features. Instead, we investigate potential mechanisms affecting the invertibility.

(Carlsson et al., 2017) give a first geometrical view on the shape of preimages of outputs from ReLU layers, which is directly related to the question of injectivity of the mapping under ReLU. (Shang et al., 2016) analyze the reconstruction property of cReLU (concatenated ReLU); however, the more general situation of using the standard rectifier is not studied. A notable other line of work assumes random weights in order to derive guarantees for invertibility, see (Gilbert et al., 2017) or (Arora et al., 2015), whereas we focus on the preimage of ReLU-activations without assumptions on the weights.

Two main resources for our view of rectifier networks as piecewise linear models are (Montufar et al., 2014) and (Raghu et al., 2017). Closest to our approach is the work of (Bruna et al., 2014) on global statements of injectivity and stability of a single layer including ReLU and pooling. The authors focus on global injectivity and stability bounds via combinatorial statements over all configurations attainable by ReLU and pooling. These conditions are valid on the entire input space, while the restriction to parts of the input space may yield situations far from these worst-case conditions.

3.3 Pre-Images of ReLU Layers

In this section, we analyze different kinds of preimages of a ReLU-layer and investigate under which conditions the inverse image of a given point is a singleton (a set containing exactly one element) or has finite/infinite volume.

The analysis of preimages of a given output can be studied in two ways: 1) study single layers separately or 2) multiple layers at once. Note that, the concatenation of two injective functions is again injective, while a non-injective function followed by an injective function

Figure 3.2: Visualization of omnidirectionality: gray lines are hyperplanes with normal vectors (arrows) from the rows of A and translation b. **(a)** Omnidirectional tuple (A, b) for $p \in \mathbb{R}^2$, as hyperplanes intersect in p and normal vectors are omnidirectional. **(b)** Intersection in p, but vector-free halfspaces (hence, not omnidirectional). **(c)** Intersection in p, but vector-free halfspaces (hence, not omnidirectional). **(d)** Hyperplanes do not intersect in a point, but normal vectors are omnidirectional.

is non-injective. Hence, studying single layers is crucial. We therefore develop a theory for the case of single layers in this section. Notice that in case of multiple layers one is also required to investigate the image space of the previous layer.

We will focus our study on the most common activation function: ReLU. One of its key features is the non-injectivity, which is caused by the constant mapping on the negative half space. It provides neural networks with an efficient way to deploy invariances. Basically all other common activation functions are injective, which would lead to a straightforward analysis of the preimages. However, injective activations like ELU (Clevert et al., 2016) and Leaky ReLU (Maas et al., 2013) only swap the invariance for robustness, which in turn leads to the problem of having instable inverses. This question of stability will be analyzed in more detail in Section 3.4.

We start by introducing one of our main tools:

Definition 3.1 (Omnidirectionality)**.** *We define following geometric properties of linear and affine mappings:*

i) *$A \in \mathbb{R}^{m \times n}$ is called omnidirectional if every linear open halfspace in $\mathbb{R}^n$ contains a row of A, i.e. for every given $x \in \mathbb{R}^n \setminus \{0\}$ there exists an index $i \in [m]$, such that $\langle a_i, x \rangle > 0$, where a_i is a row of A and $\langle \cdot, \cdot \rangle$ denotes the Euclidian inner product.*

ii) *$A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ are called omnidirectional for the point $p \in \mathbb{R}^n$ if A is omnidirectional and $b = -Ap$.*

Consider a matrix A which is omnidirectional: Then, for every direction of a hyperplane through the origin forming two halfspaces, there is a vector from the rows of A inside each open halfspace. Hence, the term *omnidirectional* (see Figure 3.2 for an illustration). Note that the hyperplanes are due to ReLU as it maps the open halfspace to positive values and the closed halfspace to zero. A straightforward way to construct an omnidirectional matrix is by taking a matrix whose rows form a spanning set $\mathcal{F}$ and use the vertical concatenation of $\mathcal{F}$ and $-\mathcal{F}$. This idea is related to cReLU (Shang et al., 2016).

Omnidirectionality is directly related to the ReLU-layer preimages and will provide us with a method to characterize their volume (see Theorem 3.3). To analyze such inverse

images, we consider $y = \mathrm{ReLU}(Ax+b)$ for a given output $y \in \mathbb{R}^m$ with $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$ and $x \in \mathbb{R}^n$. If we know A, b and y, we can write the equation as the mixed linear system

$$A|_{y\succ 0}x + b|_{y\succ 0} = y|_{y\succ 0} \tag{3.1}$$
$$A|_{y=0}x + b|_{y=0} \preceq 0, \tag{3.2}$$

where $A|_{y\succ 0}$ denotes the restriction of the matrix A to the rows, which are specified by the index set $\{i : y_i > 0\}$. Further, the notation $y \preceq 0$ refers to the elementwise relation for vectors $y \in \mathbb{R}^d$.

Remark 3.2. *It is possible to enrich the mixed system to include conditions/priors on x (e.g. $x \in \mathbb{R}^n_{\geq 0}$).*

The inequality system in (3.2) links its set of solutions, and therefore the volume of the preimages of the ReLU-layer, with the omnidirectionality of A and b. For this we define

$$\overline{A} := AO^T,$$

where $O \in \mathbb{R}^{k\times n}$ denotes an orthonormal basis of the null-space $\mathcal{N}(A|_{y\succ 0})$ with $k := \dim \mathcal{N}(A|_{y\succ 0})$. Further, we define

$$\overline{b} := b|_{y\preceq 0} + A|_{y\preceq 0}(P_{\mathcal{N}(A|_{y\succ 0})^\perp} x),$$

where P_V denotes the orthogonal projection into the closed space V. By using these definitions, we are ready to state the main result in this section:

Theorem 3.3 (Preimages of ReLU-layers)**.** *Let $\overline{A}, \overline{b}$ and $k = \dim \mathcal{N}(A|_{y\succ 0})$ be as above. The preimage of a point y under a* ReLU*-layer is*

- ***i)*** *for $k = 0$ a singleton.*
- ***ii)*** *for $k > 0$ a singleton, if and only if there exists an index set I for the rows of $\overline{A}$ and $\overline{b}$, such that $(\overline{A}|_I, \overline{b}|_I)$ is omnidirectional for some point $p \in \mathbb{R}^k$.*
- ***iii)*** *for $k > 0$ a compact polytope with finite volume, if and only if $\overline{A}$ is omnidirectional.*

Proof. We refer to (Behrmann et al., 2018a***) since this proof requires the introduction of some technical Lemmas. □

Thus, omnidirectionality allows (in theory) to distinguish whether the inverse image of a ReLU-layer is a singleton, a compact polytope or has infinite volume. In (Behrmann et al., 2018a***), omnidirectionality is further linked to the convex hull spanned by the rows of A. This allows to derive an algorithm to check uniqueness of the preimage based on linear programming.

After discussing these qualitative insights into the preimage of ReLU-layers, we next turn to the study of inverse stability to extend our understanding of the invertibility of ReLU-networks.

3.4 Inverse Stability under ReLU

3.4.1 Theoretical Analysis

In this section, we analyze the robustness of rectifier MLPs (see Definition 2.18) against large perturbations via studying the stability of the inverse mapping. In particular, we study the effect of ReLU on the singular values of the linearization of network F. While the linearization of a network F at some point x only provides a first impression on its global stability properties, the linearization of ReLU networks is exact in some neighborhood due to its piecewise-linear nature (Raghu et al., 2017). In particular, the input space $\mathbb{R}^d$ of a rectifier network F is partitioned into convex polytopes P_F, corresponding to a different affine function on each region (see Figure 3.1). Hence, for each polytope P in the set of all input polytopes P_F, the network F can be simplified as $F(x) = A_P x + b_P$ for all $x \in P$.

In particular, each of the linearized matrices A_P can be written via a chain of weight matrix multiplications that incorporates the effect of ReLU. To this end, the following definition from (Bruna et al., 2014) introduces 1) admissible index sets that formalize all possible local behaviors and 2) diagonal matrices to locally model the effect of ReLU, see (Wang et al., 2016):

Definition 3.4 (Admissible index sets, ReLU as diagonal matrix)**.** *We introduce the following notation:*

(i) An index set I^l for a layer l is admissible if

$$\bigcap_{i \notin I^l} \{x^l : \langle x^l, a_i^l \rangle > -b_i\} \cap \bigcap_{i \in I^l} \{x^l : \langle x^l, a_i^l \rangle \leq -b_i\} \neq \emptyset,$$

where a_i^l is a row of A^l.

(ii) Further, let D_I denote a diagonal matrix with $(D_I)_{ii} = 1$ for $i \notin I$ and $(D_I)_{ii} = 0$ for $i \in I$, where I is an admissible index set. Using this notation, the mapping of pre-activation $z \in \mathbb{R}^d$ under ReLU *can be written as*

$$\mathrm{ReLU}(z) = D_I\, z \qquad \text{with } I = \{i \in [d] : z_i \leq 0\}.$$

Thus, the linearization A_P of a network with L layers is a matrix chain

$$A_P = A^L D_{I^{L-1}} A^{L-1} \cdots D_{I^1} A^1,$$

where A^l are the weight matrices of layer l and

$$I^l := \{i \in [d_l] : (A^l x^{l-1} + b^l)_i \leq 0\}.$$

Of special interest for a stability analysis is the range of possible effects by the application of the rectifier. Since the effect by ReLU corresponds to the application of D_I for admissible I, we now turn to studying the changes of the singular values of a general matrix A compared to $D_I\, A$. For example, the matrix A could represent the chain of matrix products up to pre-activations in layer l. Then, the effect of ReLU can be globally upper bounded:

Lemma 3.5 (Global upper bound for largest and smallest singular value)**.** *Let σ_l be the singular values of $D_I A$. Then for all admissible index sets I, the smallest non-zero singular value is upper bounded by* $\min\{\sigma_l : \sigma_l > 0\} \leq \tilde{\sigma}_k$*, where $k = N - |I|$ and $\tilde{\sigma}_1 \geq ... \geq \tilde{\sigma}_N > 0$ are the non-zero singular values of A.*
Furthermore, the largest singular value is upper bounded by $\max\{\sigma_l : \sigma_l > 0\} \leq \tilde{\sigma}_1$.

Proof. The upper bound on the largest singular value is trivial, as ReLU is contractive as $\|D_I Ax\|_2 \leq \|Ax\|_2$ for all I and $x \in \mathbb{R}^n$.
To prove the upper bound for the smallest singular value, we assume

$$\sigma_M := \min\{\sigma_l : \sigma_l > 0\} > \tilde{\sigma}_k \tag{3.3}$$

and aim to produce a contradiction. Consider all singular vectors $\tilde{v}_{k^*}$ with $k^* \geq k$ from matrix A. It holds for all $\tilde{v}_{k^*}$

$$\tilde{\sigma}_k \geq \tilde{\sigma}_{k^*} = \|A\tilde{v}_{k^*}\|_2 \geq \|D_I A\tilde{v}_{k^*}\|_2, \tag{3.4}$$

as D_I is a projection matrix and thus non-expansive. As

$$\sigma_M = \min_{\substack{\|x\|_2=1 \\ x \in \mathcal{N}(D_I A)^\perp}} \|D_I Ax\|_2,$$

all $\tilde{v}_{k^*} \notin \mathcal{N}(D_I A)^\perp$. Otherwise, a $\tilde{v}_{k^*}$ would be a minimizer by estimation 3.4, which would violate the assumption 3.3.

Due to $\mathcal{N}(D_I A)^\perp \oplus \mathcal{N}(D_I A) = \mathbb{R}^n$, it holds $\tilde{v}_{k^*} \in \mathcal{N}(D_I A)$. As $\tilde{v}_{k^*}$ are orthogonal, $\dim(span(v_{k^*})) = |I| + 1$ (note: $k^* = k, ..., N$ and $k = N - |I|$, thus there are $|I| + 1$ singular vectors v_{k^*} in total). Furthermore, $\tilde{v}_{k^*}$ are not in $\mathcal{N}(A)$ by definition (corresponding singular values are strictly positive).
Hence, the nullspace of D_I must have $\dim(\mathcal{N}(D_I)) \geq |I|+1$. But D_I is the identity matrix except $|I|$ zeros on the diagonal, thus $\dim(\mathcal{N}(D_I)) = |I|$, which yields a contradiction. □

Lemma 3.5 analyzes the best case scenario with respect to the highest value of the smallest singular value. While this would yield a more stable inverse mapping, one needs to keep in mind that $\mathcal{N}(A_P)$ grows by the corresponding elimination of rows via D_I. Moreover, reaching this bound is very unlikely as it requires the singular vectors to perfectly align with the directions that collapse due to D_i. Thus, we now turn to study effects which could happen locally for some input polytopes P.

An example of a drastic effect through the application of ReLU is depicted in Figure 3.3. Since one vector is only weakly correlated to the removed vector and the situation is overdetermined, removing this feature for some inputs x in the blue area leaves over the strongly correlated features. While the two singular values of the 3-vectors-system were close to one, the singular vectors after the removal by ReLU are badly ill-conditioned. As many modern deep networks increase the dimension in the first layers, redundant situations as in Figure 3.3 are common, which are inherently vulnerable to such phenomena. For example, (Rodríguez et al., 2017) proposed a regularizer to avoid such strongly correlated features. The following lemma formalizes the situation exemplified before:

Figure 3.3: Removal of vectors due to ReLU (red crosses) for the marked points x. The remaining vectors are only weakly correlated to the removed one, thus yielding an unstable inverse. **Left:** Unbiased case with $b = 0$, **Right:** Biased case with $b \neq 0$.

Lemma 3.6 (Removal of weakly correlated rows)**.** *Let $A \in \mathbb{R}^{m \times n}$ with rows a_j and $I \subseteq [m]$. For a fixed $k \in I$ (I admissible) let $a_k \in \mathcal{N}(D_I A)^\perp$, where $\mathcal{N}$ denotes the null-space. Moreover, let*

$$\forall j \notin I : |\langle a_j, a_k \rangle| \leq c \frac{\|a_k\|_2}{\sqrt{M}}, \tag{3.5}$$

with $M = m - |I|$ and constant $c > 0$. Then for the singular values $\sigma_l \neq 0$ of $D_I A$, it holds

$$0 < \sigma_K = \min\{\sigma_l : \sigma_l \neq 0\} \leq c.$$

Proof. Consider $v = \frac{a_k}{\|a_k\|_2}$. Then,

$$(D_I A v)_k = 0,$$

since $k \in I$ (k-th row of D_I is zero). Furthermore, for all $j \neq k$ it holds by condition 3.5

$$(D_I A v)_j = \frac{\langle a_k, a_j \rangle}{\|a_k\|_2} \leq \frac{|\langle a_k, a_j \rangle|}{\|a_k\|_2} \leq \frac{c}{\sqrt{M}}.$$

Hence,

$$\|D_I A v\|_2 = \sqrt{\sum_{j \notin I} \left(\frac{\langle a_k, a_j \rangle}{\|a_k\|_2} \right)^2} \leq \sqrt{M \left(\frac{c}{\sqrt{M}} \right)^2} = c.$$

As $a_k \in \mathcal{N}(D_I A)^\perp$, $v \in \mathcal{N}(D_I A)^\perp$ as well. Thus,

$$\sigma_K = \min_{\substack{\|x\|_2 \\ x \in \mathcal{N}(D_I A)^\perp}} \|D_I A\|_2 \leq \|D_I A v\|_2 \leq c.$$

□

Lemma 3.6 provides an upper bound on the smallest singular value, given a condition on the correlation of all a_j and a_k. However, the condition 3.5 depends on the number M of remaining rows a_j. Hence, in a highly redundant setting, even after removal by ReLU (large N), c needs to be large such that the correlation fulfills the condition. Yet, in this case the upper bound on the smallest singular value, given by c, is high.

Effect under multiple layers: For the effect of ReLU applied to multiple layers, we are particularly interested in following questions:

- Can the application of another layer have a pre-conditioning effect yielding a stable inverse?
- What happens when we only compose orthogonal matrices which have stable inverses?

Note e.g. that a way to enforce an approximate orthogonality constraint was proposed for CNNs in (Cisse et al., 2017), however only for the filters of the convolution. For both situations the answer is similar: the nonlinear nature of ReLU induces locally different effects. Thus, if we choose a pre-conditioner A^l for a specific matrix A_P^{l-1}, it might not stabilize the matrix product for matrices $A_{P^*}^{l-1}$ corresponding to different input polytopes P^*.

For the case of composing only orthogonal matrices, consider a network up to layer $l-1$, where the linearization A_P^{l-1} has orthogonal columns (assume the network gets larger, thus A_P^{l-1} has more rows than columns). Then, the application of ReLU as in

$$A^l D_{I^l} A_P^{l-1}$$

removes the orthogonality property of the rows of A_P^{l-1}, if setting entries in the rows from I^l to zero results in non-orthogonal columns (which is likely when considering dense matrices). Hence, $D_{I^l} A_P^{l-1}$ is not orthogonal for some I^l. In this case, the matrix product $A^l D_{I^l} A_P^{l-1}$ is not orthogonal, which results in decaying singular values.

This is why, even when especially designing the network by e.g. orthogonal matrices, stability issues with respect to the inverse arise. To conclude this section, we remark that the presented results are rather of a *qualitative* nature showcasing effects of ReLU on the singular values. Yet, the analysis does not require any assumptions and is thus valid for any feedforward network. Note that this includes CNNs without pooling as convolutional layer can be expressed via matrix-vector products (see Definition 2.23). To illustrate some *quantitative* effects, we provide numerical examples in the subsequent subsection.

3.4.2 Numerical Analysis

In this section, we show how the previously discussed theoretical stability properties can be examined for a given network. In particular, we conduct experiments on CIFAR10 (Krizhevsky and Hinton, 2009) using two baseline CNNs (architectures in Table 3.1) using the standard training setups from Keras (Chollet et al., 2015) (example `cifar10_cnn`). Our CNNs use only strides instead of pooling and use no residual connections and normalization layers. Thus, the architectures fit to the theoretical study as the strided discrete convolution can be written as a matrix-vector multiplication (see Definition 2.23).

Singular values over multiple layers: Most interesting is the development of singular values over multiple layer as several effects are potentially at interplay. Figure 3.4 shows the evolution of all singular values in convolutional layers (layers 1-6, after application of ReLU). While the shape of the curve is similar for layer 1-5, it can be seen that the largest singular value grows. On the other hand, the small singular values decrease significantly.

Table 3.1: Overview of WideCIFAR and ThinCIFAR architectures.

Index	Type	kernel size	stride	# feature maps	# output units
0	Input layer	-	-	3	
1	Convolutional layer	(3,3)	(1,1)	32 / 32	-
2	Convolutional layer	(3,3)	(2,2)	64 / 32	-
3	Convolutional layer	(3,3)	(1,1)	64 / 16	-
4	Convolutional layer	(3,3)	(1,1)	32 / 16	-
5	Convolutional layer	(3,3)	(1,1)	32 / 16	-
6	Convolutional layer	(3,3)	(2,2)	64 / 32	-
7	Dense layer	-	-	-	512
8	Dense layer (softmax)	-	-	-	10

Note that this growth of the largest singular values is in line with observations in the adversarial examples literature, see e.g. (Szegedy et al., 2014). While many defense strategies like (Cisse et al., 2017) or (Jia, 2017) focus on the largest singular value, the behavior of the smaller singular values is usually not studied.

Relationship between stability and invariance: While invariance is characterized by zero singular values, the condition number only takes non-zero singular values into account, see e.g. layer 6 in Figure 3.4. This tight relationship is further investigated in Figure 3.4, which compares the output size, the condition number and the number of non-zero singular values vs. the layers for WideCIFAR and ThinCIFAR. In combination with lower output dimension, ReLU has a different effect for ThinCIFAR. The number of singular values decreases in layer 5, which cuts off the smallest singular values, resulting in a lower condition number. Yet, there are more invariant directions within the corresponding linear region. For a visual comparison on the synthetic example see Figure 3.1.

Computational costs and scaling analysis: First, we remark that the linearization of a network F for an input point x^0 can be computed via backpropagation. In particular, reverse-mode automatic differentiation requires d'-calls, where d' is the output dimension of F (see section 2.2.3 for more details). After acquiring the linearization A_P, computing the full singular value decomposition (SVD) scales cubically with the dimension of A_P. Especially, early CNN-layers have high dimensional outputs which may cause memory issues when computing the entire SVD. We thus choose a small CNN trained on CIFAR10 as these inputs are only of size $32 \times 32 \times 3$. To scale this analysis up to e.g. ImageNet with VGG-networks, a restriction to a window of the input image is necessary to reduce the complexity of the full SVD especially for early layers. See (Jacobsen et al., 2018), where the linearization A_P was acquired only for an input window. Then, the SVD was computed on this smaller matrix A_p, in order to estimate the stability of the entire i-RevNet trained on ImageNet.

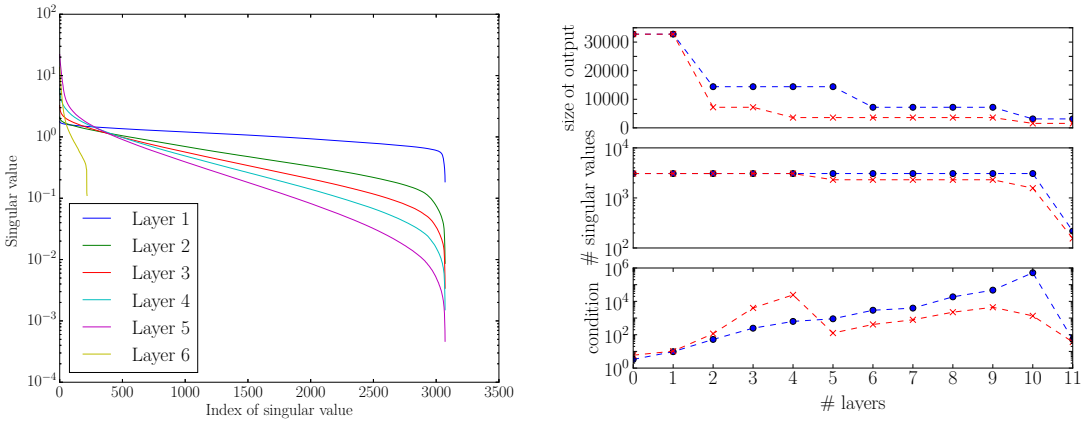


Figure 3.4: Left: Decay of singular values over the layers of the network. Here, each layer includes the convolution and ReLU-activation. Reported number are taken from median over 50 samples. **Right:** Comparison of WideCIFAR (blue) and ThinCIFAR (red). Top: number of output units per layer, Middle: number of singular values, Bottom: Behavior of condition number, each curve over the layers. Here, layers are split into conv-layer and ReLU-activation layer. Singular values and condition number are the median over 50 samples from the CIFAR10 test set.

3.5 Conclusion and Future Work

We presented the study of the inverse as an approach to better understand the invariance and robustness properties of ReLU networks. Particularly, we studied two main effects:

(i) Conditions under which the preimage of a ReLU layer is a point, finite or infinite

(ii) Effects of ReLU on the inverse stability of the linearization.

By deriving approaches to numerically examine these effects, we highlighted the broad range of possible effects. While above analysis allowed us to better understand the key mechanisms affecting invertibility of rectifier networks, several limitations remain, which could be tackled in future work:

Inverse stability for convolutional neural networks: For inverse stability, we consider the linearization $A_P = A^L D_{I^{L-1}} A^{L-1} \cdots D_{I^1} A^1$ for an input polytope P. In convolutional networks, each A^l implements a multi-channel discrete convolution. While singular values of each A^l can be efficiently computed by leveraging the convolutional structure, see (Sedghi et al., 2019), the shared structure is not preserved in the matrix chain A_P due to the application of ReLU (expressed via D_{I^l}). Thus, a tighter analysis that leverages the convolutional structure in A^l, compared to our general assumption of A^l being any linear mapping, is not straightforward with current tools. Yet, advances in this direction would certainly lead to further insights.

Extension of inverse stability across polytopes: In our stability analysis, we employ a piecewise-linear viewpoint. This allows to characterize stability via the singular values

of the linearization, which is exact within an input polytope. However, when considering an ε-ball $B_\varepsilon(y)$ around a point $y = A_P x + b_p$ to model e.g. reconstruction from noisy activations y, further questions arise:

(i) Can all points in $B_\varepsilon(y)$ be reached by an x^* from the polytope P?

(ii) Are points from another polytope P' mapping to points in $B_\varepsilon(y)$?

In this case, the inverse stability needs to be augmented by nonlinear considerations to model movements between piecewise-linear regions.

Practical implications: Despite above limitations, the presented analysis on the preimages of ReLU-layers and inverse stability showed the breadth of potential effects arising in standard architectures. In terms of network design, a major conclusion is that standard networks allow only limited control over the invertibility. In particular:

- Injective activation functions (tanh, leakyReLU, ELU etc.) remove the discussed preimages, but yield an instable inverse due to saturation.
- Regularizing correlations (Rodríguez et al., 2017) is connected to inverse stability (see Lemma 3.6), but lacks a way to provide guarantees.

Hence, additional structure is required to allow for guaranteed invertibility of neural networks. In the following chapter, we present a principled way to design powerful bijective and inverse stable networks by drawing inspiration from stability analysis.

Chapter 4

Invertible Residual Networks

The previous chapter gave an impression, why controlling the invertibility with standard architectures is a difficult endeavor. By using the concept of stability, we thus introduce specialized invertible neural networks based on residual connections and contractions. After discussing the underlying concepts, we study the proposed invertible residual network numerically and apply it to several image classification benchmarks.

This section is mainly based on the article:

> **Jens Behrmann**, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, Jörn-Henrik Jacobsen: *Invertible residual networks*, 2019, (International Conference on Machine Learning (ICML))

The generalization to other norms is based on following article (article above only considered ℓ_2-norm):

> Ricky T. Q. Chen, **Jens Behrmann**, David Duvenaud, Jörn-Henrik Jacobsen: *Residual Flows: invertible generative modeling*, 2019, (under submission; early version presented at: ICML workshop on Invertible Networks and Normalizing Flows)

4.1 Introduction and Motivation

Invertible networks have been shown to produce competitive performance on discriminative (Gomez et al., 2017; Jacobsen et al., 2018) and generative (Dinh et al., 2014, 2017; Kingma and Dhariwal, 2018) tasks independently, albeit in the same model paradigm. They typically rely on fixed dimension splitting heuristics, but common splittings interleaved with non-volume conserving elements are constraining and their choice has a significant impact on performance (Kingma and Dhariwal, 2018; Dinh et al., 2017). This makes building reversible networks a difficult task.

