Universität Bremen

Double Backpropagation with Applications to Robustness and Saliency Map Interpretability

Author: Christian Etmann

Supervisor: Peter Maaß

Thesis submitted in partial fulfilment of the requirements
for the degree of Dr. rer. nat.

Zentrum für Technomathematik
Universität Bremen

19 December, 2019

Abstract

This thesis is concerned with works in connection to *double backpropagation*, which is a phenomenon that arises when first-order optimization methods are applied to a neural network's loss function, if this contains derivatives. Since feedforward neural networks are constructed in a layerwise fashion, the successive application of the chain rule throughout the layers of these networks yields the desired derivatives according to the famous backpropagation procedure. If these derivatives in turn appear in the form of a loss function, training the neural network results in said double backpropagation.
In this thesis, an extensive analysis of the properties of double backpropagation is performed. This includes the calculation of the gradients themselves, for whose coordinate-independent representation in Hilbert spaces a theory of adjoints of bilinear operators is developed. The explicit calculation of the weight gradients allows for a reduction in computational complexity by roughly a third for a common special case. Furthermore, empirical results are presented which demonstrate a 'pseudo-smoothing' effect on this loss landscape, when using the popular rectified linear units in combination with batch optimization.
From an application-perspective, double backpropagation can be used for reducing a neural network's vulnerability to adversarial attacks. Such an increase in adversarial robustness has been shown to improve the structure of saliency maps, i.e. gradients indicating the discriminative portions of an input image. This work offers an explanation of this so far unexplained phenomenon by considering the alignment between an input image and its saliency map. These findings are verified for networks robustified with double backpropagation.
Tumor typing of imaging mass spectrometry data is an active area of research, which aims to determine the correct type of tumor of a patient's cancerous tissue obtained during surgery. While 'classical' methods from machine learning have been successfully applied to this problem, in this thesis a neural network approach is presented, for which a task-adapted architecture called *IsotopeNet* is developed. This architecture beats both a classical baseline as well as a more standard neural network architecture on two challenging datasets. This approach however yields unsatisfactory accuracies on a multi-laboratory study. Using an attribution method called *layerwise relevance propagation*, the reason for this failure is determined to stem from measurement artifacts induced by the multi-laboratory setting. By penalizing this layerwise relevance propagation with a sparsity-inducing penalty term (a novel method which is named *deep relevance regularization*), the performance of the neural network approach is greatly improved.

Zusammenfassung

Diese Dissertation befasst sich mit Arbeiten in Verbindung mit *Double Backpropagation*, einem Phänomen, welches auftritt, wenn ableitungsbasierte Optimierungsverfahren auf Fehlerfunktionale angewendet werden, die Ableitungen enthalten. Da neuronale Netze schichtweise aufgebaut sind, führt die sukzessive Anwendung der Kettenregel durch das Netzwerk zur bekannten *Backpropagation*. Treten diese Ableitungen wiederum in Form eines Fehlerfunktionals auf, dann kommt es zu besagter Double Backpropagation.
In dieser Arbeit wird eine umfassende theoretische Untersuchung von Double Backpropagation durchgeführt. Dies schließt die Berechnung der Gradienten selbst ein, für deren koordinatenunabhängige Darstellung in Hilberträumen eine Theorie adjungierter Operatoren von bilinearen, stetigen Operatoren entwickelt wird.Die explizite Berechnung der Gewichtsgradienten führt zu einer Reduktion des Berechnungsaufwandes um etwa ein Drittel für übliche Spezialfälle. Zudem werden empirische Resultate präsentiert, die einen 'Pseudo-Glättungseffekt' aufdecken, wenn die beliebte ReLU Aktivierungsfunktion in Kombination mit Batchmethoden verwendet wird.
Aus Anwenderperspektive kann Double Backpropagation dazu genutzt werden, die Anfälligkeit eines neuronalen Netzes für sogenannte *adversarial attacks* zu senken. Es wurde gezeigt, dass ein solcher Anstieg der Robustheit die Struktur von *Saliency Maps* verbessert; also Gradienten, die die diskriminativen Teile des Eingabebilds zeigen sollen. Diese Arbeit liefert eine Erklärung für dieses bisher ungeklärte Phänomen, indem die gemeinsame Ausrichtung des Bildes und der Saliency Map betrachtet wird. Diese Resultate werden anhand mittels Double Backpropagation robustifizierter Netze empirisch gezeigt.
Die Tumortypisierung bildgebender massenspektrometrischer Aufnahmen ist ein aktiver Zweig der Wissenschaft, welcher versucht, den korrekten Tumortyp eines bei einer Operation gewonnenen Tumors zu bestimmen. Während 'klassische' Methoden aus dem Bereich Machine Learning bereits erfolgreich auf dieses Problem angewendet wurden, wird hier ein auf neuronale Netze basierter Ansatz vorgestellt, für welchen eine auf das Problem abgestimmte Architektur namens *IsotopeNet* entwickelt wird. Diese Architektur schlägt sowohl die klassische Vergleichsmethode, als auch eine eher standardmäßige Netzwerkarchitektur auf zwei herausfordernden Datensätzen. Dieser Ansatz führt jedoch zu unzufriedenstellenden Genauigkeiten auf einer Studie mit mehreren Laboren. Mittels einer Relevanz-zuweisenden Methode namens *Layerwise Relevance Propagation* wird gezeigt, dass der Grund für diese Unzulänglichkeit in Messartefakten durch den multizentrischen Versuchsaufbau liegt. Indem die Layerwise Relevance Propagation mit einem Strafterm belegt wird, welcher deren Dünnbesetztheit fördern soll (eine neuartige Methode namens *Deep Relevance Regularization*), wird die Klassifikationsgüte des neuronalen Netzes stark gesteigert.