To overcome this problem, we leverage the viewpoint of Residual Networks (He et al., 2016) (ResNets) as an Euler discretization of ordinary differential equations (ODEs), see (Haber and Ruthotto, 2018; Ruthotto and Haber, 2018; Lu et al., 2017; Ciccone et al., 2018). In particular, we show that invertible ResNets (i-ResNets) can be constructed by

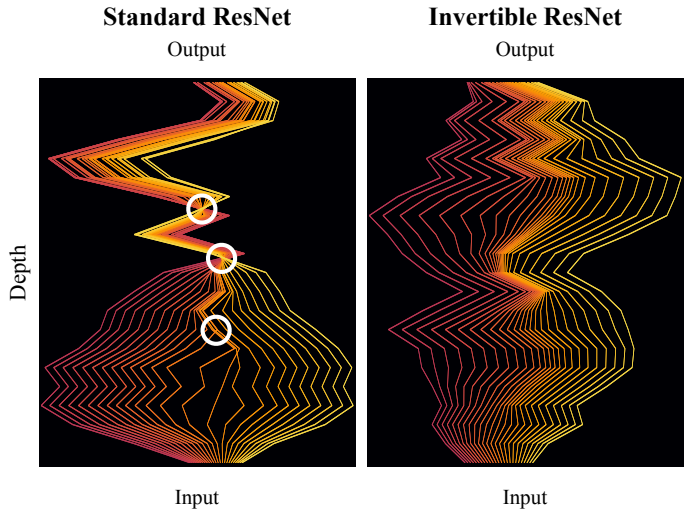


Figure 4.1: Dynamics of a standard residual network (left) and invertible residual network (right). Invertible ResNets describe a bijective continuous dynamic while regular ResNets result in crossing and collapsing paths (circled in white) which correspond to non-bijective continuous dynamics. Due to collapsing paths, standard ResNets are not a valid density model.

simply changing the normalization scheme of standard ResNets. As an intuition, Figure 4.1 visualizes the differences in the dynamics learned by standard and invertible ResNets.

This approach allows unconstrained architectures for each residual block, while only requiring a Lipschitz constant smaller than one for each block[1]. By leveraging this stability constraint, we further get stability bounds both for the forward and inverse mapping.

After introducing the i-ResNet, we show that standard ResNets are not guaranteed to be invertible without the proposed stability constraint. Finally, we demonstrate that this restriction negligibly impacts performance when building image classifiers - they perform on par with their non-invertible counterparts on classifying MNIST, CIFAR10 and CIFAR100 images.

4.2 Enforcing Invertibility in Residual Networks

Residual networks differ in the structure from standard feedforward networks in one fundamental aspect. In addition to the composition of affine and non-linear mappings, they employ identity paths. Mathematically, a residual layer can be written as

$$x_{t+1} = x_t + g_{\theta_t}(x_t),$$

where g_θ is a (usually shallow) standard network, called residual block, and t denotes the layer index. Strikingly, there is a similarity between ResNet architectures and Euler's

[1]Note that this still allows for arbitrarily high Lipschitz constants of the full network.

method for ODE initial value problems. Those two approaches can be formulated as

$$x_{t+1} = x_t + g_{\theta_t}(x_t)$$
$$x_{t+1} = x_t + h f_{\theta_t}(x_t),$$

where $x_t \in \mathbb{R}^d$ represent activations or states, t represents layer indices or time index, $h > 0$ is a time step size, and g_{θ_t} is a residual block. Note, that we use T (instead of L) to denote the number of layers in order to emphasize its relationship to time. This connection has attracted research at the intersection of deep learning and dynamical systems, see e.g. (Lu et al., 2017; Haber and Ruthotto, 2018; Ruthotto and Haber, 2018; Chen et al., 2018). However, little attention has been paid to the dynamics backwards in time

$$x_t = x_{t+1} - g_{\theta_t}(x_t)$$
$$x_t = x_{t+1} - h f_{\theta_t}(x_t),$$

which amounts to the implicit backward Euler discretization. Most notably, solving the dynamics backwards in time would implement an inverse of the corresponding ResNet. The following theorem states that a simple condition suffices to make the dynamics solvable and thus renders the ResNet invertible:

Theorem 4.1 (Sufficient condition for invertible ResNets)**.** *Consider $p \in [1, \infty]$ and let $F_\theta : (\mathbb{R}^d, \|\cdot\|_p) \to (\mathbb{R}^d, \|\cdot\|_p)$ with $F_\theta = (f_\theta^T \circ \ldots \circ f_\theta^1)$ denote a ResNet with blocks $f_\theta^t = I + g_{\theta_t}$. Then, the ResNet F_θ is invertible if*

$$\mathrm{Lip}(g_{\theta_t}) < 1, \text{ for all } t = 1, \ldots, T,$$

where $\mathrm{Lip}(g_{\theta_t})$ is a Lipschitz constant of g_{θ_t}.

Proof. Since the ResNet F_θ is a composition of functions, it is invertible if each block f_θ^t is invertible. Let $x_{t+1} \in \mathbb{R}^d$ be arbitrary and rearrange to get the fixed-point equation $x_t = x_{t+1} - g_{\theta_t}(x_t)$. Rewriting as an iteration yields

$$x_t^{[0]} := x_{t+1} \quad \text{and} \quad x_t^{[k+1]} := x_{t+1} - g_{\theta_t}(x_t^{[k]}), \tag{4.1}$$

where $\lim_{k\to\infty} x_t^{[k]} = x_t$ is the fixed-point if the iteration converges. As $g_{\theta_t} : (\mathbb{R}^d, \|\cdot\|_p) \to (\mathbb{R}^d, \|\cdot\|_p)$ is an operator on the Banach space $(\mathbb{R}^d, \|\cdot\|_p)$, the contraction condition $\mathrm{Lip}(g_{\theta_t}) < 1$ guarantees convergence due to the Banach fixed-point theorem. □

Remark 4.2. *The condition above was also stated in (Zhao et al., 2019) (Appendix D), however, their proof restricts the domain of the residual block g to be bounded and applies only to linear operators g, because the inverse was given by a convergent Neumann series.*

Remark 4.3. *This condition is not necessary for invertibility. Other approaches (Dinh et al., 2014, 2017; Jacobsen et al., 2018; Chang et al., 2018; Kingma and Dhariwal, 2018) rely on partitioning dimensions or autoregressive structures to create analytical inverses.*

In addition to ensuring invertibility of a ResNet, we can make further statements about the properties of the inverse:

Algorithm 1 Inverse of i-ResNet layer via fixed-point iteration.

Input: activation y, contractive residual block g, number of fixed-point iterations n
Init: $x^{[0]} := y$
for $k = 0, \dots, n$ **do**
  $x^{[k+1]} := y - g(x^{[k]})$
end for

Corollary 4.4 (Invertible ResNet as diffeomorphism)**.** *Let F_θ denote an invertible ResNet, where the condition from Theorem 4.1 holds. Further, let the activation functions ϕ used in the residual blocks be continuously differentiable, i.e. $\phi \in C^1(\mathbb{R})$. If this holds, then*

$$F_\theta \in C^1(\mathbb{R}^d) \quad \text{and} \quad F_\theta^{-1} \in C^1(\mathbb{R}^d),$$

which makes the invertible ResNet a diffeomorphism.

Proof. The first statement $F_\theta \in C^1(\mathbb{R}^d)$ directly follows from the compositional structure of feedforward neural networks (see Def. 2.15), since compositions of C^1-functions are in C^1. Furthermore, F_θ^{-1} is assumed to exist. Hence, $F_\theta^{-1} \in C^1(\mathbb{R}^d)$ follows directly by the inverse function theorem. □

While enforcing $\mathrm{Lip}(g) < 1$ makes the ResNet invertible, we have no analytic closed-form of its inverse. As shown in the proof of Theorem 4.1, however, we can approximate the inverse up to an arbitrary precision through a simple fixed-point iteration (see Algorithm 1). Note that the starting value $x^{[0]}$ for the fixed-point iteration is arbitrary, because the fixed-point is unique. However, using $x^{[0]} = y$ for initializing is a good starting guess, since y was obtained from x only via a bounded perturbation of the identity. Furthermore, we can make the following statement about the error after k iterations of the fixed-point iteration:

Lemma 4.5 (Error of fixed-point iteration)**.** *Let $\mathrm{Lip}(g) =: L < 1$ and $x^{[0]}$ be the initial iterate of the fixed-point iteration $x^{[k+1]} := x^{[0]} - g(x^{[k]})$. Furthermore, denote the fixed-point as x. Then the error after k iterations is bounded by*

$$\|x - x^{[k]}\|_p \leq \frac{L^k}{1-L} \|x^{[1]} - x^{[0]}\|_p.$$

Proof. First, via induction we prove that

$$\|x^{[k+1]} - x^{[k]}\|_p \leq L^k \|x^{[1]} - x^{[0]}\|_p \tag{4.2}$$

holds. The base case with $k = 0$ is trivial, since $L^0 = 1$. Consider the inductive step, where $k \to k+1$. For this introduce the mapping $g^*(x^{[k]}) := x^{[0]} - g(x^{[k]}) = x^{[k+1]}$. Since $x^{[0]}$ acts as a constant offset, we have $\mathrm{Lip}(g^*) = \mathrm{Lip}(g) = L < 1$. Using this in the inductive steps yields

$$\begin{aligned} \|x^{[k+2]} - x^{[k+1]}\|_p &= \|g^*(x^{[k+1]}) - g^*(x^{[k]})\|_p \\ &\leq L \|x^{[k+1]} - x^{[k]}\|_p \\ &\leq L^{k+1} \|x^{[1]} - x^{[0]}\|_p, \end{aligned}$$

where we used that the statement (4.2) holds for k.
Next, let $m > k > 0$. Then by adding zeros, using the triangular inequality and using (4.2), we have

$$\begin{aligned}\|x^{[m]} - x^{[k]}\|_p &\leq \sum_{l=k+1}^{m} \|x^{[l]} - x^{[l-1]}\|_p \\ &\leq \sum_{l=k+1}^{m} L^{l-1}\|x^{[1]} - x^{[0]}\|_p \\ &= L^k\|x^{[1]} - x^{[0]}\|_p \sum_{l=0}^{m-(k+1)} L^l.\end{aligned}$$

Now, let $m \to \infty$. Then, we have

$$\begin{aligned}\|x - x^{[k]}\|_p &= \lim_{m\to\infty} \|x^{[m]} - x^{[k]}\|_p \\ &\leq L^k\|x^{[1]} - x^{[0]}\|_p \sum_{l=0}^{\infty} L^l \\ &= \frac{L^k}{1-L}\|x^{[1]} - x^{[0]}\|_p,\end{aligned}$$

which holds by the properties of a geometric series. □

Thus, the error decay exponentially in the number of iterations k and smaller Lipschitz constants will yield faster convergence. Furthermore, it is important to note that the rate of the convergence does not depend on the input dimension d. Hence, this approach naturally scales to high-dimensional settings.

Additional to invertibility, a contractive residual block also renders the residual layer bi-Lipschitz:

Lemma 4.6 (Lipschitz constants of forward and inverse mapping)**.** *Let $F(x) = x + g(x)$ with $\mathrm{Lip}(g) =: L < 1$ denote the residual layer. Then, it holds*

$$\mathrm{Lip}(F) \leq 1 + L \quad \text{and} \quad \mathrm{Lip}(F^{-1}) \leq \frac{1}{1-L}.$$

Proof. First note, that $\mathrm{Lip}(F) \leq 1 + L$ follows directly from the addition of Lipschitz constants. For the inverse, consider

$$\begin{aligned}\|F(x) - F(y)\|_p &= \|x - y + g(x) - g(y)\|_p \\ &\geq \|x - y\|_p - \|g(x) - g(y)\|_p \qquad (4.3) \\ &\geq \|x - y\|_p - L\|x - y\|_p, \qquad (4.4)\end{aligned}$$

where we apply the reverse triangular inequality in (4.3) and use the Lipschitz constant of g in (4.4). For all $z, w \in \mathbb{R}^d$ denote $x = F^{-1}(z)$ and $y = F^{-1}(w)$. Inserting above yields

$$\begin{aligned}&\|F(F^{-1}(z)) - F(F^{-1}(w))\|_p \geq (1-L)\|F^{-1}(z) - F^{-1}(w)\|_p \\ \iff \quad &\frac{1}{1-L}\|z - w\|_p \geq \|F^{-1}(z) - F^{-1}(w)\|_p,\end{aligned}$$

which proves the claimed Lipschitz constant. □

To conclude, invertible ResNets offer stability guarantees for both their forward and inverse mapping by construction.

So far, we discussed when ResNets are guaranteed to be invertible, but without presenting an approach to satisfy the condition. In the following subsection, we thus discuss several ways to enforce the Lipschitz condition.

4.2.1 Satisfying the Lipschitz Condition

We implement residual blocks as a composition of 1-Lipschitz nonlinearities ϕ and affine mappings. For example, in our convolutional networks we use blocks with three layers, i.e.

$$g(x) = A_3\phi(A_2(\phi(A_1\phi(x) + b_1) + b_2) + b_3, \tag{4.5}$$

where A_i are convolutional layers and b_i denote biases.

Remark 4.7. *We slightly changed notation (A_i instead of A^i to denote the i-th linear layer) in order to avoid confusion, because we need matrix powers in chapter 5 in connection with invertible ResNets.*

In this discussion and subsequent experiments we focus on a Lipschitz condition for $p = 2$ (contractive residual blocks in $\|\cdot\|_2$), because the Euclidean norm is often a natural choice. For a more general discussion involving different p-norms and even mixed matrix norms, we refer to the follow-up work (Chen et al., 2019***).

In the case of the 3-layers residual block in (4.5), the Lipschitz constant of g can be upper bounded in a data-independent manner via

$$\mathrm{Lip}(g) \leq \prod_{i=1}^{3} \|A_i\|_2, \tag{4.6}$$

if the activation function is 1-Lipschitz, i.e. $\mathrm{Lip}(\phi) \leq 1$.

Remark 4.8 (On different activation functions)**.** *Most activation functions like ReLU, ELU or tanh are 1-Lipschitz and thus this condition poses only a minor restriction. However, in (Chen et al., 2019***) a less-common activation function was used, which required further scaling to satisfy this upper bound.*

Hence, by the data-independent upper bound it holds

$$\mathrm{Lip}(g) < 1, \quad \text{if } \|A_i\|_2 < 1 \text{ for } i = 1, 2, 3, \tag{4.7}$$

where $\|\cdot\|_2$ denotes the spectral norm of the linear operator. Note, that regularizing the spectral norm of the Jacobian of g as e.g. in (Sokolić et al., 2017) only reduces it locally and does not guarantee the above condition. Thus, we explicitly enforce $\|A_i\|_2 < 1$ for each layer.

To ensure that $\|A_i\|_2 < 1$ holds, we employ a variant of spectral normalization, which was introduced in (Miyato et al., 2018) for the regularization of Generative Adversarial

Networks (GANs) (Goodfellow et al., 2014). In particular, (Miyato et al., 2018) relied on a power-iteration on the parameter matrix $\tilde{W}$ of a convolutional operator A_i, which computes an approximation $\tilde{\sigma}_1 \leq \|\tilde{W}\|_2$ (for details on the difference between $\tilde{W}$ and A_i see section 2.2.2). However, if the filter kernels of the convolutional layer A_i are larger than 1×1, one needs to consider the bound

$$\|A_i\|_2 \leq \sqrt{n}\|\tilde{W}\|_2, \tag{4.8}$$

which connects the spectral norm of the parameter matrix with the spectral norm of the convolution operator, see (Tsuzuku et al., 2018, Corollary 1). In this bound, n acts a constant, that depends on the filter size, strides and boundary handling of the convolution. Hence, using the upper bound (4.8) to ensure $\|A_i\|_2 < 1$ is possible, but might be more restrictive than necessary. This is why we directly perform the power-iteration on A_i and A_i^T to approximate the spectral norm of the convolutional operator directly as proposed in (Gouk et al., 2018).

The power-iteration then yields an approximation $\tilde{\sigma}_i$ from below, i.e. $\tilde{\sigma}_i \leq \|A_i\|_2$. Using this approximation, we normalize via

$$\tilde{A}_i = \begin{cases} c\, A_i/\tilde{\sigma}_i, & \text{if } c/\tilde{\sigma}_i < 1 \\ A_i, & \text{else} \end{cases}, \tag{4.9}$$

where the hyper-parameter $c \in (0, 1)$ is a scaling coefficient. Since $\tilde{\sigma}_i$ is only an approximation from below, $\|A_i\|_2 \leq c$ is not guaranteed, which is why we need to run an additional check after training:

Remark 4.9 (Checking spectral norm of linear layers after training)**.** *After training (Sedghi et al., 2019) offer an approach to inspect $\|A_i\|_2$ exactly using the singular value decomposition (SVD) on the Fourier transformed (FFT) parameter matrix $\tilde{W}$. As a consequence, this allow to check if* $\mathrm{Lip}(g) < 1$ *holds in all cases. However, we note that the approach in (Sedghi et al., 2019) relies on circular convolutions and is not exact if other boundary extensions were used for the convolution. Hence, a second way to check if the Lipschitz condition holds after training is to run the power-iteration until convergence (power-iteration is guaranteed to converge for an approximation of $\|\cdot\|_2$).*

*In (Behrmann et al., 2019***), the SVD-FFT based variant from (Sedghi et al., 2019) was used, while the follow-up study from (Chen et al., 2019***) used an adaptive version of the spectral normalization to ensure that the power-iteration converged up to a given tolerance.*

Motivation for power-iteration on linear layers: The approach described above relies on two key properties: First, data-independent upper bounds of Lipschitz constants (4.6). Second, efficient approximation of spectral norm via power-iterations. While these data-independent bounds are often loose in practice, finding an alternative is difficult, because the computation of Lipschitz constants of even two-layer neural networks is NP-hard (Virmaux and Scaman, 2018).
There are several alternative to ensure $\|A_i\|_2 < 1$ during training, yet each approach has either computational drawbacks or does not apply to convolutional layers:

- Parametrization of orthogonal matrices, e.g. via Householder reflections as proposed for recurrent neural networks (RNNs) in (Mhammedi et al., 2017): not applicable to convolutional layers as the spectrum of the convolutional operator A_i (see Definition 2.23) differs from the spectrum of the parameter matrix $\tilde{W}$ (obtained by writing the 4D kernel tensor as a matrix). For details, we refer to the discussion in (Sedghi et al., 2019) and the bound (4.8) from (Tsuzuku et al., 2018).

- Björck orthogonalization as in (Anil et al., 2019): so far only applied to fully-connected layers. The application to convolutional layer might be computationally expensive or less straightforward due to their more complex structure.

- Projection via efficient computation of SVD of convolutional layers as in (Sedghi et al., 2019): this approach presents an alternative to spectral normalization, but did not scale as good as spectral normalization in our experiments. The main bottleneck of this approach are convolutional layers with many input and output channels, which unfortunately was a common setting in our experiments.

To summarize, spectral normalization presents a salable approach to enforce the Lipschitz-constraint with respect to the ℓ_2-norm. However, the approximation via power-iteration only yields a lower bound. To still ensure invertibility, either an appropriate number of iterations or an adaptive version with a given tolerance needs to be used. Furthermore, power iteration adds computational cost, which increases if a higher number of power iterations are used.

After introducing the condition to turn a ResNet into an invertible ResNet and proposing an efficient methodology to satisfy this condition, we now turn to a numerical analysis.

4.3 Numerical Analysis

To compare the invertibility and discriminative performance of i-ResNets with standard ResNet architectures, we train both models on CIFAR10, CIFAR100, and MNIST. First, we numerically validate the invertibility of i-ResNets by the fixed-point iteration and observe that the invertibility of vanilla ResNets is task-dependent. Second, we compare the performance on the mentioned image classification benchmarks.

4.3.1 Validating Invertibility

The CIFAR and MNIST models have 54 and 21 residual blocks, respectively and we use identical settings for all other hyperparameters. We replace strided downsampling with *invertible downsampling* (Jacobsen et al., 2018) to ensure bijectivity, see Appendix A.1 for training and architectural details. Furthermore, we increase the number of input channels to 16 by padding with zeros. This is analogous to the common practice of projecting the data into a higher-dimensional space using a standard convolutional layer at the input of a model, but this mapping is injective.

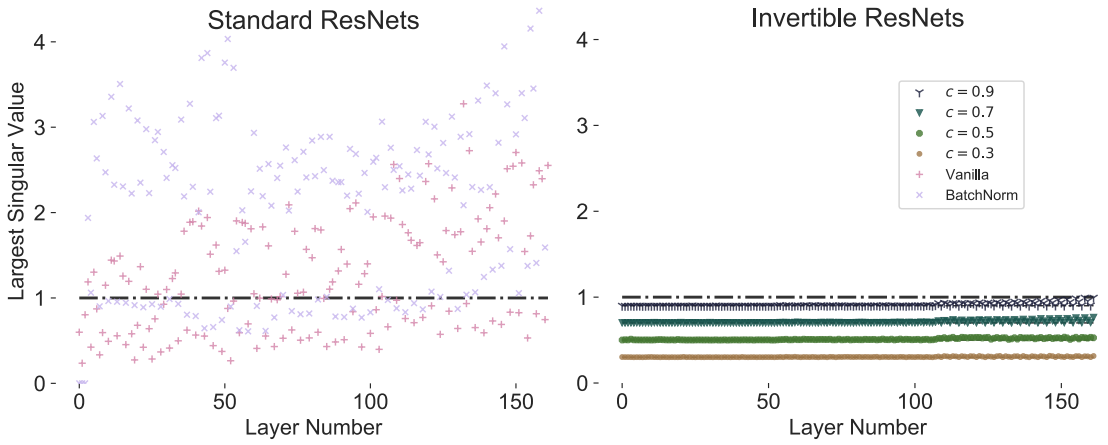


Figure 4.2: Maximal singular value of each convolutional layer for various trained ResNets on Cifar10. **Left:** Vanilla and Batchnorm ResNet singular values. It is likely that the baseline ResNets are not invertible as roughly two thirds of their layers have singular values fairly above one, making the blocks non-contractive. **Right:** Singular values for our 4 spectrally normalized ResNets. The regularization is effective and in every case the single ResNet block remains a contraction.

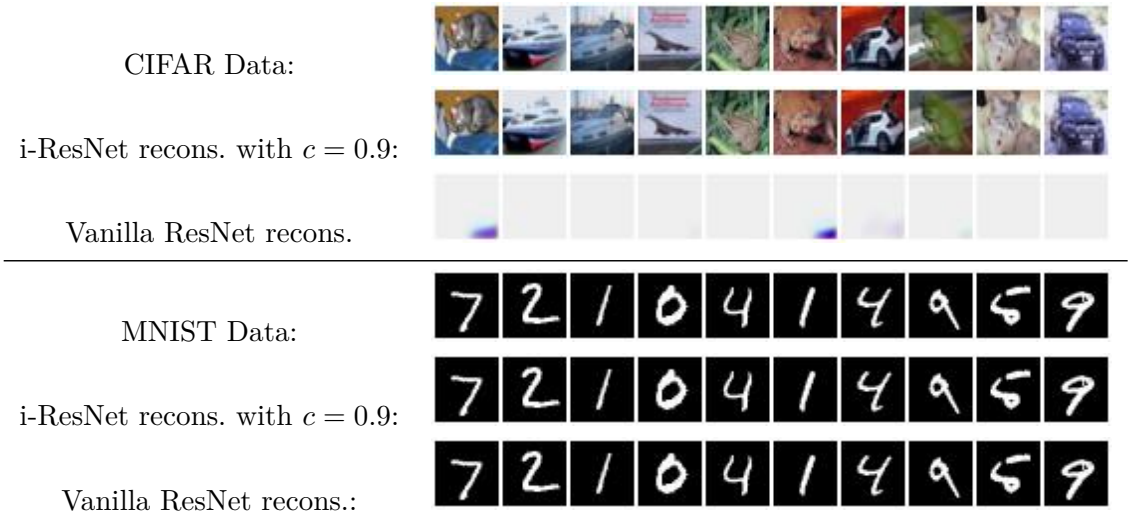


Figure 4.3: Original images (top), i-ResNets with $c = 0.9$ (middle) and reconstructions from vanilla (bottom). Surprisingly, MNIST reconstructions are close to exact for both models, even without explicitly enforcing the Lipschitz constant. On CIFAR10 however, reconstructions completely fail for the vanilla ResNet, but are qualitatively and quantitatively exact for our proposed network.

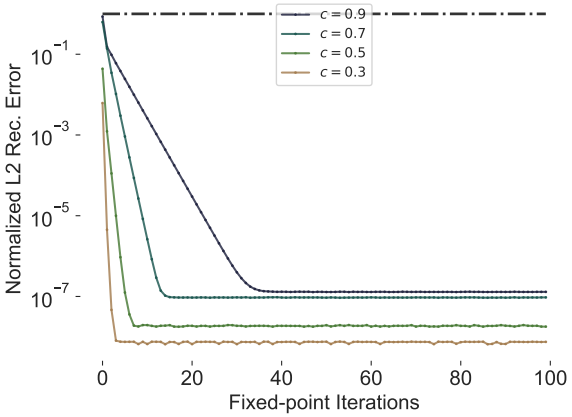


Figure 4.4: Trade-off between number of fixed-point iterations and reconstruction error (log scale) for computing the inverse for different normalization coefficients of trained invertible ResNets (on CIFAR10). The reconstruction error decays quickly. 5-20 iterations are sufficient respectively to obtain visually perfect reconstructions. Note that one iteration corresponds to the time of one forward pass, thus inversion is approximately 5-20 times slower than inference. This corresponds to a reconstruction time of 0.15-0.75 seconds for a batch of 100 CIFAR10 images with 5-20 iterations and 4.3 seconds with 100 iterations (see section 4.3.2 for more details).

Singular values of learned mappings: We train standard ResNets and i-ResNets with various layer-wise scaling coefficients ($c \in \{.3, .5, .7, .9\}$). After training, we inspect the learned transformations at each layer by computing the largest singular value of each convolutional layer based on the approach in (Sedghi et al., 2019). It is clearly visible (Figure 4.2 (left)) that the vanilla and BatchNorm models have many singular values above one, making their residual connections (potentially) non-invertible. Conversely, in the i-ResNet models (Figure 4.2 (right)), all singular values are below 1 (and roughly equal to c), which indicates that their residual connections are invertible. Hence, the proposed spectral normalization scheme works and allows to enforce invertibility.

Computing image reconstructions with fixed-point iteration: To obtain the numerical inverse, we apply 100 fixed-point iterations for each block (Algorithm 1). We observe that i-ResNets can be inverted using this method, whereas with standard ResNets this is not guaranteed (Figure 4.3 (top)). Interestingly, on MNIST we find that standard ResNets are indeed invertible after training on MNIST (Figure 4.3 (bottom)). Note, that a high number of fixed-point iteration is chosen to ensure that the poor reconstructions for vanilla ResNets are not due to using too few iterations.

Convergence of fixed-point iteration: In practice, far fewer iterations than 100 suffice, as the trade-off between reconstruction error and number of iterations visualized in Figure 4.4 shows. Thus, the Lipschitz coefficient c further controls the convergence speed as discussed in Lemma 4.5. For example, the number of iterations needed for convergence is approximately halved when reducing the spectral normalization coefficient by 0.2.

	ResNet-164	Vanilla	$c = 0.9$	$c = 0.8$	$c = 0.7$	$c = 0.6$	$c = 0.5$
MNIST	-	0.38	0.40	0.42	0.40	0.42	0.86
CIFAR10	5.50	6.69	6.78	6.86	6.93	7.72	8.71
CIFAR100	24.30	23.97	24.58	24.99	25.99	27.30	29.45
Guaranteed Inverse	**No**	**No**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

Table 4.1: Comparison of classification error (in %) of i-ResNets to a ResNet-164 baseline architecture of similar depth and width with varying Lipschitz constraints via coefficients c. Vanilla shares the same architecture as the i-ResNets, but without the Lipschitz constraint.

4.3.2 Image Classification

Classification and reconstruction results for a baseline pre-activation ResNet-164, a ResNet with architecture like i-ResNets without Lipschitz constraint (denoted as vanilla) and five invertible ResNets with different spectral normalization coefficients are shown in Table 4.1. The results illustrate that for larger constants, our proposed invertible ResNets perform competitively with the baselines in terms of classification performance, while being provably invertible. When applying very conservative normalization (small c), the classification error becomes higher on all datasets tested.

Runtime of forward and backward pass: The runtime on 4 GeForce GTX 1080Ti GPUs with one spectral norm iteration was 0.5 sec for a forward and backward pass using a batch with 128 samples, while it took 0.2 sec without spectral normalization.

Costs of reconstruction: The reconstruction error decays quickly and the errors are already imperceptible after 5-20 fixed-point iterations. To better understand the computational cost of one fixed-point iteration, it is helpful to compare a residual layer to a step from Algorithm 1. Since the only difference is the input and the plus/minus sign, their computational costs are identical. Thus, above reconstruction has the same cost as 5-20 forward passes. On our hardware the reconstruction of 100 CIFAR10 images took between 0.15-0.75 seconds.

In addition to the presented classification results in Table 4.1, we refer to (Behrmann et al., 2019***) for a comparison of i-ResNets with the current state-of-the-art flow-based density model Glow (Kingma and Dhariwal, 2018).

In summary, we observe that guaranteed invertibility without additional constraints is unlikely, but possible, whereas it is hard to predict if networks will have this property. In our proposed model, we can guarantee the existence of an inverse without significantly harming classification performance.

4.4 Related Work

Due to the similarity of ResNets and Euler discretizations, there are many connections between i-ResNets and ODEs, which we review in this section. Note, that a review of other

approaches to design invertible networks is postponed to the next chapter on generative models in order to jointly discuss the networks architectures and their applicability to flow-based density modeling.

Relationship of i-ResNets to Neural ODEs: The view of deep networks as dynamics over time offers two fundamental learning approaches:

(i) Direct learning of dynamics using discrete architectures like ResNets, see (Haber and Ruthotto, 2018; Ruthotto and Haber, 2018; Lu et al., 2017; Ciccone et al., 2018)

(ii) Indirect learning of dynamics via parametrizing an ODE with a neural network, see (Chen et al., 2018; Grathwohl et al., 2019).

The dynamics $x(t)$ of a fixed ResNet F_θ are only defined at time points t_i corresponding to each block $g_{\theta_{t_i}}$. However, a linear interpolation in time can be used to generate continuous dynamics. See for example Figure 4.1, where the continuous dynamics of a linearly interpolated invertible ResNet are shown against those of a standard ResNet. Invertible ResNets are bijective along the continuous path, while regular ResNets may result in crossing or merging paths. The indirect approach of learning an ODE, on the other hand, adapts the discretization based on an ODE-solver, but does not have a fixed computational budget compared to an i-ResNet.

Stability of ODEs: There are two main approaches to study the stability of ODEs:

(i) Studying the behavior for $t \to \infty$.

(ii) Considering Lipschitz stability over finite time intervals $[0, T]$.

Based on time-invariant dynamics $f(x(t))$, (Ciccone et al., 2018) constructed asymptotically stable ResNets using anti-symmetric layers such that $Re(\lambda(J_x g)) < 0$. Here, $Re(\lambda(\cdot))$ denotes the real-part of eigenvalues and $J_x g$ the Jacobian at point x. By projecting weights based on the Gershgorin circle theorem, they further fulfilled $\rho(J_x g) < 1$, where $\rho(\cdot)$ is the spectral radius. This approach yields an asymptotically stable ResNet with shared weights over layers.

On the other hand, (Haber and Ruthotto, 2018; Ruthotto and Haber, 2018) considered time-dependent dynamics $f(x(t), \theta(t))$ corresponding to standard ResNets. They induce stability by using anti-symmetric layers and projections of the weights. Contrarily, initial value problems on $[0, T]$ are well-posed for Lipschitz continuous dynamics (Ascher, 2008). Thus, the invertible ResNet with $\text{Lip}(f) < 1$ can be understood as a stabilizer of an ODE for step size $h = 1$, yet without a restriction to anti-symmetric layers as in (Ruthotto and Haber, 2018; Haber and Ruthotto, 2018; Ciccone et al., 2018).

4.5 Conclusion and Future Work

Invertible residual networks (i-ResNets) share a clear analogy to the forward and backward Euler discretization of ODEs. Building invertible neural networks based on stability allows

unconstrained architectures without the need of arbitrarily splitting input dimensions like previous approaches (Dinh et al., 2014, 2017; Chang et al., 2018). Compared to vanilla ResNets, only a small adaption via spectral normalization was necessary to guarantee invertibility based on the ℓ_2-norm. For results using i-ResNets based on other norms, we refer to the follow-up work (Chen et al., 2019***).

Lipschitz conditions: An interesting avenue for future work is the investigation of other approaches to satisfy the Lipschitz condition. While spectral normalization was mainly chosen for computational reasons, other choices might be favorable in lower-dimensional settings where scalability is less of a concern. In general, however, learning and designing networks with a Lipschitz constraint is challenging. For example, we need to constrain each linear layer in the block instead of being able to directly control the Lipschitz constant of a block. See (Anil et al., 2019) for a promising approach for addressing this problem.

Generalization to other discretization schemes: Besides exploring the Lipschitz condition, a generalization of i-ResNets beyond their corresponding forward and backward Euler discretization schemes could further increase the flexibility of invertible networks. For example, trading off the one-pass forward of i-ResNets with a faster inversion method than fixed-point iterations, might allow to use these invertible networks in applications where faster inverses are necessary.
However, it is unclear if invertible networks based e.g. on multi-step discretization schemes are strictly better. Whereas multi-step discretization is usually favored over simple schemes like Euler integration when solving ODEs, the situation might be different when designing architectures. When solving ODEs, the ODE itself is fixed since it describes a given (physical) behavior, thus each integration scheme aims to solve the same ODE. For architecture design, however, there is no fixed underlying dynamics to be solved. Only the training data provides a rough outline of potentially useful dynamics. Hence, besides generalizing i-ResNets to other connection schemes, an interesting aspect for future work would be to gain a better understanding of changes in the dynamics when only training data is given, instead of a fixed ODE.

Universal approximation: A theoretical understanding of the expressivity of invertible neural networks is still in its infancy. As each invertible architecture requires certain constraints or conditions, it would be beneficial for further architecture design to have a mathematical understanding how each design choice influences the functions that the architecture is able to approximate.

Chapter 5

Generative Modeling with Invertible Residual Networks

"What I cannot create, I do not understand." - Richard Feynman.

In this chapter, we focus on maximum likelihood based density modeling using the proposed invertible residual networks (i-ResNets). After an introduction to deep generative modeling, we start by discussing a change of variables of probability densities under a diffeomorphism. Second, we leverage i-ResNets as a bijective function approximator to describe a likelihood model based on the mentioned change of variables. This approach, however, is difficult to scale to high-dimensional settings such as density models for images. Thus, we discuss approximations based on a truncation of a matrix power series, randomization and automatic differentiation. The chapter finishes with a numerical comparison of i-ResNets to other flow-based density models.

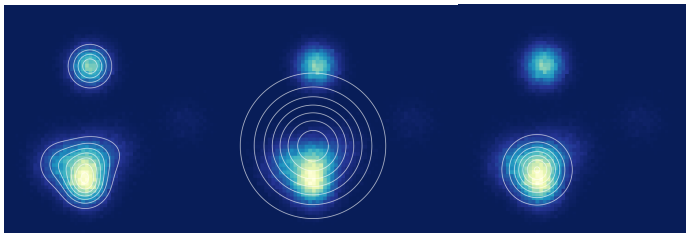
This chapter is mainly based on the following article:

> **Jens Behrmann**, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, Jörn-Henrik Jacobsen: *Invertible residual networks*, 2019, (International Conference on Machine Learning (ICML))

5.1 Generative Modeling - Overview

Besides regression, classification and dimensionality reduction, density estimation is another major pillar of machine learning. Estimating the density $p(x)$ from a set of training samples $\mathcal{T} = \{x^{(i)}\}_{i=1}^{N}$ is at the core of probabilistic unsupervised learning and generative modeling. Being able to both evaluate the model density $p_\theta(x)$ and generate samples from it, opens a multitude of use-cases like:

- Learning priors from data for Bayesian inference, see e.g. (Zoran and Weiss, 2011)
- Semi-supervised learning, see e.g. (Atanov et al., 2019)
- Detecting out-of-distribution examples, see e.g. (Nalisnick et al., 2019)



Data forward KL reverse KL

Figure 5.1: From (Theis et al., 2016): Different objective functions lead to different density models. **Left:** data distribution. **Middle:** uni-modal Gaussian model fitted via the forward Kullback-Leibler (KL) divergence to a multi-modal distribution. **Right:** uni-modal Gaussian model fitted via the reverse KL-divergence.

- Detecting adversarial examples, see e.g. (Fetaya et al., 2019).

While there is a vast amount of different approaches to generative modeling like mixture models, energy-based models, graphical models or implicit models requiring Markov Chain Monte Carlo approximations, this overview discusses the currently most popular deep learning approaches to generative modeling. For an extensive overview on (probabilistic) generative models, we refer to (Murphy, 2013).

Consider a parametrized density model $p_\theta(x)$ with parameters $\theta \in \mathbb{R}^p$. A natural way to evaluate density models p_θ is by computing the *forward Kullback-Leibler (KL) divergence*

$$D_{KL}(p(x) \parallel p_\theta(x)) = \int_{\mathbb{R}^d} p(x) \log \frac{p(x)}{p_\theta(x)} dx, \tag{5.1}$$

for inputs $x \in \mathbb{R}^d$ (see Definition 2.9). Minimizing the KL-divergence is equivalent to maximizing likelihood (see Lemma 2.14) and thus follows a standard statistical approach to density estimation. However, as noted e.g. in (Theis et al., 2016), density models p_θ need to cover the whole data space, as $p_\theta(x^{(i)}) \approx 0$ for a sample $x^{(i)}$ from the data distribution can result in arbitrarily large objective, since $\log p_\theta(x^{(i)})$ goes to infinity.

As models trained to minimize the forward KL-divergence need to cover the entire data space, models with limited capacity will fail to grasp the essence of the data. See for example Figure 5.1 (middle) from (Theis et al., 2016), where a uni-modal Gaussian is fitted to a mixture of Gaussians. The model needs to cover both modes and thus assigns high likelihood to data which lies in low-density regions. To avoid such misspecifications, it is essential to use models with the capacity to model complex multi-modal distributions when training via maximum likelihood. As i-ResNets have been shown to be a powerful family of (invertible) neural networks in the previous chapter, we study how to train i-ResNets as generative models via maximum likelihood in this chapter.

Since the KL-divergence is not symmetric (Cover and Thomas, 2006, Sec. 2.3), another

measure of interest is the *reverse KL-divergence*

$$D_{KL}(p_\theta(x) \parallel p(x)) = \int_{\mathbb{R}^d} p_\theta(x) \log \frac{p_\theta(x)}{p(x)} dx \tag{5.2}$$

$$= \mathbb{E}_{p_\theta(x)}[\log p_\theta(x)] - \mathbb{E}_{p_\theta(x)}[\log p(x)], \tag{5.3}$$

where the first term is the negative entropy of $x \sim p_\theta(x)$ and the second term is the log probability of samples from the density model $p_\theta(x)$ under the true data distribution $p(x)$. Hence, minimizing the reverse KL-divergence leads towards a model $p_\theta(x)$ with high entropy and samples $x \sim p_\theta(x)$ with high likelihood under the true distribution.

Remark 5.1. *We identified the distribution with the density in the considerations above. By assuming that there is a density for random variable X (which is given by our density model p_θ by construction), this identification is valid, because there is a unique correspondence between the distribution and density via the theorem of Radon-Nikodym.*

Given a model with low capacity (like a uni-modal Gaussian), minimizing the reverse KL-divergence on a mixture of Gaussians may lead to a density model which covers a single mode well, see Figure 5.1 (right) from (Theis et al., 2016). While this approach is favorable if high-quality samples are the main goal, it might not capture the whole data distribution. Yet more importantly, minimizing the second term requires access to the unknown true density $p(x)$. However, different variants of Generative Adversarial Networks (GANs) were shown to correspond to certain *f-divergences* (Nowozin et al., 2016). Thus, by using GANs one can circumvent the need for the true density $p(x)$ for the reverse KL-divergence. Yet, the correspondence between GANs and f-divergences relies on the tightness of lower bounds (Nowozin et al., 2016), which is hard to evaluate or even achieve in practical scenarios.

Similarly, a Variational Autoencoder (VAE) (Kingma and Welling, 2014) only provides a variational lower bound using an inference model (encoder) $q_\phi(z|x)$ and a decoder $p_\theta(x, z) = p_\theta(x|z)p_\theta(z)$. This bound on the likelihood (evidence) $p_\theta(x)$ can be derived using Lemma 2.10 (Kullback-Leibler divergence non-negative). Let $x \in \mathbb{R}^d$ be the input data and $z \in \mathbb{R}^{d'}$ a latent variable. Then the bound (evidence lower bound, ELBO)

$$\begin{aligned}
\log p_\theta(x) &= \int_{\mathbb{R}^{d'}} q_\phi(z|x) dz \log p_\theta(x) \\
&= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x)] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{p_\theta(z|x)}\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(x,z) q_\phi(z|x)}{q_\phi(z|x) p_\theta(z|x)}\right] \\
&= \underbrace{\mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z|x)}\right]}_{=:\mathcal{L}_{\phi,\theta}(x)} + \underbrace{\mathbb{E}_{q_\phi(z|x)}\left[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}\right]}_{=D_{KL}(q_\theta(z|x) \parallel p_\theta(z|x))} \\
&\geq \mathcal{L}_{\phi,\theta}(x)
\end{aligned}$$

holds and is tight under a perfect inference model, i.e. if and only if $q_\theta(z|x) = p_\theta(z|x)$, see (Kingma and Welling, 2014; Kingma, 2017).

While both GANs and VAEs are promising directions for generative modeling, we focus on models which can be trained exactly via maximum likelihood, as these models do not require a mechanism to tighten bounds. Furthermore, to avoid situations visualized in Figure 5.1 (middle), we aim towards high-capacity models using deep neural networks.

A popular approach for maximum likelihood modeling with deep networks relies on the chain rule of probability

$$p_\theta(x_{1,\dots,d}) = \prod_{i=1}^{d} p_{\theta_i}(x_i|x_{1\dots,i-1})$$

where $x = (x_1, \dots, x_d) \in \mathbb{R}^d$. For example, Masked Autoregressive Flows (MAFs) learn a neural network to model parameters of a one-dimensional Gaussian distribution conditioned on previous inputs (Papamakarios et al., 2017). While allowing for exact likelihood models, this approach has two main drawbacks. First, an ordering of the input dimensions needs to be chosen, which might be non-obvious for some input data like images. Second, while evaluation and training via maximum likelihood is fast, sampling from the learned density can be slow since it scales with the input dimensions (Papamakarios et al., 2017).

A second approach to maximum likelihood modeling is given by *normalizing flows* (Deco and Brauer, 1995; Rezende and Mohamed, 2015), which rely on invertible mappings. Thus, this approach both fits our goal of using exact maximum likelihood models and leverages the main architecture principle of this thesis - invertible neural networks. In this chapter, we first introduce the modeling approach underlying normalizing flows and discuss how to use i-ResNets in this framework.

5.2 Maximum-likelihood Modeling with i-ResNets

We can define a simple generative model for data $x \in \mathbb{R}^d$ by first sampling $z \sim p_Z(z)$, where $z \in \mathbb{R}^d$ and $p_Z(z)$ is a corresponding density given by a simple base distribution like a standard normal distribution. Then, by defining $x := \Phi(z)$ for some function $\Phi : \mathbb{R}^d \to \mathbb{R}^d$, we can map samples z to the data space. If Φ is a diffeomorphism, there is an inverse $F := \Phi^{-1}$, which allows to compute the likelihood of any input x under this model using the change of variables formula:

Lemma 5.2 (Change of variables for probability density functions)**.** *Let $F : \mathbb{R}^d \to \mathbb{R}^d$ be a diffeomorphism. Furthermore, let p_X and p_Z be the probability density function for random variable X and Z. Then by a change of variables using F, it holds*

$$\log p_X(x) = \log p_Z(F(x)) + \log |\det J_F(x)|, \tag{5.4}$$

where $J_F(x)$ is the Jacobian of F evaluated at x.

Proof. We adapt the derivations from (Billingsley, 1995). Let $A \subseteq \mathbb{R}^d$ be an open set and $\Phi \in C^1(\mathbb{R}^d)$ be a diffeomorphism. Then, the probability for the d-dimensional real random variable X with pdf $p_X(x)$ under transform Φ^{-1} can be re-written via

$$\mathbb{P}[\Phi^{-1}(X) \in A] = \mathbb{P}[\Phi(\Phi^{-1}(X)) \in \Phi(A)] = \mathbb{P}[X \in \Phi(A)].$$

Since X is a continuous random variable with pdf $p_X(x)$, it is

$$\mathbb{P}[X \in \Phi(A)] = \int_{\Phi(A)} p_X(x)\, dx.$$

By the change of variables (Theorem 17.2 in (Billingsley, 1995)), above integral can be reformulated by the transform Φ as

$$\int_{\Phi(A)} p_X(x)\, dx = \int_A p_{\Phi^{-1}(X)}(\Phi^{-1}(x))|\det J_{\Phi^{-1}}(x)|\, dx.$$

Now denote $F := \Phi^{-1}$, $Z := \Phi^{-1}(X)$ and $z := \Phi^{-1}(x)$. Then, random variable X has the density (with respect to Lebesgue measure)

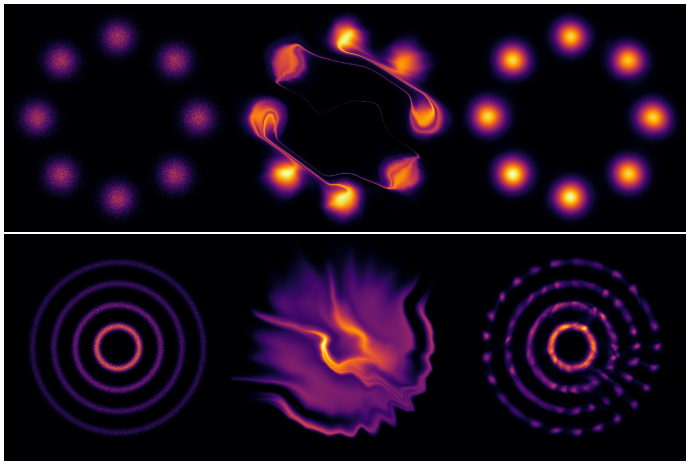
$$p_X(x) = p_Z(z)|\det J_F(x)|$$

and (5.4) follows by applying the logarithm. □

Models of this form are known as *normalizing flows* (Deco and Brauer, 1995; Rezende and Mohamed, 2015), because the transform F acts as a flow that normalizes the inputs. They have recently become a popular modeling approach for high-dimensional data due to the introduction of powerful bijective function approximators whose Jacobian log-determinant can be efficiently computed (Dinh et al., 2014, 2017; Kingma and Dhariwal, 2018; Chen et al., 2018) or approximated (Grathwohl et al., 2019).

Since i-ResNets are diffeomorphisms (if C^1-activation functions are used, see Corollary 4.4), we can use them to parameterize F in Equation (5.4). Samples from this model can be drawn by first sampling $z \sim p_Z(z)$ and then computing $x = F^{-1}(z)$ with Algorithm 1. In Figure 5.2, we show an example of using an i-ResNet to define a generative model on some two-dimensional datasets and compare the performance to Glow (Kingma and Dhariwal, 2018).

Remark 5.3 (Sampling speed when using i-ResNets)**.** *A major advantage of normalizing flows over autoregressive deep generative models is their fast inference and sampling speed, as both forward and inverse can be computed in a single pass of the network (Dinh et al., 2017; Kingma and Dhariwal, 2018). While being slower to invert compared to Real-NVP (Dinh et al., 2017) or Glow (Kingma and Dhariwal, 2018), i-ResNets present a different tradeoff to autogressive models like MAF (Papamakarios et al., 2017). Whereas autoregressive approaches need a fixed number of iterations equal to the dimensionality of inputs, i-ResNets rely on an approximation of the inverse using the fixed-point iteration. However, the convergence rate is independent of the dimension (see Lemma 4.5). Thus, a good approximation may be faster to achieve than the sampling in autoregressive models, especially in high-dimensional settings such as image generation.*



Data Samples Glow i-ResNet

Figure 5.2: Visual comparison of i-ResNet normalizing flow and Glow (Kingma and Dhariwal, 2018). In this setting, $p_Z(z)$ is a factorized Gaussian prior. Samples $z \sim p_Z(z)$ get mapped by F^{-1} to the data space, which allows to model complex probability distributions. We refer to (Behrmann et al., 2019***) for details of this experiment.

5.2.1 Scaling to Higher Dimensions

While the invertibility of i-ResNets allows to define a normalizing flow, we must compute $\log|\det J_F(x)|$ to evaluate the likelihood of the data under the model. Computing this quantity has a time cost of $\mathcal{O}(d^3)$ in general, which makes naïvely scaling to high-dimensional data impossible.

To bypass this constraint we present a tractable approximation to the log-determinant term in Equation (5.4), which will scale to high dimensions d. First, we note that the Lipschitz constrained perturbations $x + g(x)$ of the identity yield positive determinants. Note that we restrict the discussion to a Lipschitz constraint based on the ℓ_2-norm. In general, any $p \in [1, \infty]$ suffices, because any induced matrix norm upper bounds the spectral radius (Horn and Johnson, 2012).

Lemma 5.4 (Positive determinant of Jacobian of invertible residual layers)**.** *Let $F(x) = (I + g(\cdot))(x)$ denote a residual layer and $J_F(x) = I + J_g(x)$ its Jacobian at $x \in \mathbb{R}^d$. If $\mathrm{Lip}(g) < 1$ holds, then*

$$\det J_F(x) > 0.$$

Proof. Let λ_i denote the eigenvalues. First, we have

$$\lambda_i(J_F(x)) = \lambda_i(J_g(x)) + 1 \tag{5.5}$$

and $\|J_g(x)\|_2 < 1$ for all x due to the condition $\mathrm{Lip}(g) < 1$. Since the spectral radius is upper bounded as $\rho(J_g(x)) \leq \|J_g(x)\|_2$, we also have $|\lambda_i(J_g(x))| < 1$ for all i. Hence, the

real part $Re(\lambda_i(J_F(x)))$ is positive by inserting into (5.5). Thus

$$\det J_F(x) = \prod_i \lambda_i(J_F(x)) > 0,$$

because complex eigenvalues come in conjugated pairs, which renders the product above real-valued and positive. □

Combining the result from Lemma 5.4 with the matrix identity $\log\det(A) = \mathrm{tr}(\log(A))$ for non-singular $A \in \mathbb{R}^{d\times d}$ (see e.g. (Withers and Nadarajah, 2010)), we have

$$\log|\det J_F(x)| = \mathrm{tr}(\log J_F),$$

where tr denotes the matrix trace and log the matrix logarithm. Thus, for $z = F(x) = (I + g)(x)$, it is

$$\log p_x(x) = \log p_z(z) + \mathrm{tr}\left(\log\left(I + J_g(x)\right)\right).$$

The trace of the matrix logarithm can be expressed as a power series (Hall, 2015)

$$\mathrm{tr}\left(\log\left(I + J_g(x)\right)\right) = \sum_{k=1}^{\infty}(-1)^{k+1}\frac{\mathrm{tr}\left(J_g^k\right)}{k}, \tag{5.6}$$

which converges if $\rho(J_g) < 1$. Hence, due to the Lipschitz constraint, we can compute the log-determinant via the above power series with guaranteed convergence.

Before we present a stochastic approximation to the above power series, we observe following properties of i-ResNets:

Lemma 5.5 (Lower and upper bounds on log-determinant)**.** *Let $F_\theta : \mathbb{R}^d \to \mathbb{R}^d$ with $F_\theta = (f_\theta^T \circ \ldots \circ f_\theta^1)$ denote an invertible ResNet with blocks $f_\theta^t = I + g_{\theta_t}$. Then, we can obtain the following bounds*

$$d\sum_{t=1}^{T}\log(1 - \mathrm{Lip}(g_t)) \leq \log|\det J_F(x)|$$
$$d\sum_{t=1}^{T}\log(1 + \mathrm{Lip}(g_t)) \geq \log|\det J_F(x)|,$$

for all $x \in \mathbb{R}^d$.

Proof. First, the sum over the layers is due to the function composition, because $J_F(x) = \prod_t J_{f^t}(x)$ and

$$\log|\det J_F(x)| = \log\left(\prod_{t=1}^{T}\det J_{f^t}(x)\right) = \sum_{t=1}^{T}\log\det J_{f^t}(x),$$

where we use the positivity of the determinant (see Lemma 5.4). Furthermore, note that

$$(\sigma_d(A))^d \leq \prod_i \sigma_i(A) = |\det A| \leq (\sigma_1(A))^d$$

for a matrix A with largest singular values σ_1 and smallest σ_d. Further, we have $\sigma_i(J_{f^t}) \leq (1 + \text{Lip}(g_t))$ and $\sigma_d(J_{f^t}) \leq (1 - \text{Lip}(g_t))$, which follows from Theorem 4.6. Inserting this and applying the logarithm rules finally yields the claimed bounds. □

Thus, both the number of layers T and the Lipschitz constant affect the contraction and expansion bounds of i-ResNets and must be taken into account when designing such an architecture.

5.2.2 Stochastic Approximation of log-determinant

Expressing the log-determinant with the power series in (5.6) has three main computational drawbacks:

1) Computing $\text{tr}(J_g)$ exactly costs $\mathcal{O}(d^2)$, or approximately needs d evaluations of g as each entry of the diagonal of the Jacobian requires the computation of a separate derivative of g, see (Grathwohl et al., 2019).

2) Matrix powers J_g^k are needed, which requires the knowledge of the full Jacobian.

3) The series is infinite.

Fortunately, drawback 1) and 2) can be alleviated. First, vector-Jacobian products $v^T J_g$ can be computed at approximately the same cost as evaluating g through reverse-mode automatic differentiation (see Definition 2.28). Second, a stochastic approximation of the matrix trace is given by the following lemma, see (Hutchinson, 1990; Avron and Toledo, 2011):

Lemma 5.6 (Stochastic estimation of matrix trace)**.** *Let $A \in \mathbb{R}^{d\times d}$ and v be a random vector on $\mathbb{R}^d$ with $\mathbb{E}[v] = 0$ and $\mathbb{E}[vv^T] = I$, where I denotes the identity. Then, we get an unbiased estimate of the matrix trace via*

$$\text{tr}(A) = \mathbb{E}[v^T A v].$$

Proof. Following (Adams et al., 2018), we have

$$\begin{aligned}
\mathbb{E}[v^T A v] &= \mathbb{E}[\text{tr}\Big(v^T A v\Big)] && (5.7)\\
&= \mathbb{E}[\text{tr}\Big(A v v^T\Big)] && (5.8)\\
&= \text{tr}\Big(\mathbb{E}[A v v^T]\Big) && (5.9)\\
&= \text{tr}\Big(A\mathbb{E}[v v^T]\Big) && (5.10)\\
&= \text{tr}(A). && (5.11)
\end{aligned}$$

Here we used in (5.7) the trace of scalar equals the scalar, in (5.8) the invariance to cyclic permutation, in (5.9) the linearity of trace, in (5.10) the linearity of the expectation operator and matrix A and in (5.11) the assumption $\mathbb{E}[vv^T] = I$. □

Algorithm 2 Forward pass of an invertible ResNet with Lipschitz constraint and log-determinant approximation, SN denotes spectral normalization based on (4.9).

Input: data point x, network F, residual block g, number of power series terms n
for Each residual block **do**
  Lip constraint: $\hat{A}_j := \mathrm{SN}(A_j, x)$ for linear Layer A_j.
  Draw v from $\mathcal{N}(0, I)$ (single vector v for trace estimation or multiple vectors)
  $w^T := v^T$
  $\log\det := 0$
  for $k = 1$ **to** n **do**
    $w^T := w^T J_g$ (vector-Jacobian product)
    $\log\det := \log\det + [(-1)^{k+1} w^T v/k]$
  end for
end for

This estimator is known as the Hutchinson's trace estimator and can be used to estimate $\mathrm{tr}\left(J_g^k\right)$. Common choices of distributions for the random vectors v are Gaussian or Rademacher distributions, see (Avron and Toledo, 2011).

While this allows for an unbiased estimate of the matrix trace, the computational cost is still unbounded due to drawback 3). Hence, the power series (5.6) will be truncated at index n. For a summary of the proposed steps, we refer to Algorithm 2.

Remark 5.7 (Biased stochastic gradient descent (SGD))**.** *In addition to randomly sampling training subsets, the unbiased trace estimator introduces a second source of randomness in the computation of gradients used for SGD (see Definition 2.26). However, the truncation turns the unbiased estimator into a biased estimator, where the bias depends on the truncation error. Fortunately, this error can be bounded as we demonstrate in the next section.*

5.2.3 Error of Power Series Truncation

We estimate $\log|\det(I + J_g)|$ with the finite power series

$$PS(J_g, n) := \sum_{k=1}^{n} (-1)^{k+1} \frac{\mathrm{tr}\left(J_g^k\right)}{k}, \tag{5.12}$$

where we have (with some abuse of notation) $PS(J_g, \infty) = \mathrm{tr}(\log(I + J_g))$. In particular, we are interested in bounding the truncation error of the log-determinant as a function of the data dimension d, the Lipschitz constant $\mathrm{Lip}(g)$ and the number of terms in the series n.

Theorem 5.8 (Approximation error of loss)**.** *Let g denote the residual function and J_g the Jacobian as before. Then, the error of a truncated power series at term n is bounded as*

$$|PS(J_g, n) - \log\det(I + J_g)| \leq -d \left(\log(1 - \mathrm{Lip}(g)) + \sum_{k=1}^{n} \frac{\mathrm{Lip}(g)^k}{k} \right).$$

Proof. We begin by noting that

$$\begin{aligned}
|PS(J_g, n) - \operatorname{tr}\log(J_g)| &= \left| \sum_{k=n+1}^{\infty} (-1)^{k+1} \frac{\operatorname{tr}\left(J_g^k\right)}{k} \right| \\
&\leq \sum_{k=n+1}^{\infty} \left| (-1)^{k+1} \frac{\operatorname{tr}\left(J_g^k\right)}{k} \right| \\
&\leq \sum_{k=n+1}^{\infty} \left| \frac{\operatorname{tr}\left(J_g^k\right)}{k} \right| \\
&\leq d \sum_{k=n+1}^{\infty} \frac{\operatorname{Lip}(g)^k}{k}, \qquad (5.13)
\end{aligned}$$

where inequality (5.13) follows from

$$\begin{aligned}
|\operatorname{tr}\left(J_g^k\right)| \leq \left| \sum_{i=1}^{d} \lambda_i(J_g^k) \right| &\leq \sum_{i=1}^{d} |\lambda_i(J_g^k)| \leq d\rho(J_g^k) \\
&\leq d\|J_g^k\|_2 \leq d\|J_g\|_2^k \leq d \operatorname{Lip}(g)^k. \qquad (5.14)
\end{aligned}$$

We note that the full series can be written as

$$\sum_{k=1}^{\infty} \frac{\operatorname{Lip}(g)^k}{k} = -\log(1 - \operatorname{Lip}(g)).$$

Hence, we can bound the approximation error by

$$|PS(J_g, n) - \operatorname{tr}\log(J_g)| \leq -d \left(\log(1 - \operatorname{Lip}(g)) + \sum_{k=1}^{n} \frac{\operatorname{Lip}(g)^k}{k} \right).$$

□

While the result above gives an error bound for the evaluation of the loss, the error in the gradient of the loss is of greater interest during training. Similarly to the preceding result, we can make a statement about the error:

Theorem 5.9 (Convergence rate of gradient approximation)**.** *Let $\theta \in \mathbb{R}^p$ denote the parameters of network F and let g, J_g be as before. Further, assume bounded inputs and a Lipschitz activation function with Lipschitz derivative. Then, we obtain the convergence rate*

$$\|\nabla_\theta \left(\log\det\left(I + J_g\right)\right) - PS(J_g, n)) \|_\infty = \mathcal{O}(c^n),$$

where $c := \operatorname{Lip}(g)$ and n denotes the number of terms used in the power series.

Proof. First, we derive an expression for the derivative with respect to parameter θ_i via

$$\begin{aligned}
&\frac{\partial}{\partial \theta_i} \log\det\left(I + J_g(x,\theta)\right) \\
&= \frac{1}{\det(I + J_g(x,\theta))}\left[\frac{\partial}{\partial \theta_i}\det\left(I + J_g(x,\theta)\right)\right] && (5.15)\\
&= \frac{1}{\det(I + J_g(x,\theta))}\left[\det(I + J_g(x,\theta))\,\mathrm{tr}\left((I + J(x,\theta))^{-1}\frac{\partial(I + J_g(x,\theta))}{\partial \theta_i}\right)\right] && (5.16)\\
&= \mathrm{tr}\left((I + J_g(x,\theta))^{-1}\frac{\partial(I + J_g(x,\theta))}{\partial \theta_i}\right)\\
&= \mathrm{tr}\left((I + J_g(x,\theta))^{-1}\frac{\partial(J_g(x,\theta))}{\partial \theta_i}\right)\\
&= \mathrm{tr}\left(\left[\sum_{k=0}^{\infty}(-1)^k J_g(x,\theta)^k\right]\frac{\partial(J_g(x,\theta))}{\partial \theta_i}\right). && (5.17)
\end{aligned}$$

Note, that (5.15) follows from the chain rule of differentiation. For the derivative of the determinant in (5.16), see (Petersen and Pedersen, 2012, Eq. 46). Furthermore, (5.17) follows from the properties of a Neumann series which converges due to $\|J_g(x,\theta)\|_2 < 1$.

By definition of $\|\cdot\|_\infty$, it is

$$\|\nabla_\theta PS(J_g(\theta),\infty) - \nabla_\theta PS(J_g(\theta),n)\|_\infty = \max_{i=1,\dots,p}\left|\frac{\partial}{\partial \theta_i}PS(J_g(\theta),\infty) - \frac{\partial}{\partial \theta_i}PS(J_g(\theta),n)\right|,$$

which is why, we consider an arbitrary i from now on. We have

$$\begin{aligned}
\left|\frac{\partial}{\partial \theta_i}PS(J_g(\theta),\infty) - \frac{\partial}{\partial \theta_i}PS(J_g(\theta),n)\right| &= \sum_{k=n+1}^{\infty}(-1)^k\,\mathrm{tr}\left(J_g^k(x,\theta)\frac{\partial(J_g(x,\theta))}{\partial \theta_i}\right)\\
&\leq d\sum_{k=n+1}^{\infty}\mathrm{Lip}(g)^K\left\|\frac{\partial J_g(x,\theta)}{\partial \theta_i}\right\|_2, && (5.18)
\end{aligned}$$

where we used the same arguments as in estimation (5.14) (in proof of Theorem 5.8).

In order to bound $\left\|\frac{\partial J_g(x,\theta)}{\partial \theta_i}\right\|_2$, we need to look into the design of the residual block. We assume contractive and element-wise activation functions (hence $\phi'(\cdot) < 1$) and N linear layers A_i in a residual block. Then, we can write the Jacobian as a matrix product

$$J_g(x,\theta) = A_N^T D_N \cdots A_1^T D_1,$$

where $D_i = diag(\phi'(z_{i-1}))$ with pre-activations $z_{i-1} \in \mathbb{R}^d$.

Since we need to bound the derivative of the Jacobian with respect to weights θ_i, double backpropagation (Drucker and Lecun, 1992) is necessary. In general, the terms $\|A_i^T\|_2$, $\|D_i\|_2$, $\|D_i^*\|_2 := \|diag(\phi''(z_{i-1}))\|_2$, $\left\|\left(\frac{\partial A_i}{\partial \theta_i}\right)\right\|_2$ and $\|x\|_2$ appear in the bound of the derivative, see (Etmann, 2019). Hence, in order to bound $\left\|\frac{\partial J_g(x,\theta)}{\partial \theta_i}\right\|_2$, we bound the previous

terms as follows

$$\|A_i^T\|_2 \leq \mathrm{Lip}(g) \tag{5.19}$$
$$\|D_i\|_2 \leq const, \tag{5.20}$$
$$\|D_i^*\|_2 \leq const \tag{5.21}$$
$$\|x\|_2 \leq const \tag{5.22}$$
$$\left\| \left(\frac{\partial A_i}{\partial \theta_i} \right)^T \right\|_2 \leq \|A_i\|_F + s. \tag{5.23}$$

The bound (5.20) is due to the assumption of a Lipschitz activation function and (5.21) is due to assuming a Lipschitz derivative of the activation function. Note, that we are using continuously differentiable activations functions (i.e. not ReLU). This assumptions holds for common functions like ELU, softplus and tanh. Furthermore, (5.22) holds by assuming bounded inputs and due to the network being Lipschitz continuous. To understand the bound (5.23), we denote s as the amount of parameter sharing of θ_i. For example, if θ_i is a entry from a convolution kernel, $s = w \cdot h$ with w spatial width and h spatial height. Let $\|\cdot\|_F$ denote the Frobenius-norm. Then

$$\left\| \left(\frac{\partial A_i}{\partial \theta_i} \right)^T \right\|_2 \leq \|A_i\|_F + s,$$

since

$$\frac{\partial A_{lm}(x,\theta)}{\partial \theta_i} = \begin{cases} 1, & \text{if } A_{lm} = \theta_i \\ 0, & \text{else} \end{cases}.$$

Hence, each term appearing in the second derivative $\left\| \frac{\partial J_g(x,\theta)}{\partial \theta_i} \right\|_2$ is bounded. We denote this bound by $a(g,\theta,x) < \infty$. Note that we do not give an exact bound on $\left\| \frac{\partial J_g(x,\theta)}{\partial \theta_i} \right\|_2$, since we are only interested in the existence of such a bound in order to proof the claimed convergence rate.

Inserting above in (5.18) and denoting $c := \mathrm{Lip}(g)$, yields

$$\begin{aligned} \left| \frac{\partial}{\partial \theta_i} PS(J_g(\theta), \infty) - \frac{\partial}{\partial \theta_i} PS(J_g(\theta), n) \right| &\leq d\, a(g,\theta,x) \sum_{k=n+1}^{\infty} c^K \\ &= \tilde{a}(d,g,\theta,x) \left(\frac{1}{1-c} - \left(\frac{1-c^n}{1-c} + c^n \right) \right). \end{aligned}$$

Letting $f(n) := \tilde{a}(d,g,\theta,x) \left(\frac{1}{1-c} - \left(\frac{1-c^n}{1-c} + c^n \right) \right)$ and $g(n) = c^n$, then

$$\lim_{n\to\infty} \left| \frac{f(n)}{g(n)} \right| = const < \infty,$$

which proves the claimed convergence rate. □

5.3 Dequantization and Evaluation

So far, we discussed how to fit a continuous density model with inputs $x \in \mathbb{R}^d$. However, many practical applications involve quantized inputs. For example, RGB-images are quantized to $x \in \{0, \dots, 255\}^d$. Since the discrete data distribution has differential entropy of negative infinity, fitting a continuous density model to discrete data can lead to arbitrarily high likelihoods even on test data (Theis et al., 2016).

For this reason, the discrete data is usually dequantized using uniform noise. Fortunately, this lead to a lower bound on the log-likelihood of the discrete data, see (Theis et al., 2016):

Lemma 5.10 (Lower bound on discrete data log-likelihood)**.** *Let $x \in \{0, \dots, 255\}$ be the discrete data and $y = x + u$ the dequantized data, where u is drawn uniformly from $[0, 1)^d$. Further, denote the discrete probability mass function as $P_{model}(x)$ and $p_{model}(x)$ as the density on the dequantized data. Then, we obtain the lower bound*

$$\mathbb{E}_{y \sim p_{data}}[\log p_{model}(y)] \leq \mathbb{E}_{y \sim P_{data}}[\log P_{model}(y)],$$

where P_{data} denotes the original distribution of the discrete data and p_{data} the density of the dequantized data.

Proof. Following (Theis et al., 2016), by Jensen's inequality we have

$$\begin{aligned}
\mathbb{E}_{y \sim p_{data}}[\log p_{model}(y)] &= \sum_x P_{data}(x) \int_{[0,1)^d} \log p_{model}(x + u) du \\
&\leq \sum_x P_{data}(x) \log \int_{[0,1)^d} p_{model}(x + u) du \\
&= \mathbb{E}_{y \sim P_{data}}[\log P_{model}(y)].
\end{aligned}$$

□

As a consequence, maximizing the log-likelihood of the continuous model on the dequantized data cannot lead to degenerate solutions, since its bounded from above by the log-likelihood of a discrete model. However, (Ho et al., 2019) argue that assigning uniform density to a hypercube $x + [0, 1)^d$ results in an unnatural task for continuous density models like neural networks. For this reason, (Ho et al., 2019) go a step further and explore variational dequantization. While this is an interesting further research direction for i-ResNets, we use uniform dequantization in the following experiments since it is more commonly used.

Evaluation: Log-likelihoods are usually reported in bits/dim, hence for evaluation we need to transform the logarithm with base e to a base of 2. Furthermore, the input data from $\{0, \dots, 255\}^d$ is usually scaled to $[0, 1]^d$, which can be accounted for by adding $\log_2(256)$. Lastly, dividing by the input dimension d allows to compare results across input domain with different dimension.

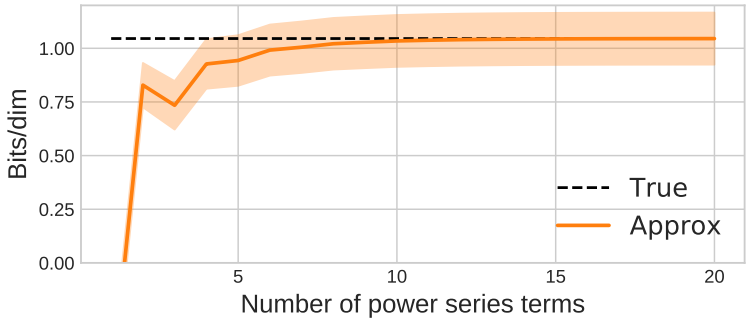


Figure 5.3: Bias and standard deviation of our log-determinant estimator as the number of power series terms increases. Variance is due to the stochastic trace estimator.

5.4 Numerical Analysis

We run a number of experiments to verify the utility of i-ResNets for building generative models. First, Figure 5.2 shows the density learned by an i-ResNet compared to Glow (Kingma and Dhariwal, 2018) on 2-dimensional synthetic tasks. For details on this experiment, we refer to (Behrmann et al., 2019***).

Second, we evaluate i-ResNets as a generative model for images on MNIST and CIFAR10. Our models consist of multiple invertible residual layers followed by invertible downsampling or dimension "squeezing" to downsample the spatial dimensions. See (Jacobsen et al., 2018) for more details on these downsampling operations. Furthermore, we use multi-scale architectures like those of (Dinh et al., 2017; Kingma and Dhariwal, 2018). In these experiments we train i-ResNets using the log-determinant approximation from Algorithm 2 with a single random vector v for trace estimation (single vector for each input sample and each residual layer). A complete description of the architectures, as well as experimental and evaluation details can be found in Appendix A.2.

In terms of the computation times, the log-determinant approximation with 5 series terms roughly increases the computation times by a factor of 4 (in comparison to the classification model). Furthermore, we plot the bias and variance of our log-determinant estimator in Figure 5.3. To get a visual idea of the trained generative model, we show some random samples from our CIFAR10 model in Figure 5.4.

The resulting performance in bits/dim and a comparison to other generative models can be found in Table 5.1. While our models did not perform as well as Glow (Kingma and Dhariwal, 2018) and FFJORD (Grathwohl et al., 2019), we find it intriguing that ResNets, with very little modification, can create a generative model competitive with these highly engineered models. Even more interesting, follow-up work from (Chen et al., 2019***) showed that constructing an unbiased estimator can close this gap in performance and improve upon all other results from Table 5.1.

Figure 5.4: CIFAR10 samples from our i-ResNet, see Appendix A.2 for details.

Method	MNIST	CIFAR10
NICE (Dinh et al., 2014)	4.36	4.48†
MADE (Germain et al., 2015)	2.04	5.67
MAF (Papamakarios et al., 2017)	1.89	4.31
Real NVP (Dinh et al., 2017)	1.06	3.49
Glow (Kingma and Dhariwal, 2018)	1.05	3.35
FFJORD (Grathwohl et al., 2019)	0.99	3.40
i-ResNet	1.06	3.45

Table 5.1: MNIST and CIFAR10 bits/dim results compared to other likelihood-based models. † Uses ZCA preprocessing making results not directly comparable.

5.5 Related Work

After introducing i-ResNets in the previous chapter and discussing how to use them as a generative model, we now review other approaches to design invertible networks. Afterwards, we shortly discuss the growing body of literature on the approximation of spectral functions like the log-determinant.

5.5.1 Comparison of Invertible Architectures

We put our focus on invertible architectures with efficient inverse computation, namely NICE (Dinh et al., 2014), i-RevNet (Jacobsen et al., 2018), Real-NVP (Dinh et al., 2017), Glow (Kingma and Dhariwal, 2018) and Neural ODEs (Chen et al., 2018) and its stochastic density estimator FFJORD (Grathwohl et al., 2019). A summary of the comparison between different invertible networks is given in Table 5.2.

The dimension-splitting approach used in NICE, i-RevNet and Real-NVP results in analytic forward and inverse mappings. However, this restriction required the introduction of additional steps like invertible 1×1 convolutions in Glow (Kingma and Dhariwal, 2018). These 1×1 convolutions need to be inverted numerically, making Glow altogether not analytically invertible. In contrast, i-ResNet can be viewed as an intermediate approach, where the forward mapping is given in closed-form, while the inverse can be approximated via a fixed-point iteration.

Method	ResNet	NICE	Real-NVP	Glow	FFJORD	i-ResNet
Free-form	✓	✗	✗	✗	✓	✓
Analytic Forward	✓	✓	✓	✓	✗	✓
Analytic Inverse	N/A	✓	✓	✗	✗	✗
Non-volume Preserving	N/A	✗	✓	✓	✓	✓
Exact Likelihood	N/A	✓	✓	✓	✗	✗
Unbiased Stoch. Log-Det Est.	N/A	N/A	N/A	N/A	✓	✗

Table 5.2: Comparing i-ResNet and ResNets to NICE (Dinh et al., 2014), Real-NVP (Dinh et al., 2017), Glow (Kingma and Dhariwal, 2018) and FFJORD (Grathwohl et al., 2019). Non-volume preserving refers to the ability to allow for contraction and expansions and exact likelihood to compute the change of variables (5.4) exactly. The unbiased estimator refers to a stochastic approximation of the log-determinant, see section 5.2.2. We note, however, that the follow-up work (Chen et al., 2019***) introduces an unbiased estimator for i-ResNet architectures.

Furthermore, an i-ResNet block has a Lipschitz bound both for forward and inverse (Lemma 4.6), while other approaches do not have this property by construction. Thus, employing i-ResNets in stability-critical applications could be a promising avenue for future work. Particularly interesting would be the usage of invertible ResNets in inverse problems, see e.g. (Ardizzone et al., 2019) for first studies in this area.

Neural ODEs (Chen et al., 2018) allow free-form dynamics similar to i-ResNets, meaning that any architecture could be used as long as the input and output dimensions are the same. To obtain discrete forward and inverse dynamics, Neural ODEs rely on adaptive ODE solvers, which allows for an accuracy vs. speed tradeoff. Yet, scalability to very high input dimension such as high-resolution images remains unclear.

5.5.2 Spectral Sum Approximations

The approximation of spectral sums like the log-determinant is of broad interest for many other machine learning applications such as Gaussian Process regression (Dong et al., 2017). Among others, Taylor approximation (Boutsidis et al., 2017) of the log-determinant similar to our approach or Chebyshev polynomials (Han et al., 2016) are used. In (Boutsidis et al., 2017), error bounds on the estimation via truncated power series and stochastic trace estimation are given for symmetric positive definite matrices. However, $I + J_g$ is not symmetric in our case and thus, their analysis does not apply here.

Recently, unbiased estimates (Adams et al., 2018) and unbiased gradient estimators (Han et al., 2018) were proposed for symmetric positive definite matrices. Furthermore, Chebyshev polynomials have been used to approximate the log-determinant of Jacobian of deep neural networks in (Ramesh and LeCun, 2018) to evaluate the likelihood of GANs.

5.6 Conclusion and Future Work

Generative modeling is often understood as a key step towards a holistic understanding of data. Among generative approaches, normalizing flows stand out since they allow to use the maximum likelihood principle by leveraging invertible transforms for a change of variables. Due to their flexibility, i-ResNets are suitable candidates for such generative models. They allow a free-form Jacobian (no structural constraint), in contrast to other approaches based on dimension splitting (Dinh et al., 2014, 2017; Kingma and Dhariwal, 2018). Yet, this free-form has one major drawback: the log-determinant needs to be computed, which is prohibitive in high-dimensions. To circumvent this problem, we introduced a simple estimator and demonstrated its usefulness for image data.

The log-determinant estimator introduces two major disadvantages:

(i) Additional memory-cost due to saving all summands of the power series for back-propagation

(ii) Biased estimation of the log-determinant.

However, both disadvantages can be addressed. First, a memory-efficient gradient estimator can be derived based on a Neumann series. Second, the estimator presented in section 5.2.2 can be made unbiased via randomizing the truncation index and reweighting each summand. These changes not only allows training i-ResNets with less memory resources, it further improves the results presented in this chapter. For details on this, we refer to the follow-up work:

> Ricky T. Q. Chen, **Jens Behrmann**, David Duvenaud, Jörn-Henrik Jacobsen: *Residual flows for invertible generative modeling*, 2019, (under submission; early version presented at: ICML workshop on Invertible Networks and Normalizing Flows)

While the follow-up work above already removed multiple drawbacks, future work could focus on further improving the presented methodology. In particular, a better understanding of the variance of the used log-determinant estimator might lead to variance reduction approaches, which may speedup convergence. Furthermore, studying whether i-ResNets (or other invertible architectures) can approximate any distribution via a change of variables could guide future algorithmic research.

Besides a mathematical analysis, using these flow-based models in multiple applications should be one of the next steps. For example, maximum-a-posteriori (MAP) inference, where the prior is learned via an i-ResNet, could offer a novel deep learning approach to inverse problems. Additionally, using flows for out-of-distribution detection or semi-supervised learning could further increase the application spectrum of invertible networks.

Chapter 6

Reverse View on Adversarial Examples

"All models are wrong, but some are deadly." - Nassim Taleb.

Since the discovery of adversarial examples (Biggio et al., 2013; Szegedy et al., 2014), they were mostly studied via worst-case robustness under an attack to the model. In this chapter, we introduce a second viewpoint on adversarial examples by reversing this definition. This reverse view uncovers excessive invariance of deep networks to changes in the input space, which might then be exploited by an adversary.

Since invariance is hard to control when using standard architectures (see chapter 3), we study the usage of invertible networks, which are (by design) not invariant to any transformations in the input. When performing classification, however, we need to project features onto classes which necessarily introduces invariant directions. Yet, as we will show, invertible networks still offer a mechanism to control invariance.

This chapter focuses on conceptual insights and theoretical considerations, which are based on the articles below. We also briefly review invariance-based attacks and further empirical studies, but encourage the reader to consult the following articles for an in-depth:

> Jörn-Henrik Jacobsen, **Jens Behrmann**, Richard Zemel, Matthias Bethge: *Excessive invariance causes adversarial vulnerability*, 2019, (International Conference on Learning Representations (ICLR))
>
> Jörn-Henrik Jacobsen, **Jens Behrmann**, Nicholas Carlini, Florian Tramer, Nicolas Papernot: *Exploiting excessive invariance caused by norm-Bounded adversarial robustness*, 2019, (under submission; early version presented at: ICLR workshop on Safe Machine Learning: Specification, Robustness and Assurance)

6.1 Introduction and Motivation

Research on adversarial examples is motivated by a spectrum of questions. These range from the security of models deployed in the presence of real-world adversaries to the need

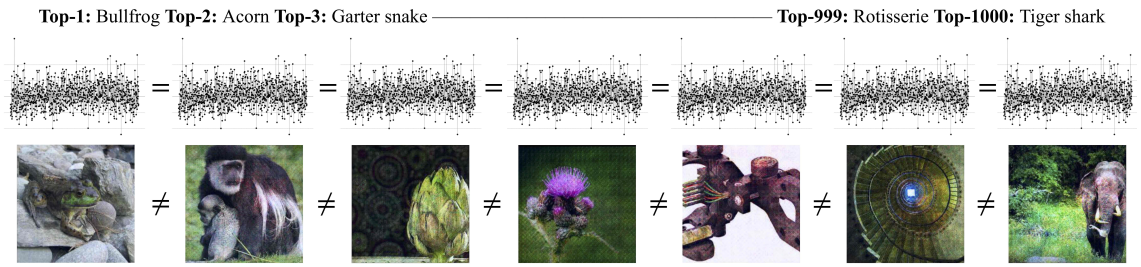


Figure 6.1: From (Jacobsen et al., 2019b***): All images shown cause a competitive ImageNet-trained network to output the *exact same* probabilities over all 1000 classes (logits shown above each image). The leftmost image is from the ImageNet validation set; all other images are constructed such that they match the non-class related information of images taken from other classes (for details see section 6.3). The excessive invariance revealed by this set of adversarial examples demonstrates that the logits contain only a small fraction of the information perceptually relevant to humans for discrimination between the classes.

to capture limitations of representations and their (in)ability to generalize from training data (Gilmer et al., 2018a). The broadest accepted definition of an adversarial example is "an input to a ML model that is intentionally designed by an attacker to fool the model into producing an incorrect output" (Goodfellow and Papernot, 2017). To enable concrete progress, many definitions of adversarial examples were introduced in the literature since their initial discovery (Biggio et al., 2013; Szegedy et al., 2014). Each of them employs a different degree of formalism and corresponds to a distinct set of assumptions about the adversary, i.e., *threat model*.

So far, the study of adversarial examples has mostly been concerned with the setting of small perturbation, or ϵ-adversaries (Goodfellow et al., 2015; Madry et al., 2017; Raghunathan et al., 2018). Perturbation-based adversarial examples are appealing because they allow to quantitatively measure notions of adversarial robustness (Brendel et al., 2018). However, recent work argued that the perturbation-based approach is unrealistically restrictive and called for the need of generalizing the concept of adversarial examples to the unrestricted case (Song et al., 2018; Brown et al., 2018). Yet, settings beyond ϵ-robustness are hard to formalize (Gilmer et al., 2018a).

We argue here for an alternative, complementary viewpoint on the problem of adversarial examples. Rather than perturbing the input to change the classifier's output, *invariance-based adversarial examples* are obtained by modifying the input as much as possible, while keeping the decision of the classifier *identical*. Accordingly, they correspond to a *lack* of sensitivity of the model: the model's constant prediction does not reflect the expected change in the input's true label. In Figure 6.1, an illustration of such model failures is shown.

The invariance perspective suggests that adversarial vulnerability is a consequence of narrow learning, yielding classifiers that rely only on few highly predictive features in their decisions and fail to capture all predictive features. This has also been supported by the observation that deep networks strongly rely on spectral statistical regularities (Jo

and Bengio, 2017), or stationary statistics (Gatys et al., 2015) to make their decisions, rather than more abstract features like shape and appearance. We hypothesize that a major reason for this excessive invariance can be understood from an information-theoretic viewpoint of cross-entropy training. This procedure maximizes a bound on the mutual information between labels and representation, thus it is giving no incentive to explain all class-dependent aspects of the input. This may be desirable in some cases, but to achieve truly general understanding of a scene or an object, machine learning models have to learn to successfully separate essence from nuisance and subsequently generalize even under shifted input distributions (e.g. due to distortion of inputs or due to using the model in a new environment).

6.2 Two Complementary Approaches to Adversarial Examples

In this section, we define preimages and establish a link to adversarial examples.

Definition 6.1 (Preimages / Invariance)**.** *Let $F : \mathbb{R}^d \to \mathbb{R}^C$ be a neural network, where $F = f^L \circ \cdots \circ f^1$ with layers f^i and let F^i denote the network up to layer i. Further, let $D : \mathbb{R}^d \to \{1, \ldots, C\}$ be a classifier with $D = \operatorname{argmax}_{k=1,\ldots,C} \operatorname{softmax}(F(x))_k$. Then, for input $x \in \mathbb{R}^d$, we define the following* ***preimages****:*

(i) i-th layer preimage: $\{x^* \in \mathbb{R}^d \mid F^i(x^*) = F^i(x)\}$

(ii) Logit preimage: $\{x^* \in \mathbb{R}^d \mid F(x^*) = F(x)\}$

(iii) Argmax preimage: $\{x^* \in \mathbb{R}^d \mid D(x^*) = D(x)\}$,

where (i) $\subset$ (ii) $\subset$ (iii) by the compositional nature of classifier D.
Let G denote the subnetwork F^i, the entire network F or the classifier D. Then, we say that the (sub-)network G is ***invariant*** *to perturbations Δx if $G(x + \Delta x) = G(x)$.*

Non-singleton preimages (preimages containing more elements than input x) after the i-th layer can occur, if the chain $f^i \circ \cdots \circ f^1$ is not injective, for instance due to subsampling or non-injective activation functions like ReLU. For a discussion on the effect of ReLU on preimages see chapter 3. This accumulated invariance can become problematic if not controlled properly, as we will show in the following.

We define perturbation-based adversarial examples by introducing the notion of an oracle (e.g., a human decision-maker or the unknown input-output function considered in learning theory):

Definition 6.2 (Perturbation-based Adversarial Examples)**.** *Let G denote the i-th layer, logit or argmax of the classifier. A* ***perturbation-based adversarial example*** *(or perturbation adversarial) $x^* \in \mathbb{R}^d$ corresponding to a legitimate test input $x \in \mathbb{R}^d$ fulfills:*

(i) Created by adversary: $x^* \in \mathbb{R}^d$ is created by an algorithm $\mathcal{A} : \mathbb{R}^d \to \mathbb{R}^d$ with $x^* = \mathcal{A}(x)$.

(ii) Perturbation of output: $\|G(x^*) - G(x)\| > \delta$ *and* $\mathcal{O}(x^*) = \mathcal{O}(x)$*, where perturbation* $\delta > 0$ *is set by the adversary and* $\mathcal{O} : \mathbb{R}^d \to \{1, \dots, C\}$ *denotes the* ***oracle****.*

Furthermore, x^* *is* ϵ***-bounded*** *if* $\|x - x^*\| < \epsilon$*, where* $\|\cdot\|$ *is a norm on* $\mathbb{R}^d$ *and* $\epsilon > 0$*.*

Property (i) allows us to distinguish perturbation-based adversarial examples from points that are misclassified by the model without adversarial intervention. Furthermore, the above definition incorporates adversarial perturbations designed for hidden features as in (Sabour et al., 2016), while usually the decision of the classifier D (argmax-operation on logits) is used as the perturbation target. Our definition also identifies ϵ-bounded perturbation-based adversarial examples (Goodfellow et al., 2015) as a specific case of unbounded perturbation-based adversarial examples. However, as our goal is to characterize general invariances of the network, we do not restrict ourselves to bounded perturbations.

Definition 6.3 (Invariance-based Adversarial Examples)**.** *Let* G *denote the* i*-th layer, logit or argmax of the classifier. An* ***invariance-based adversarial example*** $x^* \in \mathbb{R}^d$ *corresponding to a legitimate test input* $x \in \mathbb{R}^d$ *fulfills:*

(i) Created by adversary: $x^* \in \mathbb{R}^d$ *is created by an algorithm* $\mathcal{A} : \mathbb{R}^d \to \mathbb{R}^d$ *with* $x \mapsto x^*$*.*

(ii) Lies in preimage of x *under* G*:* $G(x^*) = G(x)$ *and* $\mathcal{O}(x) \neq \mathcal{O}(x^*)$*, where* $\mathcal{O} : \mathbb{R}^d \to \{1, \dots, C\}$ *denotes the* ***oracle****.*

As a consequence, $D(x) = D(x^*)$ also holds for invariance-based adversarial examples, where D is the classifier. Intuitively, adversarial perturbations cause the output of the classifier to change while the oracle would still consider the new input x^* as being from the original class. Whereas perturbation-based adversarial examples exploit the classifier's *excessive sensitivity in task-irrelevant directions*, invariance-based adversarial examples explore the classifier's preimage to identify *excessive invariance in task-relevant directions*: its prediction is unchanged while the oracle's output differs. Briefly put, perturbation-based and invariance-based adversarial examples are complementary failure modes of the learned classifier.

When not restricting to ϵ-perturbations, perturbation-based and invariance-based adversarial examples yield the same input x^* via

$$x^* = x_1 + \Delta x_1, \quad D(x^*) \neq D(x_1), \quad \mathcal{O}(x^*) = \mathcal{O}(x_1) \tag{6.1}$$

$$x^* = x_2 + \Delta x_2, \quad D(x^*) = D(x_2), \quad \mathcal{O}(x^*) \neq \mathcal{O}(x_2), \tag{6.2}$$

with different reference points x_1 and x_2 (see Figure 6.2). Hence, the **key difference is the change of reference**, which allows us to approach these failure modes from different directions. To connect these failure modes with an intuitive understanding of variations in the data, we now introduce the notion of invariance to nuisance and semantic variations, see also (Achille and Soatto, 2018).

Definition 6.4 (Semantic and nuisance perturbations of an input)**.** *Let* $\mathcal{O}$ *be an oracle (Definition 6.2) and* $x \in \mathbb{R}^d$*. Then, a perturbation* Δx *of an input* $x \in \mathbb{R}^d$ *is called* ***semantic****, if* $\mathcal{O}(x) \neq \mathcal{O}(x + \Delta x)$ *and* ***nuisance*** *if* $\mathcal{O}(x) = \mathcal{O}(x + \Delta x)$*.*

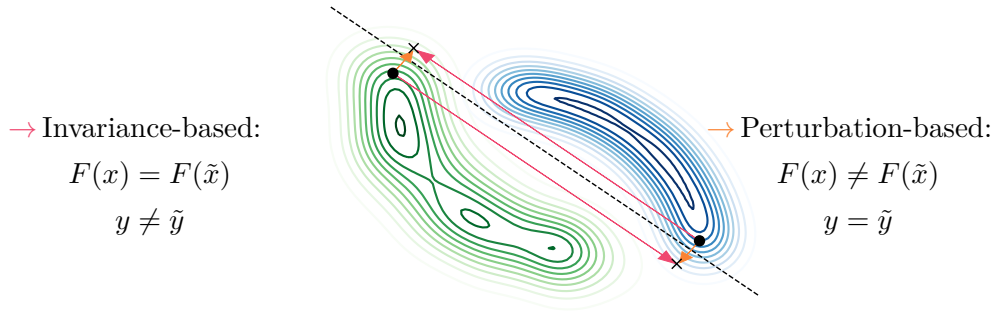


Figure 6.2: Connection between invariance-based (long pink arrow) and perturbation-based adversarial examples (short orange arrow). Class distributions are shown in green and blue; the dashed line is the decision-boundary of a classifier. All adversarial examples can be reached either by crossing the decision-boundary of the classifier via perturbations, or by moving within the preimage of the classifier to mis-classified regions. The two viewpoints are *complementary* to one another and highlight that adversarial vulnerability is not only caused by excessive *sensitivity* to semantically meaningless perturbations, but also by excessive *insensitivity* (invariance) to semantically meaningful transformations.

For example, such a nuisance perturbation could be a translation or occlusion in image classification. However, these concepts are easier to understand for following synthetic example, where nuisance and semantics can be explicitly formalized as rotation and norm scaling:

Example 6.5 (Semantic and nuisance on Adversarial Spheres (Gilmer et al., 2018b))**.** *Consider the task: Distinguish points from two cocentric spheres (class 1:* $\|x\|_2 = R_1$ *and class 2:* $\|x\|_2 = R_1$*). Further, let* (r, ϕ) *denote the spherical coordinates of* x*. Then, any perturbation* Δx*,* $x^* = x + \Delta x$ *with* $r^* \neq r$ *is semantic. On the other hand, if* $r^* = r$*, then the perturbation is a nuisance with respect to the task of discriminating two spheres.*

In this example, the max-margin classifier $D(x) = sign\left(\|x\| - \frac{R_1+R_2}{2}\right)$ is invariant to any nuisance perturbation, while being only sensitive to semantic perturbations. In summary, the transformation to spherical coordinates allows to linearize semantic and nuisance perturbations. Using this notion, invariance-based adversarial examples can be attributed to perturbations of $x^* = x + \Delta x$ with the following two properties:

(i) The perturbed sample x^* stays in the preimage $\{x^* \in \mathbb{R}^d \mid D(x^*) = D(x)\}$ of the classifier

(ii) The perturbation Δx is semantic, as $\mathcal{O}(x) \neq \mathcal{O}(x + \Delta x)$.

Thus, the failure of the classifier D can be thought of a mis-alignment between its invariance (expressed through the preimage) and the semantics of the data/ task (expressed by the oracle).

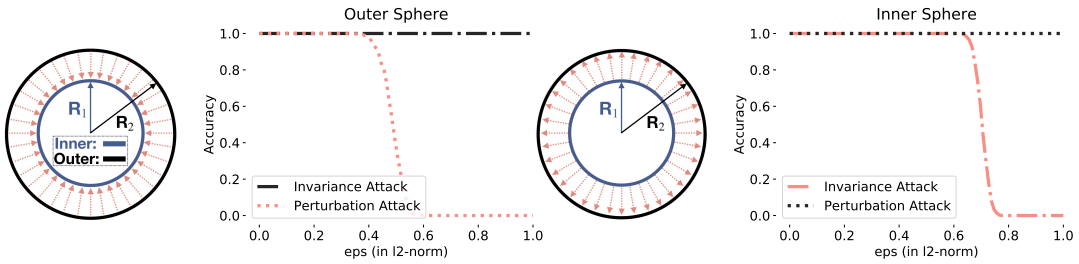


Figure 6.3: Robustness experiment on spheres with radii $R_1 = 1$ and $R_2 = 1.3$ and mis-aligned classifier for $d = 500$ dimensions and $n = 10$ unused dimensions. **Left:** Attacking points from the outer sphere with perturbation-based attacks, with accuracy dropping when increasing the upper bound on l2-norm perturbations. **Right:** Attacking points from the inner sphere with invariance-based attacks, with accuracy dropping when increasing the upper bound on ℓ_2-norm perturbations. Each attack mode has a different effect on the manifold. Red arrows indicate the only possible direction of attack for each sphere. Perturbation attacks fail on the inner sphere, while invariance attacks fail on the outer sphere. Hence, both attacks are needed for a full account of model failures.

6.2.1 Comparing Invariance-based and Perturbation-based Robustness

We now investigate the relationship between the two adversarial example definitions from the previous section. In a general setting, invariance and stability may be uncoupled. For this consider a linear classifier with matrix A. The perturbation-robustness is tightly related to forward stability, since the effect of an input perturbation can be upper bounded by the largest singular value of A. On the other hand, the invariance-view relates to the stability of the inverse (smallest singular value of A) and to the null-space of A. As largest and smallest singular values are uncoupled when considering arbitrary matrices A, the relationship between both viewpoints is likely non-trivial in practice.

Next, we show how the analysis of perturbation-based and invariance-based adversarial examples can uncover different model failures. To do so, we again consider the synthetic *adversarial spheres problem* of (Gilmer et al., 2018b) from Example 6.5. As mentioned, the dataset was designed such that a robust (i.e. max-margin) classifier exists. Our analysis considers a similar, but slightly sub-optimal classifier in order to study model failures in a controlled setting:

Example 6.6 (Mis-aligned classifier on adversarial spheres)**.** *Let input $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$ be sampled from one of two spheres with radii $R_1 < R_2$. Then, consider the classifier*

$$D(x) = \text{sign}\left(\|x_{1,\ldots,d-n}\|_2 - b\right),$$

which computes the norm of x from its first $d - n$ cartesian coordinates and outputs -1 (respectively +1) for the inner (respectively outer) sphere. The bias b is chosen, such that the classifier D is the max-margin classifier on the (finite) training set $\mathcal{T} = \{x^{(i)}\}_{i=1}^N$ (under the assumption of separability, i.e. $l \leq u$):

$$l = \max_{\substack{x^{(i)} \in \mathcal{T} \\ \|x^{(i)}\|_2 = R_1}} \|x_{1,\ldots,d-n}\|_2, \quad u = \min_{\substack{x^{(i)} \in \mathcal{T} \\ \|x^{(i)}\|_2 = R_2}} \|x_{1,\ldots,d-n}\|_2, \quad b = l + \frac{u - l}{2}.$$

Even though this sub-optimal classifier reaches nearly 100% on finite test data, the model is imperfect in the presence of adversaries that operate on the manifold (i.e., produce adversarial examples that remain on one of the two spheres, but are misclassified by the classifier). Most interestingly, the perturbation-based and invariance-based approaches uncover different failures:

(Analytic) perturbation-based attack: All points x from the outer sphere (i.e., $\|x\|_2 = R_2$) can be perturbed to x^*, where $\mathcal{O}(x) = D(x) \neq D(x^*)$, while staying on the outer sphere (i.e., $\|x^*\|_2 = R_2$):

(i) Perturbation of decision: $x^*_{1,\dots,d-n} = a\ (x_{1,\dots,d-n})$, where scaling $a > 0$ is chosen such that $\|x^*_{1,\dots,d-n}\|_2 < b$

(ii) Projection to outer sphere: $x^*_{d-n,\dots,d} = c\ (x_{d-n,\dots,d})$, where scaling $c > 0$ is chosen such that $\|x^*_{d-n,\dots,d}\|_2 = \sqrt{R_2^2 - \|x^*_{1,\dots,d-n}\|_2^2}$.

For points x from the inner sphere, this is not possible if $b > R_1$.

(Analytic) invariance-based attack: All points x from the inner sphere ($\|x\|_2 = R_1$) can be perturbed to x^*, where $D(x) = D(x^*) \neq \mathcal{O}(x^*)$, despite being in fact on the outer sphere after the perturbation has been added (i.e., $\|x^*\|_2 = R_2$):

(i) Fixing the used dimensions: $x^*_{1,\dots,d-n} = x_{1,\dots,d-n}$

(ii) Perturbation of unused dimensions: $x^*_{d-n,\dots,d} = a\ (x_{d-n,\dots,d})$, where scaling $a > 0$ is chosen such that $\|x^*_{d-n,\dots,d}\|_2 = \sqrt{R_2^2 - \|x^*_{1,\dots,d-n}\|_2^2}$.

For points x from the outer sphere, this is not possible if $b > R_1$.

In Figure 6.3, we plot the mean accuracy over points sampled either from the inner or outer sphere. It is visualized as a function of the norm of the adversarial manipulation added to create perturbation-based and invariance-based adversarial examples. This illustrates how the robustness regime differs significantly between the two variants of adversarial examples. Therefore, by looking only at perturbation-based (respectively invariance-based) adversarial examples, important model failings may be overlooked. This is exacerbated when the data is sampled in an unbalanced fashion from the two spheres: the inner sphere is robust to perturbation adversarial examples while the outer sphere is robust to invariance adversarial examples (for accurate models).

Furthermore, (Jacobsen et al., 2019a***) showed on nearly all state-of-the-art robust MNIST models that hardening the model towards bounded perturbation-robustness can induce new invariance-based vulnerabilities. Thus when investigating a model, both failures should be taken into account. In the following section, we review recent approaches to attack a model based on excessive invariance.

6.3 Review of Invariance Attacks

There has been some recent work that pushed the boundary to understand attacks in the context of excessive invariance:

- An attack leveraging bijective networks to analytically compute invariance-based adversarial examples (Jacobsen et al., 2019b***). This attack is applicable for large-scale problems like ImageNet (see Figure 6.1) and will be discussed in more detail in this section.
- Model-agnostic attacks on MNIST to reveal invariance-based vulnerability even within small ℓ_p-norm balls (Jacobsen et al., 2019a***).
- Linear programming based attacks that exploit ReLU layer preimage vulnerabilities (Behrmann et al., 2018a***). See section 3.3 and, in particular, Theorem 3.3 for more details on the preimage of ReLu layers.
- Feature collisions that identify polytopes, in which all inputs are misclassified while causing identical activations (Li et al., 2019).
- Unrestricted adversarial examples, leveraging Generative Adversarial Networks (GANs) to generate misclassified, but realistic inputs (Song et al., 2018).

Of special interest for the discussion here is the *metameric sampling* attack from (Jacobsen et al., 2019b***), which makes explicit use of invertible networks. First, metameric sampling relies on classifiers with a simplified read-out:

Definition 6.7 (Bijective classifier with simplified read-out)**.** *Let $F_\theta : \mathbb{R}^d \to \mathbb{R}^d$ denote bijective network and denote the classifier D_θ as*

$$D_\theta = \underset{k=1,\dots,C}{\operatorname{argmax}} \operatorname{softmax}(F_\theta(x)_{1,\dots,C})_k,$$

where $C \leq d$ denotes the number of classes. Further, denote $z = F_\theta(x)$ and $z_s = z_{1,\dots,C}$ as the logits (semantic variables) and $z_n = z_{C+1,\dots,d}$ as the nuisance variables (z_n is not used for classification). Then, the classifier D_θ has a simplified read-out structure, since it only uses the first C outputs of the bijective network F_θ without further transformations.

An example of such a classifier, called fi-RevNet (fully-invertible reversible networks), was introduced in (Jacobsen et al., 2019b***). This structure allows to define a simple analytic invariance-based attack, see (Jacobsen et al., 2019b***):

Definition 6.8 (Metameric sampling)**.** *Let F_θ and D_θ define a classifier with simplified read-out and F_θ^{-1} be the inverse. Further, let $x \in \mathbb{R}^d$ be an input sample from class i and $\tilde{x} \in \mathbb{R}^d$ be a reference sample from class $j \neq i$. Then, metameric sampling generates invariance-based adversarial examples via*

$$x_{met} = F^{-1}(z_s, \tilde{z}_n),$$

where z_s are the semantic variables from input x and $\tilde{z}_n$ are the nuisance variables from the reference sample $\tilde{x}$.

Remark 6.9. *Metameric sampling only generates invariance-based adversarial examples by Definition 6.3, if the output x_{met} is* ***legitimate****. While this termed was not defined in a technical manner, for images we could think of naturally looking examples or non-suspicious examples as judged visually by a human observer. Experimentally (Jacobsen et al., 2019b***) find that this sampling procedure reveals adversarial subspaces, that are indeed visually close to natural images on ImageNet (see Figure 6.1).*

Thus, metameric sampling (which relies on invertible networks) offers an analytic tool to inspect dependencies between semantic and nuisance variables without the need for expensive and approximate optimization procedures. In conclusion, invertible networks can play a crucial role to uncover this striking behavior. In the subsequent section, we go even one step further and study following two questions:

(i) How do invertible networks allow to control invariance and why do we need invertible networks?

(ii) Can this be leveraged to avoid those model failures, which are uncovered by invariance-based adversarial examples?

6.4 Controlling the Behavior of the Inverse Mapping

6.4.1 On the Need of Invertible Networks

A particularly promising architecture class to control invariance-based robustness may be invertible networks, such as i-RevNet (Jacobsen et al., 2018) or i-ResNets (chapter 4). They cannot build any invariance up until the final layer by construction, which is typically followed by a linear layer that projects down to the class logits. Thus in invertible networks, analyzing invariance-based vulnerability boils down to analyzing the invariance caused by this final linear operator. As an example, see the classifier with simplified read-out structure (Definition 6.7).

On the other hand, controlling invariance in standard architectures is difficult. See for example Theorem 3.3, where conditions of the preimage of ReLU-layers are given. Even this single layer can show entirely different behavior over the input space. Thus, in this section we concentrate on invertible networks, which simplify the study of invariance drastically.

Remark 6.10. *While we focus on invariance in this section, extending the viewpoint to the stability of the inverse would be of major interest. Especially the previously proposed i-ResNet would be a promising model class, because it allows to control forward and inverse stability by design (see Theorem 4.6 on the Lipschitz constant of the forward and inverse mapping).*

6.4.2 Independence Cross-Entropy via Information Theory

In this section, we identify why the cross-entropy objective (or equivalently the negative log-likelihood, see Remark 2.8) does not necessarily encourage to explain all task-dependent variations of the data and propose a way to fix this. In particular, we employ

an information-theoretic viewpoint for our analysis, which is why we use the term cross-entropy instead of negative log-likelihood (for their equivalence see Lemma 2.14).

We leverage bijective classifiers with simplified readout-structure (Definition 6.7) and turn the splitting of semantics and nuisances into a formal explanation framework using information theory: Let $(X, Y) \sim \mathcal{D}_{\mathcal{X}\times\mathcal{Y}}$,where the label Y is a discrete random variable over $\{0,1\}^C$. Then, the goal of a feature extractor can be stated as maximizing the mutual information (see Definition 2.11) between semantic features Z_s (logits) extracted by network F_θ and labels Y, denoted by $I(Y; Z_s)$.

Remark 6.11. *We use capital letters to denote the random variables X, Y, Z_s, Z_n and reason about their statistical properties, while small letters are used to denote their realizations (samples x, transformed features z_s, z_n and labels y). Here, random variables Z_s and Z_n are deterministic transformations of random variable X, i.e.*

$$Z_s = F_\theta(X)_{1,\dots,C} \quad \text{and} \quad Z_n = F_\theta(X)_{C+1,\dots,d},$$

where the transform is given by an invertible network F_θ.

The previously discussed failures required a modification of the inputs $X \sim \mathcal{D}$ (we drop the subscript for convenience), for example via metameric sampling. To formalize these modifications, we now introduce the concept of an ***adversarial distribution shift $\mathcal{D}_{Adv} \neq \mathcal{D}$***. Our first assumptions for $\mathcal{D}_{Adv}$ is

$$I_{\mathcal{D}_{Adv}}(Z_n; Y) \leq I_{\mathcal{D}}(Z_n; Y).$$

Intuitively, the nuisance variables Z_n of our network do not become more informative about Y by the shift in the input distribution. Thus, $\mathcal{D}_{Adv}$ may reduce the predictiveness of features encoded in Z_s, but does not introduce or increase the predictive value of variations captured in Z_n. Second, we assume (see Def. 2.11 for conditional MI)

$$I_{\mathcal{D}_{Adv}}(Y; Z_s | Z_n) \leq I_{\mathcal{D}_{Adv}}(Y; Z_s),$$

which corresponds to positive or zero interaction information, see (Ghassami and Kiyavash, 2017). While the information in Z_s and Z_n can be redundant in this assumption, synergetic effects where conditioning on Z_n increases the mutual information between Y and Z_s are excluded.

Remark 6.12. *While the assumptions may be hard to verify or unrealistic in real-world applications, (Jacobsen et al., 2019b***) studied a synthetic example where above assumptions were valid. In particular, an additional feature (texture or watermark pixel) was planted into each image of an MNIST digit. During training time this feature was predictive of the target label, while the relationship was randomized at test time (corresponding to a distribution shift between training and test).*

At this point, we use that bijective networks F_θ capture all variations by design. In information-theoretic terms this translates to information invariance, see (Kraskov et al., 2004) and (Polyanskiy and Wu, 2015):

Lemma 6.13 (Invariance of mutual information (MI) under reparametrization)**.** *Let X be a continuous random variables on $\mathbb{R}^d$ and Y be a discrete random variable on $\{0,1\}^C$. Further, let $F : \mathbb{R}^d \to \mathbb{R}^d$ be a deterministic homomorphism. Then, mutual information is invariant under F and it holds*

$$I(Y;X) = I(Y;F(X)). \tag{6.3}$$

Proof. We follow the derivations from (Polyanskiy and Wu, 2015). Consider the chain of data processing

$$(Y,X) \mapsto (Y,F(X)) \mapsto (Y,F^{-1}(F(X))) = (Y,X).$$

Then by the data processing inequality (see Lemma 2.13, (iv)), it holds

$$I(Y;X) \geq I(Y;F(x)) \geq I(Y;F^{-1}(F(X))) = I(Y;X).$$

Hence equality (6.3) must hold, which proves the invariance under reparametrization. □

We can now apply this property of MI to the homomorphism (bijective) F_θ. Consider the splitting of $(Z_s, Z_n) = Z = F_\theta(X)$ as in Remark 6.11. Then by the chain rule of MI (see Lemma 2.13, (ii)), we can reformulate via

$$\begin{aligned} I(Y;X) = I(Y;F_\theta(X)) &= I(Y;Z_s,Z_n) \\ &= I(Y;Z_s) + I(Y;Z_n|Z_s) && (6.4) \\ &= I(Y;Z_n) + I(Y;Z_s|Z_n), && (6.5) \end{aligned}$$

where $I(Y;Z_n|Z_s)$ denotes the conditional MI (see Def. 2.11). Most interestingly, above reformulation offers two ways to increase the (conditional) MI between labels Y and semantic features Z_s:

(i) Direct increase of $I(Y;Z_s)$ (see formulation (6.4))

(ii) Indirect increase of $I(Y;Z_s|Z_n)$ via decreasing $I(Y;Z_n)$ (see formulation (6.5)).

Usually in a classification task, only $I(Y;Z_s)$ is actively increased via training a feature extractor. While this approach is sufficient in most cases, as expressed via high accuracies on training and test data, it may fail under $\mathcal{D}_{Adv}$. This highlights why cross-entropy training may not be sufficient to overcome excessive semantic invariance. However, by leveraging the bijection F_θ we can minimize the unused mutual information $I(Y;Z_n)$ using the intuition of a nuisance classifier. For an empirical analysis of a nuisance classifier, see the experiment on the adversarial spheres problem in (Jacobsen et al., 2019b***).

This nuisance classifier can be formulated via the following extended loss:

Definition 6.14 (Independence cross-entropy loss)**.** *Let $\mathcal{T} = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$ be the training set and let $F_\theta : \mathbb{R}^d \to \mathbb{R}^d$ a bijective network with parameters $\theta \in \mathbb{R}^{p_1}$ and*

$\tilde{F}_\theta(x) = \text{softmax}(F_\theta(x)_{1,\dots,C})$. *Furthermore, let* $D_{\theta_{nc}} : \mathbb{R}^{d-C} \to [0,1]^C$ *be the nuisance classifier with* $\theta_{nc} \in \mathbb{R}^{p_2}$. *Then, we define the independence cross-entropy loss as*

$$\mathcal{L}_{iCE}(\theta, \theta_{nc}, \mathcal{T}) = \underbrace{\sum_{i=1}^{N}\sum_{j=1}^{C} -y_j^{(i)} \log \tilde{F}_\theta^{z_s}(x^{(i)})_j}_{=:\mathcal{L}_{sCE}(\theta,\mathcal{T})} + \underbrace{\sum_{i=1}^{N}\sum_{j=1}^{C} y_j^{(i)} \log D_{\theta_{nc}}(F_\theta^{z_n}(x^{(i)}))_j}_{=:\mathcal{L}_{nCE}(\theta,\theta_{nc},\mathcal{T})}.$$

This loss is then optimized via the minimax-problem

$$\min_\theta \max_{\theta_{nc}} \mathcal{L}_{iCE}(\theta, \theta_{nc}, \mathcal{T}).$$

The underlying principles of the nuisance classification loss $\mathcal{L}_{nCE}$ can be understood using a variational lower bound on mutual information, which was given in (Barber and Agakov, 2003):

Lemma 6.15 (Variational lower bound on mutual information)**.** *Let* X, Y *be random variables (as in Lemma 6.13) with conditional density* $p(y|X = x)$. *Further, let* $q_\theta(y|X = x)$ *be a variational density depending on parameter* θ. *Then, the lower bound*

$$I(Y;X) \geq h(Y) + \mathbb{E}_{x\sim X}\mathbb{E}_{y\sim Y|X}\left[\log q_\theta(y|X=x)\right] =: I_\theta(Y;X)$$

holds with equality if and only if $p(y|X = x) = q_\theta(y|X = x)$.

Proof. Following the derivation in (Barber and Agakov, 2003) and including the information-theoretic properties from section 2.1.3, we have

$$\begin{aligned}
I(Y;X) &= H(Y) - H(Y|X) && (6.6)\\
&= H(Y) + \mathbb{E}_X\left[\sum_{j=1}^{C} p(y_j|X=x)\log p(y_j|X=x)\right] && (6.7)\\
&= H(Y) + \mathbb{E}_X\left[\sum_{j=1}^{C} p(y_j|X=x)\log\left(p(y_j|X=x)\frac{q_\theta(y_j|X=x)}{q_\theta(y_j|X=x)}\right)\right] && (6.8)\\
&= H(Y) + \mathbb{E}_X\left[\sum_{j=1}^{C} p(y_j|X=x)\log\left(q_\theta(y_j|X=x)\right)\right]\\
&\quad + \mathbb{E}_X\left[D_{KL}(p(y|X=x)\parallel q_\theta(y|X=x))\right] && (6.9)\\
&\geq H(Y) + \mathbb{E}_X\left[\sum_{j=1}^{C} p(y_j|X=x)\log\left(q_\theta(y_j|X=x)\right)\right].
\end{aligned}$$

In above derivation, (6.6) was given in Lemma 2.13 (i), (6.7) is the conditional entropy from Definition 2.12, (6.8) is due to the addition of a zero term and (6.9) applies the logarithm rules and the KL divergence (see Definition 2.9). Most importantly, the inequality (and the iff-statement in the Lemma) is due to the information inequality (Lemma 2.10). □

While this lower bound removes the need of having access to the unknown conditional density $p(y|X = x)$, estimating the expectation $\mathbb{E}_{Y|X}$ still requires sampling from $\mathcal{D}_{Y|X}$. This, however, is possible using the data pairs $(x^{(i)}, y^{(i)})$ from the training set $\mathcal{T}$. By leveraging this variational bound, we can state the effect of the nuisance classification loss $\mathcal{L}_{nCE}$ in the following theorem.

Note that we consider stochastic encoders to simplify the proof and discuss the extension to deterministic encoders (neural network F_θ that maps x to z) in Remark 6.17.

Theorem 6.16 (Effect of nuisance classifier for stochastic encoders)**.** *Let random variables X, Y, Z_s, Z_n and neural network F_θ be defined as before. Further, let F_θ^* denote a stochastic transform, where $z =: F_\theta^*(x) = F_\theta(x) + \varepsilon$ and $\varepsilon \sim \mathcal{N}(0, \sigma^2)$. Thus, the conditional density $p_\theta(z|X = x)$ is given by the stochastic encoder F_θ^*.*
Then, the minimization/maximization of the nuisance classification loss $\mathcal{L}_{nCE}$ (from Def. 6.14) has following effects:

(i) Minimization of lower bound on $I_\mathcal{D}(Y; Z_n)$:
Set $\theta_{nc}^* := \operatorname{argmax}_{\theta_{nc}} \mathcal{L}_{nCE}(\theta, \theta_{nc})$. Then

$$\theta^* = \operatorname*{argmin}_\theta \mathcal{L}_{nCE}(\theta, \theta_{nc}^*)$$

minimizes $I_{\theta_{nc}^*}(Y; Z_n)$, where $I_{\theta_{nc}^*}(Y; Z_n) \leq I_\mathcal{D}(Y; Z_n)$ as given in Lemma 6.15.

(ii) Maximization to tighten bound on $I_\mathcal{D}(Y; Z_n)$:
Set $\theta^* := \operatorname{argmin}_\theta \mathcal{L}_{nCE}(\theta, \theta_{nc})$. Then

$$\theta_{nc}^* = \operatorname*{argmax}_{\theta_{nc}} \mathcal{L}_{nCE}(\theta^*, \theta_{nc})$$

maximizes $I_{\theta_{nc}^*}(Y; Z_n)$, which is a lower bound since $I_{\theta_{nc}^*}(Y; Z_n) \leq I_\mathcal{D}(Y; Z_n)$.

Proof. We start by using the variational lower bound on MI from Lemma 6.15. Let the nuisance classifier $D_{\theta_{nc}}(Z_n)$ model the variational posterior $q_{\theta_{nc}}(y|Z_n = z_n)$. Then we have the lower bound

$$I(Y; Z_n) \geq h(Y) + \mathbb{E}_{Z_n}\mathbb{E}_{Y|Z_n}\left[\log D_{\theta_{nc}}(z_n)\right] =: I_{\theta_{nc}}(Y; Z_n). \tag{6.10}$$

Estimating this bound via Monte Carlo simulation requires sampling from the conditional density $p(y|Z_n = z_n)$, but we have only access to samples from $p(y|X = x)$ via the training set $\mathcal{T}$. Thus, we need to consider the stochastic encoder $p_\theta(z_s|X = x)$ given by the stochastic neural network F_θ^*. Hence, we have

$$\begin{aligned} p(y|Z_n = z_n)p(z_n) &= p(y, z_n) \\ &= \int_{\mathbb{R}^d} p(x, y, z_n)\, dx \\ &= \int_{\mathbb{R}^d} p(z_n|X = x, Y = y)p(y|X = x)p(x)\, dx \qquad (6.11) \\ &= \int_{\mathbb{R}^d} p(z_n|X = x)p(y|X = x)p(x)\, dx, \qquad (6.12) \end{aligned}$$

where we used in (6.11) the chain rule of probability. For (6.12), we used the conditional independence of Z_n and Y when conditioned on X, since Z_n is given by the stochastic transform $F_\theta^*(X)$.

Rewriting (6.12) using expectations and inserting into (6.10) thus yields

$$\mathbb{E}_{Z_n}\mathbb{E}_{Y|Z_n}\left[\log D_{\theta_{nc}}(z_n)\right] = \mathbb{E}_X\mathbb{E}_{Y|X}\mathbb{E}_{Z_n|X}\left[\log D_{\theta_{nc}}(z_n)\right].$$

Hence, the right hand side above is estimated by the nuisance classifier loss $\mathcal{L}_{nCE}$, if the limit for $\sigma \to 0$ in the stochastic encoding F_θ^* converges to the deterministic encoding F_θ (see the subsequent remark). □

Remark 6.17 (Stochastic vs. deterministic encoding)**.** *Theorem 6.16 assumed a stochastic neural network F_θ^* to allow a reasoning via densities. However, in the definition of the loss (Def. 6.14) we considered only deterministic neural networks F_θ so far. To keep the focus on an understanding of the main principles of the proposed loss, we only sketch the necessary steps to lift the statement to the deterministic case: First, consider*

$$z =: F_\theta^*(x) = F_\theta(x) + \varepsilon, \quad \mathcal{N}(0, \sigma^2)$$

in the limit when variance $\sigma \to 0$, which converges to F_θ in Lebesgue spaces (when fixing the normalization constant $\frac{1}{\sqrt{2\pi\sigma^2}}$). Then insert this limit outside the integral in (6.12). This limit can be moved inside the integral, if the theorem of dominated convergence holds. This is possible, if $f(z_n) = \log D_{\theta_{nc}}(z_n)$ is continuous and compactly supported, which holds by assuming bounded inputs x to the (continuous) network F_θ.

By using these results, we can now state the main result under the assumed distribution shift and successful minimization:

Theorem 6.18 (Information $I_{\mathcal{D}_{Adv}}(Y; Z_s)$ maximal after distribution shift)**.** *Let $\mathcal{D}_{Adv}$ denote the adversarial distribution and $\mathcal{D}$ the training distribution. Assume $I_{\mathcal{D}}(Y; Z_n) = 0$ by minimizing $\mathcal{L}_{iCE}$ and assume that the distribution shift satisfies $I_{\mathcal{D}_{Adv}}(Z_n; Y) \leq I_{\mathcal{D}}(Z_n; Y)$ and $I_{\mathcal{D}_{Adv}}(Y; Z_s|Z_n) \leq I_{\mathcal{D}_{Adv}}(Y; Z_s)$ as previously. Then, it holds*

$$I_{\mathcal{D}_{Adv}}(Y; Z_s) = I_{\mathcal{D}}(Y; X).$$

Proof. By information invariance under reparametrization (Lemma 6.13) and the chain rule of mutual information (Lemma 2.13), we have

$$\begin{aligned} I_{\mathcal{D}_{Adv}}(Y; X) &= I_{\mathcal{D}_{Adv}}(Y; Z_s, Z_n) \\ &= I_{\mathcal{D}_{Adv}}(Y; Z_n) + I_{\mathcal{D}_{Adv}}(Y; Z_s|Z_n) \\ &\leq I_{\mathcal{D}_{Adv}}(Y; Z_s). \end{aligned}$$

As $Z_s = F_\theta(X)_{1,\dots,C}$ is given by a deterministic transform F_θ, we further have the inequality $I_{\mathcal{D}_{Adv}}(Y; X) \geq I_{\mathcal{D}_{Adv}}(Y; Z_s)$ via the data processing inequality (Lemma 2.13). Hence, the claimed equality must hold. □

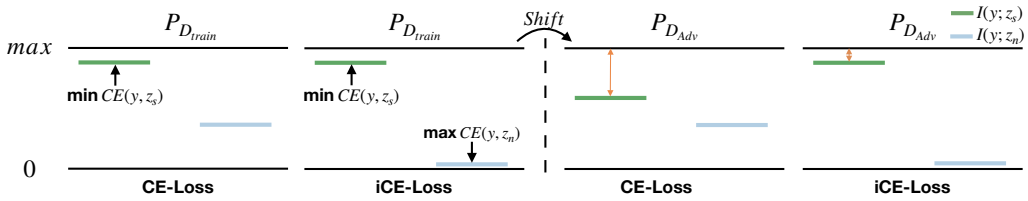


Figure 6.4: Left: Mutual information under distribution $\mathcal{D}_{train}$, **Right:** Effect of distributional shift to $\mathcal{D}_{Adv}$. Each case under training with cross-entropy (CE) and independence cross-entropy (iCE). Under distribution $\mathcal{D}$, the iCE-loss minimizes $I(Y; Z_n)$ (Theorem 6.16), but has no effect as the CE-loss already maximizes $I(Y; Z_s)$. However under the shift to $\mathcal{D}_{Adv}$, the information $I(Y; Z_s)$ decreases when training only under the CE-loss (orange arrow), while the iCE-loss induces $I(Y; Z_n) = 0$ and thus leaves $I(Y; Z_s)$ unchanged (Theorem 6.18).

Thus, incorporating the nuisance classifier allows for the discussed indirect increase of $I_{\mathcal{D}_{Adv}}(Y; Z_s)$ under the assumptions of the specified adversarial distribution shift. A visualization of the studied effects is given in Figure 6.4.

To aid stability and further encourage factorization of Z_s and Z_n in practice, we add a maximum likelihood term to our independence cross-entropy objective as

$$\mathcal{L}(\theta, \theta_{nc}, \mathcal{T}) := \mathcal{L}_{iCE}(\theta, \theta_{nc}) - \underbrace{\sum_{i=1}^{N} \sum_{k=1}^{d-C} \log \left(p_k(F_\theta^{z_n}(x^{(i)})_k) |\det(J_{F_\theta}(^{(i)}))| \right)}_{=:\mathcal{L}_{MLE_n}(\theta), \mathcal{T}}, \tag{6.13}$$

where $\det(J_{F_\theta}(x))$ denotes the determinant of the Jacobian of F_θ at x and $p_k \sim \mathcal{N}(\beta_k, \gamma_k)$ with β_k, γ_k learned parameter. See chapter 5 for details on using maximum likelihood via the change of variables.

To better understand this term, the following Lemma describes that optimizing $\mathcal{L}_{MLE_n}$ on the nuisance variables together with $\mathcal{L}_{sCE}$ amounts to maximum likelihood estimation (MLE) under a factorial prior.

Lemma 6.19 (Effect of MLE-term)**.** *Let random variables Z_s, Z_n and training set $\mathcal{T}$ be defined as before. Then, the MLE-term in (6.13) together with a cross-entropy loss on the semantics z_s yields a parameter θ^* for the encoding F_θ via*

$$\theta^* = \operatorname*{argmin}_\theta \mathcal{L}_{sCE}(\theta, \mathcal{T}) + \mathcal{L}_{MLE_n}(\theta, \mathcal{T}),$$

which minimizes the mutual information $I(Z_s; Z_n)$.

Proof. Let $\tilde{z}_s := \text{softmax}(z_s)$. Then minimizing the loss terms $\mathcal{L}_{sCE}$ and $\mathcal{L}_{MLE_n}$ is a maximum likelihood estimation under the factorial prior

$$p(\tilde{z}_s, z_n) = p(\tilde{z}_s)p(z_n) = Cat((\tilde{z}_s)_1, \ldots, (\tilde{z}_s)_C) \prod_{k=1}^{d-C} p_k(z_n)_k,$$

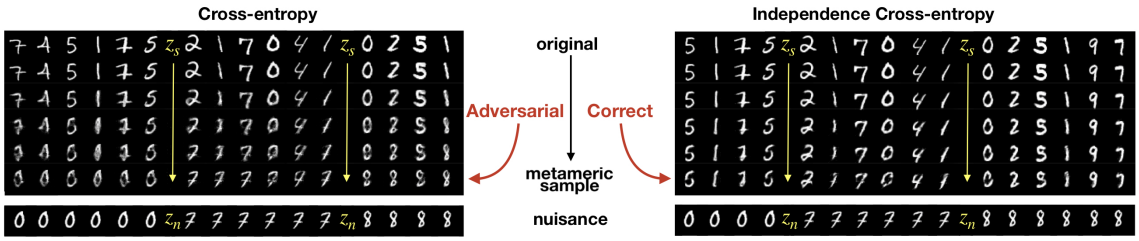


Figure 6.5: From (Jacobsen et al., 2019b): Samples $\tilde{x} = F^{-1}(z_s, \tilde{z}_n)$ with logit activations z_s taken from original image and $\tilde{z}_n$ obtained by linearly interpolating from the original nuisance z_n (first row) to the nuisance of a target example z_n^* (last row upper block). The used target example is shown at the bottom. When training with cross-entropy, virtually any image can be turned into any class without changing the logits z_s, illustrating strong vulnerability to invariance-based adversaries. Yet, training with independence cross-entropy solves the problem and interpolations between nuisances z_n and z_n^* preserve the semantic content of the image.

where Cat is a categorical distribution. Further, note that softmax is shift-invariant, i.e. $\text{softmax}(x + c) = \text{softmax}(x)$. Thus, the factorial prior on $\tilde{Z}_s$ (which is fitted via cross-entropy on the semantics) and Z_n yields independence between logits Z_s and Z_n up to a constant c. Finally note, that the *log* term and summation in $\mathcal{L}_{MLE_n}$ and $\mathcal{L}_{CE}$ is a reformulation for the sake of computational simplicity, but does not change its minimizer as the logarithm is strictly monotone. □

Remark 6.20 (Mutual information bounded from above)**.** *Since our goal is to simultaneously maximize $I(Y; Z_s)$ and minimize $I(Y; Z_n)$ (even after a distribution shift), we need to ensure that this objective is well-defined. In general, MI can be unbounded from above for continuous random variables. However, due to the data processing inequality (Lemma 2.13), we have*

$$I(Y; Z_n) = I(Z; F_\theta(X)) \leq I(Y; X).$$

Hence, we have a fixed upper bound given by our data $\mathcal{T} = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$. In comparison to work by (Belghazi et al., 2018), there is no need for gradient clipping in our setting. Furthermore, switching to other divergences like the bounded Jensen-Shannon divergence as in (Hjelm et al., 2019) is not necessary, but might be an interesting avenue for future work.

Just as in GANs (Goodfellow et al., 2014), our analysis relies on having a tight bound, which is met under a perfect nuisance classifier and convergence of the MLE-term. Since this is hard to achieve in practice, it is important to analyze the success of the objective after training. For a numerical analysis of the application of independence cross-entropy (Definition 6.14) under a simple distribution shift, we refer to (Jacobsen et al., 2019b***). Furthermore, we add a qualitative result from (Jacobsen et al., 2019b***) in Figure 6.5.

6.4.3 Related Work on Information Theory in Deep Learning

Information theory, and especially the study of mutual information, has gained recent interest in deep learning due to the information bottleneck theory (Tishby and Zaslavsky, 2015; Shwartz-Ziv and Tishby, 2017; Alemi et al., 2017) and usage in generative modelling (Chen et al., 2016; Hjelm et al., 2019). As a consequence, the estimation of mutual information has attracted growing attention, see e.g. (Barber and Agakov, 2003; Alemi et al., 2018; Achille and Soatto, 2018; Belghazi et al., 2018).

The concept of group-wise independence between latent variables goes back to classical independent subspace analysis (Hyvärinen and Hoyer, 2000) and received attention in learning unbiased representations, e.g. see the Fair Variational Autoencoder (Louizos et al., 2015). Furthermore, extending cross-entropy losses via entropy terms (Pereyra et al., 2017) or minimizing predictability of variables (Schmidhuber, 1991) has been introduced for other applications. Our proposed loss also shows similarity to the GAN loss (Goodfellow et al., 2014). However, in our case there is no notion of real or fake samples. Despite those differences, exploring similarities in the underlying minimax-optimization problem is a promising avenue for future work.

6.5 Conclusion and Future Work

To conclude, we introduced excessive invariance as a reverse viewpoint on adversarial examples. We showed, that the invariance-based adversarial examples is a complementary failure mode to the commonly studied case of perturbation-based adversarial examples for discriminative models. A first study of their relationship was done using the synthetic example of *adversarial spheres* by (Gilmer et al., 2018b). Further studies in (Jacobsen et al., 2019a***) show how models trained to be ℓ_p-robust are often even more vulnerable to invariance-based attacks. Thus, the presented reverse viewpoint should be employed in addition to the common robustness studies in order to obtain a holistic understanding of the worst-case behavior of a model under a distribution shift.

In line of the main focus of this work, invertible neural networks played a crucial role to uncover the invariance-based model failures. Even more importantly, these architectures may allow to control the invariant directions via the concept of a splitting into semantic and nuisance features as advocated in this chapter. In summary, the ability to invert the feature extraction process, paired with guarantees about information preservation, may be crucial towards obtaining reliable models even under worst-case scenarios.

Despite these promising first steps, many open questions remain:

- How to quantify invariance-based vulnerability in order to enable a comparison between models?
- How to improve the proposed independence cross-entropy loss? In particular, studying upper bounds on $I(Y; Z_n)$ could allow to drop the assumption of a perfect nuisance classifier, which is hard to achieve in practice.

- How to extend the study of invariance to input perturbations that allow small changes in the output?

Furthermore, an exciting future direction would be the usage of i-ResNets, because they are designed to be forward and inverse stable (see Chapter 4 and in particular Lemma 4.6).

Chapter 7

Domain Knowledge: Architectures for Imaging Mass Spectrometry

In this chapter, we focus on the second pillar of this thesis: using domain knowledge to design principled neural network architectures. First, we briefly discuss common approaches to incorporate knowledge about the data structure into architectures. This overview is followed by a comparison of the structures underlying images and mass spectra, which leads us to a customized convolutional network architecture. Finally, we study the behavior of these networks on two challenging tumor classification tasks.

This chapter is mainly based on following article:

> **Jens Behrmann**, Christian Etmann, Tobias Boskamp, Rita Casadonte, Jörg Kriegsmann, Peter Maass: *Deep learning for tumor classification in imaging mass spectrometry*, 2018, Bioinformatics, 34(7), pages 1215 - 1223

7.1 Domain Knowledge in Deep Learning

Fully-connected neural networks (MLPs) are universal function approximators (Cybenko, 1989) and thus allow to solve any function approximation task, provided the network is sufficiently wide or deep. Yet, network architectures for practical learning tasks often differ drastically from vanilla fully-connected designs. The most prominent example are convolutional neural networks (see Definition 2.23), which formed the basis of the architectures studied in the previous chapters.

In mathematical terms, the selection of the hypothesis space $\mathcal{F}$ for empiricial risk minimization

$$F^* \in \operatorname*{argmin}_{F \in \mathcal{F}} L_{\mathcal{D}}(F),$$

induces a certain bias (besides other choices like the regularization procedure or the optimizer). Ideally, this hypothesis space $\mathcal{F}$, and thus the design of the network's architecture, is guided by a-priori knowledge about the data and/or the learning task. When applied

meaningfully, this domain knowledge can significantly reduce the sample complexity of the learning process and improve generalization. In this section, we shortly discuss some approaches to encode domain knowledge into architectures. For a more detailed overview we refer to (Goodfellow et al., 2016) as well as to a more recent discussion by (Battaglia et al., 2018).

Most prominently, convolutional neural networks (CNNs) are equivariant to spatial translations (Kondor and Trivedi, 2018). They induce a bias via locality, since the filters used for the convolution operation have a limited spatial extend. Recurrent neural networks (RNNs), on the other hand, are equivariant to time translations. Typically, they are used for sequential data and for tasks where a hidden state of some dynamical process needs to be modelled. While many tasks involve sequential data or images, there is a multitude of more specialized architectures. For example, when considering relational data like graphs, graph neural networks (GNNs) have become a predominant architecture, see e.g. (Battaglia et al., 2018).

In this chapter, we are interested in extending the design space of neural network architectures to incorporate knowledge about mass spectra arising in tumor classification from Imaging Mass Spectrometry (IMS). Since this data structure shares similarity to images, concepts from CNN-architectures can be transferred. For example, both images and IMS spectra are defined over grids and exhibit local structures. However, many differences appear at closer inspection, which is why we introduce a modified CNN-architecture for mass spectra.

This chapter is structured as follows: We first give a brief introduction to tumor typing with IMS data to better understand the broader field of application. Then, we discuss approaches to incorporate domain knowledge into architectures for IMS data processing and test their performance on two real-world tasks.

7.2 Tumor Typing with Imaging Mass Spectrometry

Imaging Mass spectrometry (IMS) is a label-free technique for spatially resolved molecular analysis of small to large molecules. Given a thin tissue section, mass spectra are recorded at multiple spatial positions on the tissue, yielding an image where each spot represents a mass spectrum. These spectra relate the molecular masses to their relative molecular abundances and thus offer insights into the chemical composition of a region within the tissue, see e.g. (Stoeckli et al., 2001). In this chapter, we consider matrix-assisted laser desorption/ionization imaging mass spectrometry (MALDI IMS) (Caprioli et al., 1997) for our study. However, the analysis and methods should also be applicable to other IMS modalities like SIMS (Benninghoven and Loebach, 1971).

In MALDI IMS, molecules of interest co-crystallize with an organic matrix compound that assists in the desorption and ionization of the molecules on irradiation with a laser beam. This sample preparation of applying a matrix in the last step is also applicable to formalin-fixed paraffin-embedded (FFPE) tissue, a common tissue storage solution in pathology.

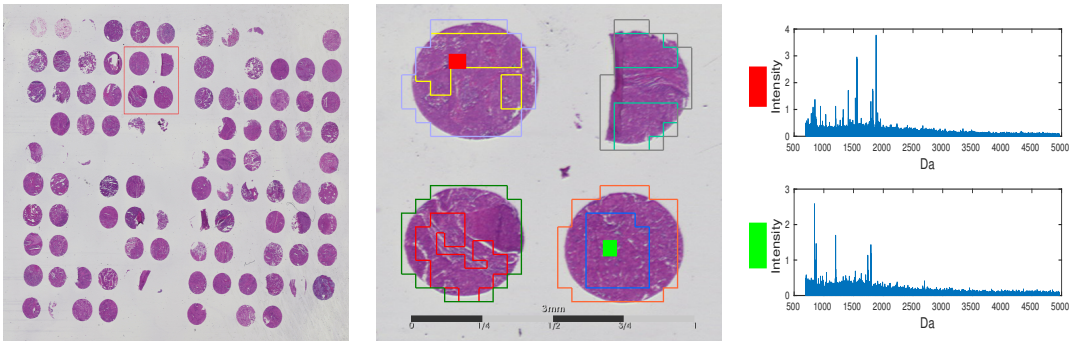


Figure 7.1: Overview on structural hierarchies of the IMS data, from TMA to tissue core to a single spectrum. Left, a HE-image of a TMA is shown, which is measured in a single IMS measurement. The red box (left) marks the four tissue cores shown in the middle. These tissue cores have two annotations, the outer region marks the measurement region for the laser, while the inner region are the Region-of-Interest (ROI) annotated by a pathologist. Furthermore, the red and green dots correspond to a spot of the imaging data. Each of these spots correspond to a mass spectrum shown in the right figure.

Hence, MALDI IMS has a high potential for many pathological applications, as discussed by (Aichler and Walch, 2015) or (Kriegsmann et al., 2015). One of the main advantages of MALDI IMS is that it allows high-throughput analysis of several tumor cores from different patients by arranging them in a single tissue microarray (TMA) (Casadonte et al., 2017). See Figure 7.1 (left) for an image of a TMA. Thus, within a single run of the mass spectrometer a large cohort of potentially cancerous tissue can be analyzed in order to extract biochemical information in a spatial manner. This biochemical information may then be used for 1) the determination of the cancer subtypes or 2) the identification of the origin of the primary tumor in patients with metastatic disease. In both cases, an accurate typing of a tumor is crucial for successful treatment of patients. For related studies see e.g. (Casadonte et al., 2014).

While current MALDI IMS instruments are able to acquire molecular information at a high spatial resolution ($< 20\mu m$ center-to-center spacing between each ablated laser spot) at short measurement times ($>$ 20 pixels/s), advanced bioinformatic tools may help to extract knowledge in a robust manner. This has been recognized as a challenging task in bioinformatics as it involves analyzing spatially distributed high-dimensional spectra (Alexandrov, 2012). Especially in tumor classification, a robust feature extraction procedure is required in order to integrate this workflow into a reliable routine.

In our application, we focus on processing each spectrum separately, because the mass spectra (see Figure 7.1 (right)) are measured from small tissue core regions (diameter of 0.6 - 1.0 mm) with little varying structure within a single core (see Figure 7.1 (middle)). This core represents only a small portion of the original tumor biopsy ($\sim$ 20x20 mm) from which it was extracted. Thus, the morphological heterogeneity of such tissue cores is strongly reduced when compared with the original tumor sample. In this study, $\geq 80\%$ of the tissue analyzed was composed of tumor cells only.

Beside classical approaches like the ones studied by (Boskamp et al., 2017), where a sep-

arate feature extraction and classification stage was used, end-to-end learning where features and classification are learned in one step offers a promising alternative. Due to the astonishing success of analyzing high-dimensional data with deep learning, we aim towards solving the challenges when applying deep networks to tumor classification based on MALDI IMS. One of the main challenges is to find a suitable architecture, which we discuss in-depth here.

7.3 Architectures for IMS Spectra

7.3.1 Comparing IMS spectra to Images

After preprocessing the data (see (Behrmann et al., 2018b***) for details), each spectrum measured in a tissue spot by IMS is handled separately. For an illustration of a spectrum, see Figure 7.1 (right). We denote a spectrum as a data point $x \in \mathbb{R}^d$, where d denotes the number of m/z-bins. For example, $d = 27286$ in one of the conducted experiments. These spectra can thus be viewed as structured data points on a pre-defined grid (the m/z-bins). In this regard spectra are similar to images, where the grid is given by the pixels. Thus, mass spectra can be viewed as one-dimensional images. However, one major difference to images is that the underlying grid of these spectra is not necessarily equidistant. Still, as a first step a reasonable assumption is that methods commonly applied to image classification are also suitable for mass spectra.

Two of the main driving forces/ ideas behind the design of deep CNNs for images are:

- The need to handle high-dimensional data, which is why the idea of convolutional transforms with their few parameters of the filter kernel plays a key role.
- The assumption of several levels of abstraction in the data, which motivates the layered architecture of CNNs.

While the first layers may be able to extract edges in images, the goal of higher layers is to extract more complex shapes like curves or even entire structures like faces of humans. However, this common interpretation of the functionality of CNNs on images is often too simple. As (Geirhos et al., 2019) show empirically on ImageNet (Deng et al., 2009), CNNs can be strongly biased towards texture or shape, depending on the learning task and the available data. Thus, following analogies of CNNs on images extracting shapes and our modified CNN on IMS spectra extracting similar hierarchical characteristics need to be taken with caution.

Assuming CNNs indeed follow such hierarchical extraction steps, we now discuss the analogies of images and IMS spectra. First, IMS spectra are also high-dimensional data lying on a grid, where the area between grid points are called m/z-bins. Hence, CNNs can offer the same remedy for working in a high-dimensional domain by grouping neighboring m/z-bins together via convolutions. As the spectra are transformed through the network, this grouping of neighboring m/z-bins grows. By applying subsampling to these groups via

Table 7.1: Some analogies of image features and IMS spectra features on several hierarchical levels.

	Images	IMS spectra
low-level	edges	peaks
mid-level	contours	isotope patterns
high-level	objects	protein patterns

pooling or strided convolution (see basics on CNNs in section 2.2.2), the obtained feature maps become lower dimensional.

Altogether, similar ideas are transferable to IMS data in order to extract features from high-dimensional data. However, it is important to discuss the underlying assumptions. The main assumption of a convolutional transform is that neighboring m/z-bins are correlated, which can be exploited by a filter kernel. This is certainly plausible when considering raw data from a time-of-flight (TOF) mass spectrometer, where a peak is spread over several m/z-bins. On the other hand, deep CNNs perform the mentioned grouping also on transformed data in order to extract higher-level features. Here, its impact onto spectral data is less obvious. While peaks may be the counterpart of edges in images, mid-level features may be represented by isotope patterns or even adduct patterns of the same peptide. See for example (Shank et al., 2015), where this assumption was used to extract patterns from the data. On the highest level, tryptic-digested proteins may contribute to several measured peptides, resulting in patterns across the entire mass range. See for example (Boskamp et al., 2017), where the idea is to extract these characteristic spectral patterns. The key difference between these patterns is their position on the mass grid. While isotope patterns can be considered local as they are formed in a small connected mass interval, protein patterns are non-local as the digested peptides may be spread irregularly over the entire measurement range. See Table 7.1 for a summary of the discussed analogies.

7.3.2 Constructing an IMS Architecture

This assumption of the composition of the spectral data leads us to an adapted architecture design for IMS spectra, which we name *IsotopeNet* (see Figure 7.2 for a visualization of its working principle). In particular, we encode our domain knowledge as follows:

- We estimate the number of m/z-bins of large measurable isotope patterns of peptides based on a model of an average amino acid.
- Using this estimate, we restrict the local grouping roughly to the size of large isotope patterns, such that one variable is able to encode such a local feature.
- The downsampling rate by *strided convolutions* is determined by an average distance between peaks.

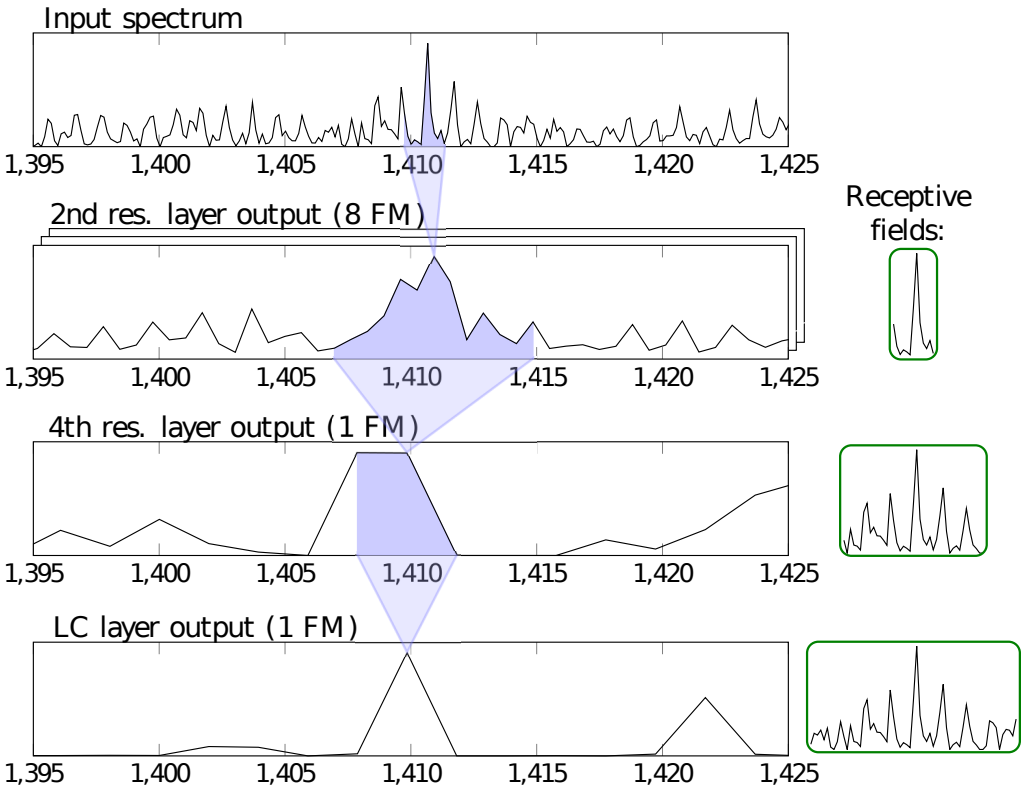


Figure 7.2: From (Behrmann et al., 2018b***): Overview over the working principle of *IsotopeNet*. The first row shows a section of a recorded mass spectrum of a squamous cell carcinoma. Several residual layers of depths 2 extract interesting features of small portions of the previous layers' outputs. Due to their consecutive (and partially strided) convolutions, an increasingly large portion of the input spectrum influences each spot in the deeper layers of the neural network. This is signified by the *receptive fields* on the right hand side, which reach the size of a whole isotopic pattern after the 4th residual layer (before the locally connected layer).

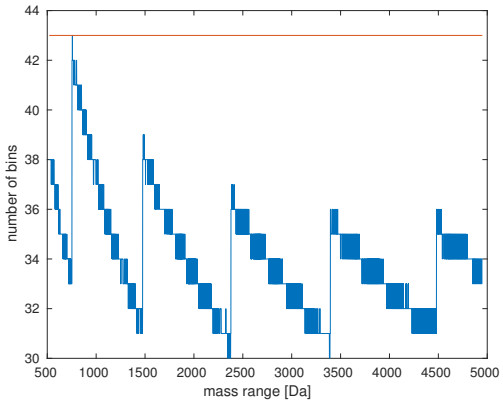


Figure 7.3: Number of bins per isotope pattern over the given mass range, shown in blue. Due to the decreasing m/z-resolution the number of bins decrease despite a growing number of peaks. As a comparison, the number of bins of the receptive field of *IsotopeNet* is drawn in red. Note, that the jumps are due to threshold t_{iso}, which rejects very unlikely isotopes.

Above design choices are crucial, but only partially describe the network architecture. Additionally, the filter sizes and number of filters needs to be chosen. Thus, we discuss these aspects and the non-equidistant binning in more detail in the following paragraphs:

Estimation of Isotope Size: The size of large isotope patterns of peptides served as a guideline to design the convolutional part of *IsotopeNet*. More specifically, the receptive fields after the convolutional transforms should be able to cover even the largest observable isotope patterns, because they are considered to be the mid-level features the deep network needs to extract. Hence, an estimation of the size of isotope patterns in terms of m/z-bins is required to design appropriate receptive fields. Note that the m/z-resolution decreases over the mass range (see discussion on non-equidistant binning after the next paragraph). A simple model for isotope patterns of peptides has been proposed by (Senko et al., 1995), where an average amino acid named *Averagine* serves as a basis for modeling peptides at a given mass. This model takes into account the proportion of each amino acid in homo sapiens and estimates the number of carbon atoms. Based on this estimate, the isotope distribution is modeled by a Bernoulli distribution, using the stable isotope rates of carbon. Furthermore, we set a threshold $t_{iso} = 0.005$ by visual examination to cut off very unlikely and, compared to the most abundant peak, insignificant isotopes. A way to adjust this threshold to the data at hand is by 1) simulating isotope patterns of molecules with high intensity and 2) visually comparing peak height to the noise level of the data. Yet, a thorough study of the influence of this parameter is beyond the scope of this study as it only serves as a guideline for the architecture design. Finally, we convolve each peak with a Gaussian filter.
This model allows for a computation of the number of m/z-bins for the estimated isotope patterns, as shown in Figure 7.3 (in blue). Next, we go back to the original task of designing the convolutional transforms and try to answer following question: how large should we

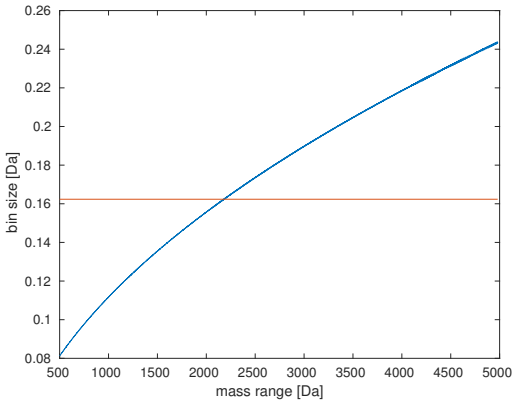


Figure 7.4: Non-equidistant binning of TOF data (examplified for task ADSQ). In blue, the development of m/z-bin size over the mass range is shown. In red, an average m/z-resolution is drawn, which was used as a guideline for the stride.

set the receptive field? Our choice is depicted in Figure 7.3 as the straight line (43 bins over the entire mass range). Hence, the used receptive field is larger than the estimated isotope patterns, which enables the network to encode these local features. A description of the computation of receptive fields and further discussions can be found in (Luo et al., 2016).

Filter sizes and downsampling: In addition to the desired size of the receptive field as described above, the exact scheme of filtering and downsampling needs to be specified as well. First of all, small filters of size 3 are used throughout *IsotopeNet* in order to capture small differences between neighboring bins.
Moreover, the first application of strided convolutions employs the convolutional kernel with a step size of 5 (see Table 7.2) in order to decrease the spatial dimension of the feature maps. This step size is motivated by the distance between the individual peaks of isotope patterns. Due to the peptide nature of the measurements, the peaks occur with a distance of roughly 1 Da. As the average bin size is 0.163 Da (see Figure 7.4), on average 6.14 m/z-bins cover one such interval of width 1 Da. Thus, with a stride of 6, each variable in the following layer encodes approximately one peak on average. In order to prevent too much information loss in lower-resolved parts of the m/z-axis, we picked a slightly lower stride of 5 instead.
The second downsampling operation is realized as a convolution of stride 3. Here, the stride was chosen to reach the desired receptive field size.

Non-equidistant binning: The m/z-binning is usually not equidistant over the mass range for TOF spectra, see Figure 7.4 for an example. Yet, the convolutional kernels are applied bin-wise with the same width, which results in a broader filter for large masses due to the binning. An alternative would be to interpolate the data in order to ensure an equidistant binning. Yet, this would have the disadvantage of adding another preprocessing step to the computational pipeline and could introducing artifacts to the data.
In convolutional networks, however, several feature maps are used per layer to enable the

Table 7.2: Architecture of IsotopeNet. Total number of trainable parameters is 13935.

Layer	depth	kernel size	stride	# feature maps
Input layer	-	-	-	1
Residual layer	2	3	1	8
Residual layer	2	3	5	8
Residual layer	2	3	1	8
Residual layer	2	3	3	1
ReLU nonlinearity	-	-	-	1
Locally connected layer	-	3	1	1
Fully connected (softmax)	-	-	-	1

network to extract a variety of features. This idea is also transferable to the above case of non-equidistant binning since different filters may be able to specialize on different mass ranges in the course of the training (if necessary). Furthermore, the application of several filters enables the network to account for violations of the discussed assumptions on the distance between peaks or isotope pattern sizes. Due to the binning or mis-alignment of peaks, the width and shape may undergo different changes which can be captured using several convolutional kernels.

Processing after convolutional transforms: After the convolutional transforms, a locally-connected layer (similar to convolutional layer that uses local information, except that weights are not shared) is used to process also those local input features, which are encoded in the two neighboring variables. Furthermore, this operation enables the network to handle each local region differently due to *unshared weights*. Hence, a focus only on important peptides for the given classification task is possible. For an example, see the squamous cell carcinoma tumor spectrum in Figure 7.2. Moreover, we compare the proposed architecture to a deep Residual Network (He et al., 2016) with the architecture given in Table A.1. The architecture of *IsotopeNet* is summarized in Table 7.2.

However, we emphasized that the adapted architecture is geared towards TOF data with a focus on peptide measurements and is only a step towards a deep network design based on domain knowledge. Especially for other measurement objectives like the extraction of metabolites or non-TOF data, different design choices may be more suitable. Yet, we believe that previous considerations can serve as a starting point for those explorations.

7.4 Results

7.4.1 Datasets and Evaluation

In this study we test the proposed CNNs on two challenging real-world datasets consisting of 12 MALDI IMS measurements of a large cohort of tumor tissue cores. In this comparison, we use the same setting as in the previous study by (Boskamp et al., 2017), in order to

establish a solid comparison of the proposed methods to other common approaches. For a detailed description of the setup, we refer to (Behrmann et al., 2018b***).

We consider two different classification tasks:

- Tumor subtyping of adenocarcinoma vs. squamos cell carcinoma (called task ADSQ)
- Primary tumor typing of lung vs. pancreas tumor (called task LP).

In the ADSQ dataset there are annotated subregions called Regions-of-Interest (ROI), see Figure 7.1 (middle). These regions were marked by a pathologist as relevant subregions within the tissue core for subtyping the tumor. In order to perform classification solely on those subregions, only those spectra within each ROI are used for task ADSQ, resulting in a reduced number of spectra of 4672. Note that previous studies using whole tissue cores yielded inferior results. On the other hand, we used the entire tissue core for task LP, which also include spots with non-tumor cells, resulting in a total of 27475 spectra.

For evaluation of performance we performed randomized 4-fold cross-validation on TMA level, see (Behrmann et al., 2018b***) for more details. As a single performance measure we report the balanced accuracy

$$\text{balAcc} = \frac{1}{2}\left(\frac{TP}{P} + \frac{TN}{N}\right),$$

where TP/P denotes *true positive/ positive* ($j = 1$) and TN/N denotes *true negative/ negative* ($j = 2$). This measure is, unlike the accuracy, not biased by the relative class proportions in the data. Furthermore, we report the median balanced accuracy of the four cross-validation runs.

As a simple baseline method we use a feature extraction based on discriminative m/z-values, as it was done in (Boskamp et al., 2017). This method aims at identifying individual m/z-values by computing the Mann-Whitney-Wilcoxon statistic for each m/z-value separately (called *ROC* method here). After computing this statistic, we perform a selection of discriminative m/z-values by taking those K features with the highest test statistic in a range from $K = 5$ to $K = 100$. Subsequent to feature extraction by discriminative m/z-values, a linear discriminant analysis (LDA) classifier is used, a standard algorithm for creating classification models (Hastie et al., 2001).

7.4.2 Model Comparison

In this comparison we used a ResNet (see Table A.1 in Appendix A.3) and the specialized architecture for IMS (named *IsotopeNet*), which we discussed previously in section 7.3. The training parameters are given in Table A.2 in Appendix A.3.

Figure 7.5 (left) reports the results on both tasks ADSQ and LP, where the method *ROC/LDA* refers to the baseline model. For ROC/LDA we report the worst and the best performance over the number of features K from $K = 5$ to $K = 100$ in order to get an impression of the variance. For task ADSQ *ROC/LDA* reaches a balanced accuracy

Method	Task ADSQ		Task LP	
	Bal. Acc. (Spot)	Bal. Acc. (Core)	Bal. Acc. (Spot)	Bal. Acc. (Core)
ROC/LDA	0.758	0.788	0.794	0.876
	0.787	0.860	0.840	0.918
ResidualNet	0.824	0.870	0.921	0.973
	±0.016	±0.008	±0.014	±0.13
IsotopeNet	0.845	0.885	0.962	1.000
	±0.007	±0.020	±0.009	±0.002

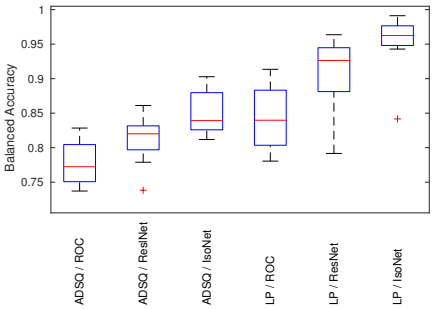


Figure 7.5: Left: Comparison table of 4-fold cross-validation on both tasks. For ROC/LDA the worst and best results over 5 to 100 features are reported in the table. For ResidualNet and IsotopeNet, the table shows the median obtained from four runs with identical parameter settings, together with the interquartile range to estimate the spread. The core level results are obtained by taking the majority of the predicted label. Right: Boxplot of the balanced accuracy from each method over the four cross-validation folds, reported on spot level for both tasks.

of 78.7% on spot level and 82.7% by aggregation on cores for task ADSQ, while the performance for task LP is about 5% higher.

Figure 7.5 (left) further shows how the *ResidualNet* compares to the domain adapted architecture *IsotopeNet*. Due to the stochasticity of the training process through stochastic gradient descent (see Definition 2.26), random initialization and regularization by dropout (Srivastava et al., 2014), both methods were run four times using the same parameter setting. From those four runs the median balanced accuracy is reported to get a robust idea of the average performance. Note that we did not use ensembles, which could improve performance even further. In addition, the interquartile range is stated below the median to estimate the variance induced by the mentioned stochasticity. Overall, the domain adapted architecture *IsotopeNet* performs better than both *ResidualNet* and *ROC/LDA*. For example, the spot level balanced accuracy for task ADSQ is 84.5%.

Whereas the previous discussion considered the variance of several runs over the entire dataset, Figure 7.5 (right) visualizes the variance over the cross-validation folds on spot level. To obtain this box plot, we computed the balanced accuracy of the four identical runs for each fold. For *ROC/LDA*, however, only the best model over the number of features was selected. As visible from the red median line, *IsotopeNet* outperforms the other methods on both tasks.

7.5 Conclusion and Future Work

This chapter discusses first steps towards the integration of knowledge about mass spectra into convolutional architectures. In particular, we studied tumor classification tasks based on IMS data from time-of-flight (TOF) devices. Most importantly, the size of the receptive

fields was guided by the size of isotope patterns. To separate neighboring isotope patterns, locally-connected layers followed the processing via convolutions. Furthermore, the filter size in the first layer was adapted to the common peak width of TOF mass spectra. Finally, the architecture was compared to a vanilla ResNet and to a simple baseline on two challenging tasks. As the evaluation via cross-validation showed, the customized architecture outperformed the baseline, which again emphasizes the need to induce a meaningful bias into the learning problem.

Next, we discuss further topics and future work:

Interpretation of models: While the focus of this chapter was on the design of customized architectures for IMS data, interpreting the learned model is crucial for sensitive applications like tumor classification. Unlike image features like shapes or texture, features of IMS data (peaks, isotope patterns) have a fixed position (when neglecting small mass mis-alignments). This property simplifies a global model interpretation, as sensitivity maps can be averaged over multiple samples in a meaningful manner. For details on the interpretation of the models presented in this chapter, we refer to (Behrmann et al., 2018b***).

Incorporating additional structure: The presented *IsotopeNet* only partially incorporates our domain knowledge. As visible in Figure 7.4, IMS spectra from TOF devices lie on a non-equidistant grid. However, the filters of the convolution operations have fixed width and are thus applied over different ranges depending on the m/z-value. Thus, resampling the data to an equidistant grid might better suit the application of a vanilla convolution. According to the topic of this thesis, this knowledge could also be incorporated into the architecture by:

(i) Modeling the filters in the continuous domain with restricted physical width (in terms of m/z).

(ii) Discretizing the filters for a usage in a modified convolutional layer.

In summary, deep neural networks may significantly impact the common approaches to classifying imaging mass spectrometry data since this data is highly structured and high-dimensional. Yet, the field is still in an early stage where only limited data is available and many data characteristics strongly vary across devices and laboratories. Thus, incorporating the data structure into the network architecture is a crucial step, besides gaining a better understanding of technical and biological variations.

Chapter 8

Summary and Conclusion

This thesis studied how to guide the modeling of neural network architectures by requiring invertibility or incorporating domain knowledge. Starting with an analysis of the invertibility of standard architectures, we arrived at the conclusion that additional structure is required to obtain guaranteed invertibility. The stability analysis then resulted in an invertible network architecture without structural constraints. Afterwards, two applications of invertible models were studied: generative modeling and understanding learned representations. Finally, the second design principle focused on incorporating domain knowledge into architectures for mass spectra arising from imaging mass spectrometry measurements.

As each chapter had a dedicated outlook focusing on next steps to improve the considered aspects, we broaden our view in this outlook. Most notably, this thesis studied neural network architectures and loss functions which required invertible models. However, the learning process involves (at least) one additional aspect: the learning algorithm. While we used standard algorithms based on stochastic gradient descent, other algorithms could be of interest. Furthermore, studying the interaction of model properties like stability enforced via Lipschitz conditions and gradient-based optimization could yield further insights into shortcomings of current approaches. While studying the full interplay of model architecture, training objectives, data and learning algorithms, instead of focusing on isolated properties is hard, it is most likely essential for long-term progress.

Furthermore, there are many more design principles for neural networks. For example, some loss functions like Wasserstein distances or adversarial robustness require models with bounded Lipschitz constants (Arjovsky et al., 2017; Anil et al., 2019). Similarly, using deep networks for ill-posed inverse problems might require stable networks to bound the influence of imprecise measurements. Whereas stability played a role for designing invertible residual networks (i-ResNets), the main focus was on invertibility.

As a summary, this thesis demonstrated how mathematical guarantees of neural network properties can be obtained and even leveraged to go beyond common objectives. Focusing on underlying principles will remain crucial, especially when models become increasingly flexible and interact in complex environments.

Bibliography

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/.

A. Achille and S. Soatto. Emergence of invariance and disentanglement in deep representations. *The Journal of Machine Learning Research*, 19(1):1947–1980, 2018.

R. P. Adams, J. Pennington, M. J. Johnson, J. Smith, Y. Ovadia, B. Patton, and J. Saunderson. Estimating the spectral density of large implicit matrices. *arXiv preprint arXiv:1802.03451*, 2018.

M. Aichler and A. Walch. Maldi imaging mass spectrometry: current frontiers and perspectives in pathology research and practice. *Laboratory Investigations*, 95:422–431, 2015.

A. A. Alemi, I. Fischer, J. V. Dillon, and K. Murphy. Deep variational information bottleneck. In *International Conference on Lerning Representations*, 2017.

A. A. Alemi, B. Poole, I. Fischer, J. V. Dillon, R. A. Saurous, and K. Murphy. An information-theoretic analysis of deep latent-variable models. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

T. Alexandrov. Maldi imaging mass spectrometry: statistical data analysis and current computational challenges. *BMC Bioinformatics*, 13(16):S11, 2012.

C. Anil, J. Lucas, and R. Grosse. Sorting out Lipschitz function approximation. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

L. Ardizzone, J. Kruse, C. Rother, and U. Köthe. Analyzing inverse problems with invertible neural networks. In *International Conference on Learning Representations*, 2019.

M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

S. Arora, Y. Liang, and T. Ma. Why are deep nets reversible: a simple theory, with implications for training. *arXiv preprint*, arXiv:1511.05653, 2015.

U. Ascher. *Numerical methods for evolutionary differential equations*. Computational science and engineering. Society for Industrial and Applied Mathematics, 2008.

A. Atanov, A. Volokhova, A. Ashukha, I. Sosnovik, and D. Vetrov. Semi-conditional normalizing flows for semi-supervised learning. *arXiv preprint arXiv:abs/1905.00505*, 2019.

H. Avron and S. Toledo. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *Journal of the ACM*, 58(2):8:1–8:34, 2011.

D. Barber and F. Agakov. The IM algorithm: A variational approach to information maximization. In *Advances in Neural Information Processing Systemss*, 2003.

P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, Ç. Gülçehre, F. Song, A. J. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018.

J. Behrmann, S. Dittmer, P. Fernsel, and P. Maass. Analysis of invariance and robustness via invertibility of relu-networks. *arXiv preprint arXiv:1806.09730*, 2018a.

J. Behrmann, C. Etmann, T. Boskamp, R. Casadonte, J. Kriegsmann, and P. Maass. Deep learning for tumor classification in imaging mass spectrometry. *Bioinformatics*, 34(7): 1215–1223, 2018b.

J. Behrmann, W. Grathwohl, R. T. Q. Chen, D. Duvenaud, and J.-H. Jacobsen. Invertible residual networks. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

M. I. Belghazi, A. Baratin, S. Rajeshwar, S. Ozair, Y. Bengio, D. Hjelm, and A. Courville. Mutual information neural estimation. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

A. Benninghoven and E. Loebach. Tandem mass spectrometer for secondary ion studies. *Review of Scientific Instruments*, 42:49–52, 1971.

B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer, 2013.

P. Billingsley. *Probability and measure*. Wiley, 3. ed edition, 1995.

T. Boskamp, D. Lachmund, J. Oetjen, Y. C. Hernandez, D. Trede, P. Maass, R. Casadonte, J. Kriegsmann, A. Warth, H. Dienemann, W. Weichert, and M. Kriegsmann. A new classification method for maldi imaging mass spectrometry data acquired on formalin-fixed paraffin-embedded tissue samples. *Biochimica et Biophysica Acta (BBA) - Proteins and Proteomics*, 1865(7):916 – 926, 2017.

C. Boutsidis, P. Drineas, P. Kambadur, E.-M. Kontopoulou, and A. Zouzias. A randomized algorithm for approximating the log determinant of a symmetric positive definite matrix. *Linear Algebra and its Applications*, 533:95 – 117, 2017.

W. Brendel, J. Rauber, A. Kurakin, N. Papernot, B. Veliqi, M. Salathé, S. P. Mohanty, and M. Bethge. Adversarial vision challenge. *arXiv preprint arXiv:1808.01976*, 2018.

T. B. Brown, N. Carlini, C. Zhang, C. Olsson, P. Christiano, and I. Goodfellow. Unrestricted adversarial examples. *arXiv preprint arXiv:1809.08352*, 2018.

J. Bruna, A. Szlam, and Y. LeCun. Signal recovery from pooling representations. In *Proceedings of the 31st International Conference on Machine Learning*, 2014.

R. M. Caprioli, T. B. Farmer, and J. Gile. Molecular imaging of biological samples: localization of peptides and proteins using maldi-tof ms. *Analytical Chemistry*, 69: 4751–4760, 1997.

S. Carlsson, H. Azizpour, A. Razavian, J. Sullivan, and K. Smith. The preimage of rectifier activities. In *International Conference on Learning Representations (workshop)*, 2017.

R. Casadonte, M. Kriegsmann, F. Zweynert, K. Friedrich, G. Bretton, M. Otto, S. Deininger, R. Paape, E. Belau, D. Suckau, D. Aust, C. Pilarsky, and J. Kriegsmann. Imaging mass spectrometry to discriminate breast from pancreatic cancer metastasis in formalin-fixed paraffin-embedded tissues. *Proteomics*, 14:956–964, 2014.

R. Casadonte, R. Longuespee, J. Kriegsmann, and M. Kriegsmann. Maldi ims and cancer tissue microarrays. *Advances in Cancer Research*, 134:173–200, 2017.

B. Chang, L. Meng, E. Haber, L. Ruthotto, D. Begert, and E. Holtham. Reversible architectures for arbitrarily deep residual neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, 2018.

R. T. Q. Chen, J. Behrmann, D. Duvenaud, and J.-H. Jacobsen. Residual flows for invertible generative modeling. In *arXiv preprint arXiv:1811.00995*, 2019.

X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2016.

F. Chollet et al. Keras. https://github.com/keras-team/keras, 2015.

M. Ciccone, M. Gallieri, J. Masci, C. Osendorfer, and F. Gomez. Nais-net: Stable deep networks from non-autonomous differential equations. In *Advances in Neural Information Processing Systems 31*, 2018.

M. Cisse, P. Bojanowski, E. Grave, Y. Dauphin, and N. Usunier. Parseval networks: improving robustness to adversarial examples. In *Proceedings of the 34 International Conference on Machine Learning*, 2017.

D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). In *International Conference on Learning Representations*, 2016.

T. M. Cover and J. A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, New York, NY, USA, 2006.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989.

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

M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and L. Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019.

M. P. Deisenroth, A. A. Faisal, and C. S. Ong. *Mathematics for Machine Learning*. To be published by Cambridge University Press, 2019. URL https://mml-book.github.io/.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

L. Dinh, D. Krueger, and Y. Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using Real NVP. In *International Conference on Learning Representations*, 2017.

K. Dong, D. Eriksson, H. Nickisch, D. Bindel, and A. G. Wilson. Scalable log determinants for gaussian process kernel learning. In *Advances in Neural Information Processing Systems 30*, 2017.

A. Dosovitskiy and T. Brox. Inverting convolutional networks with convolutional networks. In *In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

H. Drucker and Y. Lecun. Improving generalization performance using double backpropagation. *IEEE Transactions on Neural Networks*, 3:991–7, 1992.

C. Etmann. A closer look at double backpropagation. *arXiv preprint arXiv:abs/1906.06637*, 2019.

E. Fetaya, J.-H. Jacobsen, and R. Zemel. Conditional generative models are not robust. *arXiv preprint arXiv:abs/1906.01171*, 2019.

L. Gatys, A. S. Ecker, and M. Bethge. Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2015.

R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel. Imagenet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. In *International Conference on Learning Representations*, 2019.

M. Germain, K. Gregor, I. Murray, and H. Larochelle. MADE: Masked autoencoder for distribution estimation. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

A. Ghassami and N. Kiyavash. Interaction information for causal inference: The case of directed triangle. *arXiv preprint arXiv:abs/1701.08868*, 2017.

A. Gilbert, Y. Zhang, K. Lee, Y. Zhang, and H. Lee. Towards understanding the invertibility of convolutional neural networks. In *26th International Joint Conference on Artificial Intelligence*, 2017.

J. Gilmer, R. P. Adams, I. Goodfellow, D. Andersen, and G. E. Dahl. Motivating the rules of the game for adversarial example research. *arXiv preprint arXiv:1807.06732*, 2018a.

J. Gilmer, L. Metz, F. Faghri, S. S. Schoenholz, M. Raghu, M. Wattenberg, and I. Goodfellow. Adversarial spheres. *arXiv preprint arXiv:1801.02774*, 2018b.

X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011.

A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse. The reversible residual network: Backpropagation without storing activations. In *Advances in Neural Information Processing Systems*, 2017.

I. Goodfellow and N. Papernot. Is attacking machine learning easier than defending it? *Blog post on Feb*, 2017. URL http://www.cleverhans.io/security/privacy/ml/2017/02/15/why-attacking-machine-learning-is-easier-than-defending-it.html.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014.

I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

H. Gouk, E. Frank, B. Pfahringer, and M. Cree. Regularisation of neural networks by enforcing lipschitz continuity. *arXiv preprint arXiv:1804.04368*, 2018.

W. Grathwohl, R. T. Q. Chen, J. Bettencourt, and D. Duvenaud. FFJORD: Scalable reversible generative models with free-form continuous dynamics. In *International Conference on Learning Representations*, 2019.

E. Haber and L. Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, 2018.

B. C. Hall. Lie groups, lie algebras, and representations: An elementary introduction. *Graduate Texts in Mathematics, 222 (2nd ed.), Springer*, 2015.

I. Han, D. Malioutov, H. Avron, and J. Shin. Approximating the spectral sums of large-scale matrices using chebyshev approximations. *SIAM Journal on Scientific Computing*, 39, 2016.

I. Han, H. Avron, and J. Shin. Stochastic chebyshev gradient descent for spectral optimization. In *Advances in Neural Information Processing Systems 31*, 2018.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics, 2001.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

L. Held and D. S. Bove. *Applied Statistical Inference: Likelihood and Bayes*. Springer Publishing Company, Incorporated, 2013.

D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio. Learning deep representations by mutual information estimation and maximization. In *International Conference on Learning Representations*, 2019.

J. Ho, X. Chen, A. Srinivas, Y. Duan, and P. Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

R. A. Horn and C. R. Johnson. *Matrix analysis*. Cambridge University Press, 2012.

M. Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics - Simulation and Computation*, 19(2): 433–450, 1990.

A. Hyvärinen and P. Hoyer. Emergence of phase-and shift-invariant features by decomposition of natural images into independent feature subspaces. *Neural computation*, 12 (7):1705–1720, 2000.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning*, 2015.

J.-H. Jacobsen, A. W. Smeulders, and E. Oyallon. i-revnet: Deep invertible networks. In *International Conference on Learning Representations*, 2018.

J.-H. Jacobsen, J. Behrmann, N. Carlini, F. Tramèr, and N. Papernot. Exploiting excessive invariance caused by norm-bounded adversarial robustness. *arXiv preprint arXiv:1903.10484*, 2019a.

J.-H. Jacobsen, J. Behrmann, R. Zemel, and M. Bethge. Excessive invariance causes adversarial vulnerability. In *International Conference on Learning Representations*, 2019b.

K. Jia. Improving training of deep neural networks via singular value bounding. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

J. Jo and Y. Bengio. Measuring the tendency of cnns to learn surface statistical regularities. *arXiv preprint arXiv:1711.11561*, 2017.

T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks. *arXiv preprint arXiv:1812.04948*, 2018.

D. P. Kingma. Variational inference and deep learning: a new synthesis. In *PhD thesis, University of Amsterdam*, 2017.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

D. P. Kingma and P. Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, 2018.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

R. Kondor and S. Trivedi. On the generalization of equivariance and convolution in neural networks to the action of compact groups. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

A. Kraskov, H. Stögbauer, and P. Grassberger. Estimating mutual information. *Physical Review E*, 69, 2004.

J. Kriegsmann, M. Kriegsmann, and R. Casadonte. Maldi tof imaging mass spectrometry in clinical pathology: a valuable tool for cancer diagnosis. *International Journal of Oncology*, 46:893–906, 2015.

A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images, 2009.

Y. LeCun, Y. Bengio, and G. E. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

K. Li, T. Zhang, and J. Malik. A study of robustness of neural nets using approximate feature collisions, 2019. URL https://openreview.net/forum?id=H1gDgn0qY7.

C. Louizos, K. Swersky, Y. Li, M. Welling, and R. S. Zemel. The variational fair autoencoder. *arXiv preprint arXiv:abs/1511.00830*, 2015.

Y. Lu, A. Zhong, Q. Li, and B. Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. *arXiv preprint arXiv:1710.10121*, 2017.

W. Luo, Y. Li, R. Urtasun, and R. Zemel. Understanding the effective receptive field in deep convolutional neural networks. In *Advances in Neural Information Processing Systems 29*, 2016.

A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.

A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2017.

A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

A. Mahendran and A. Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. *International Journal of Computer Vision*, 120(3):233–255, 2016.

S. Mallat. Understanding deep convolutional networks. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 374 (2065), 2016.

Z. Mhammedi, A. Hellicar, A. Rahman, and J. Bailey. Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

C. A. Micchelli, Y. Xu, and H. Zhang. Universal kernels. *Journal of Machine Learning Research*, 7:2651–2667, 2006.

T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.

G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio. On the number of linear regions of deep neural networks. In *Advances in Neural Information Processing Systems 27*, 2014.

K. P. Murphy. *Machine learning : a probabilistic perspective*. MIT Press, 2013.

E. Nalisnick, A. Matsukawa, Y. W. Teh, D. Gorur, and B. Lakshminarayanan. Do deep generative models know what they don't know? In *International Conference on Learning Representations*, 2019.

S. Nowozin, B. Cseke, and R. Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. In *Advances in Neural Information Processing Systems 29*, 2016.

G. Papamakarios, I. Murray, and T. Pavlakou. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, 2017.

N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, and D. Tran. Image transformer. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS 2017 Workshop Autodiff*, 2017.

G. Pereyra, G. Tucker, J. Chorowski, L. Kaiser, and G. E. Hinton. Regularizing neural networks by penalizing confident output distributions. In *International Conference on Lerning Representations (workshop)*, 2017.

K. B. Petersen and M. S. Pedersen. The matrix cookbook, 2012.

Y. Polyanskiy and Y. Wu. Lecture notes on information theory. *MIT*, 2015.

M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein. On the expressive power of deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

A. Raghunathan, J. Steinhardt, and P. Liang. Certified defenses against adversarial examples. In *International Conference on Learning Representations*, 2018.

A. Ramesh and Y. LeCun. Backpropagation for implicit spectral densities. *arXiv preprint arXiv:1806.00499*, 2018.

D. J. Rezende and S. Mohamed. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning*, 2015.

P. Rodríguez, J. Gonzàlez, G. Cucurull, J. M. Gonfaus, and F. X. Roca. Regularizing cnns with locally constrained decorrelations. In *International Conference on Learning Representations*, 2017.

L. Ruthotto and E. Haber. Deep neural networks motivated by partial differential equations. *arXiv preprint arXiv:1804.04272*, 2018.

S. Sabour, Y. Cao, F. Faghri, and D. J. Fleet. Adversarial manipulation of deep representations. In *International Conference on Learning Representations*, 2016.

A. M. Saxe, Y. Bansal, J. Dapello, M. Advani, A. Kolchinsky, B. D. Tracey, and D. D. Cox. On the information bottleneck theory of deep learning. In *International Conference on Learning Representations*, 2018.

J. Schmidhuber. Learning factorial codes by predictability minimization. *Department of Computer Science, University of Colorado at Boulder*, 1991. URL https://www.bibsonomy.org/bibtex/25bb96ec7c4b81022fd4927804e946564/idsia.

H. Sedghi, V. Gupta, and P. M. Long. The singular values of convolutional layers. In *International Conference on Learning Representations*, 2019.

M. W. Senko, S. C. Beu, and F. W. McLafferty. Determination of monoisotopic masses and ion populations for large biomolecules from resolved isotopic distributions. *Journal of the American Society for Mass Spectrometry*, 6(4):229 – 233, 1995.

S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, New York, NY, USA, 2014.

W. Shang, K. Sohn, D. Almeida, and H. Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

E. A. Shank, M. J. Powers, V. Jojic, and Y.-C. Harn. Deconvolving molecular signatures of interactions between microbial colonies. *Bioinformatics*, 31(12):i142–i150, 2015.

R. Shwartz-Ziv and N. Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

C.-J. Simon-Gabriel, Y. Ollivier, B. Schölkopf, L. Bottou, and D. Lopez-Paz. Adversarial vulnerability of neural networks increases with input dimension. *arXiv preprint arXiv:1802.01421*, 2018.

J. Sokolić, R. Giryes, G. Sapiro, and M. R. Rodrigues. Robust large margin deep neural networks. *IEEE Transactions on Signal Processing*, 65(16):4265–4280, 2017.

Y. Song, R. Shu, N. Kushman, and S. Ermon. Constructing unrestricted adversarial examples with generative models. In *Advances in Neural Information Processing Systems*, 2018.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

M. Stoeckli, P. Chaurand, D. E. Hallahan, and R. M. Caprioli. Imaging mass spectrometry: a new technology for the analysis of protein expression in mammalian tissues. *Nature Medicine*, 7:493–496, 2001.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

L. Theis, A. van den Oord, and M. Bethge. A note on the evaluation of generative models. In *International Conference on Learning Representations*, 2016.

N. Tishby and N. Zaslavsky. Deep learning and the information bottleneck principle. In *IEEE Information Theory Workshop (ITW)*, pages 1–5, 2015.

Y. Tsuzuku, I. Sato, and M. Sugiyama. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. In *Advances in Neural Information Processing Systems 31*, 2018.

A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016a.

A. Van Den Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, 2016b.

A. Virmaux and K. Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In *Advances in Neural Information Processing Systems 31*, 2018.

S. Wang, A. rahman Mohamed, R. Caruana, J. Bilmes, M. Plilipose, M. Richardson, K. Geras, G. Urban, and O. Aslan. Analysis of deep neural networks with extended data jacobian matrix. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

C. S. Withers and S. Nadarajah. log det a = tr log a. *International Journal of Mathematical Education in Science and Technology*, 41(8):1121–1124, 2010.

S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

T. Zhao, D. Zhang, Z. Sun, and H. Lee. Information regularized neural networks, 2019. URL https://openreview.net/forum?id=BJgvg30ctX.

D. Zoran and Y. Weiss. From learning models of natural image patches to whole image restoration. In *Proceedings of the 2011 International Conference on Computer Vision*, 2011.

Appendix A

Experimental Details

A.1 Classification with Invertible Residual Networks

Architecture: We use pre-activation ResNets with 39 convolutional bottleneck blocks with 3 convolution layers each and kernel sizes of $3 \times 3, 1 \times 1, 3 \times 3$ respectively. All models use the ELU nonlinearity (Clevert et al., 2016). In the BatchNorm version, we apply batch normalization before every nonlinearity and in the invertible models we use ActNorm (Kingma and Dhariwal, 2018) between each residual block. The network has 2 downsampling stages after 13 and 26 blocks, where a dimension squeezing operation is used to decrease the spatial resolution. This reduces the spatial dimension by a factor of two in each direction, while increasing the number of channels by a factor of four. All models transform the input data to a $8 \times 8 \times 256$ tensor. Then, we apply BatchNorm, a nonlinearity and average pooling to a get 256-dimensional vector. A linear classifier is used on top of this representation.

Injective Padding: Since our invertible models are not able to increase the dimension of their latent representation, we use injective padding as in (Jacobsen et al., 2018). This operation concatenates channels of zeros to the input, increasing the size of the transformed tensor. This is analogous to the standard practice of projecting the data into a higher-dimensional space using a convolution layer at the input of a model, but this mapping is injective. We add 13 channels of zero to all models tested, thus the input to our first residual block is a tensor of size $32 \times 32 \times 16$. We experimented with removing this step, but found it led to approximately a 2% decrease in accuracy for our CIFAR10 models.

Training: We train for 200 epochs with momentum SGD and a weight decay of 5e-4. The learning rate is set to 0.1 and decayed by a factor of 0.2 after 60, 120 and 160 epochs. For data augmentation, we apply random shifts of up to two pixels for MNIST and shifts/ random horizontal flips for CIFAR(10/100) during training. The inputs for MNIST are normalized to [-0.5,0.5]. For CIFAR(10/100), we normalize by subtracting the mean and dividing by the standard deviation of the training set.

A.2 Generative Modeling with Invertible Residual Networks

MNIST and CIFAR: The structure of our generative models closely resembles that of Glow (Kingma and Dhariwal, 2018). The model consists of *scale-blocks*, which are groups of i-ResNet blocks that operate at different spatial resolutions. After each scale-block, apart from the last, we perform a squeeze operation which decreases the spatial resolution by 2 in each dimension and multiplies the number of channels by 4 (invertible downsampling from (Jacobsen et al., 2018)).

Our MNIST and CIFAR10 models have three scale-blocks. Each scale-block has 32 invertible residual layers. Each layer consists of three convolutions of 3×3, 1×1, 3×3 filters with ELU (Clevert et al., 2016) nonlinearities in between. Each hidden convolutional layer (middle layer when using 3 convolutional layers) has 32 filters in the MNIST model and 512 filters in the CIFAR10 model.

We train for 200 epochs using the Adamax (Kingma and Ba, 2014) optimizer with a learning rate of 0.003. Throughout training we estimate the log-determinant in Equation (5.4) using the power-series approximation (Equation (5.6)) with ten terms for the MNIST model and 5 terms for the CIFAR10 model.

Evaluation: During evaluation, we use the bound presented in Section 5.2.3 to determine the number of terms needed to give an estimate with bias less than 0.0001 bit/dim. We then average over enough samples from Hutchinson's estimator (Hutchinson, 1990), such that the standard error is less than 0.0001 bit/dim. Thus, we can report the bit/dim of our model up to a tolerance of 0.0002.

A.3 Details on Imaging Mass Spectrometry Experiments

Table A.1: Architecture of ResidualNet used in chapter 7. Total number of trainable parameters is 2132130.

Layer	depth	kernel size	stride	# feature maps
Input layer	-	-	-	1
Residual layer	2	5	1	16
Residual layer	2	5	3	32
Residual layer	2	5	1	32
Residual layer	2	5	3	64
Residual layer	2	5	1	64
Residual layer (5x)	2	5	3/1	128
Residual layer	2	5	3	128
Residual layer	2	5	3	256
GlobalPool layer	-	-	-	1
Fully connected (softmax)	-	-	-	1

Table A.2: Description of hyperparameter. The setting is reported in the third and fourth column, where the first entry relates to *IsotopeNet* and the second to *ResidualNet*.

Parameter	Influence	Task ADSQ (Iso // Res)	Task LP (Iso // Res)
λ weight decay	coefficient for weight regularization	0.05 // 0.05	0.01 // 0.001
η^* learning rate	step size in *Adam* algorithm	0.0005 // 0.0005	0.0005 // 0.0005
$\lvert\mathcal{B}\rvert$ batch size	number of samples used per SGD update step	256 // 64	256 // 64
E number of epochs	passes of SGD over training set	300 // 100	30 // 30
p dropout probability	probability to set activations in selected layer to zero	30% // -	30% // -

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst und ausschließlich die angegebenen Quellen und Hilfsmittel verwendet habe.

Jens Behrmann

Bremen, 12. August 2019