Acknowledgements

Here, I would like to thank the people who helped me along the way, in one way or another.

I thank the people in and around the π^3 graduate school for making this time an enjoyable and interesting experience: my excellent office mate Jens, Yovany, Lena H, Max W, Max S, Georgia, Johannes, Greta, Sören D, Sören S, Daniel, Delf, Tobias. Many thanks also to the secretaries of our group, Dörte and Judith, as well as Victor for his extensive technical support.

I would also like to thank my supervisor Peter Maaß, as well as Tobias Boskamp for his guidance in all things MALDI. Thank you furthermore to Carola Schönlieb for hosting me in Cambridge and to Sebastian Lunz for the fun collaboration.

I would like to extend special thanks to everybody who answers programming questions on StackOverflow.

I thank all of my friends for reminding me that there is a world outside of the university. I also thank my family and in particular my parents for their support throughout my way so far.

Most of all, I would like to thank Vanessa for supporting me every step of the way and for just being her.

Contents

Chapter 1

Introduction

The topics of this thesis are centered around the area of machine learning. According to Shalev-Shwartz and Ben-David (2014), '[the] term machine learning refers to the automated detection of meaningful patterns in data'.
Machine learning has had a truly transformative impact on many applications over the last few years. In the age of an abundance of data, *learning* the solution to a problem with an automated system has often proven to be easier than engineering solutions purely through classical system knowledge. The recently popularized sub-discipline of *deep learning* (i.e. machine learning using neural networks with many layers) has had a particularly large impact on problems, which so far could not be tackled. The success story of deep learning has seen many state-of-the-art models for quite diverse tasks, such as image classification (Krizhevsky et al., 2012), natural language processing (Vaswani et al., 2017) or learning to play games (Silver et al., 2018).

With a diversification of tasks came a diversification of techniques used to tackle them. One particular, very general technique is *double backpropagation*, which has seen a growing use over the last few yeast. Double backpropagation comes into play when the loss function of a neural network contains derivatives of output nodes with respect to the network's input.

Chapter 2 introduces the foundations of statistical learning theory and neural networks, which are required to understand the later chapters of this thesis.

In Chapter 3, double backpropagation is analyzed analytically, which makes the derivation of an algorithm possible, which reduces the number of computations for special cases by a third. For the theoretical background, a theory of adjoint operators of continuous bilinear operators between Hilbert spaces is developed, which makes the coordinate-independent description possible.

Chapter 4 is concerned with the phenomenon of adversarial perturbations, which are very small perturbations to a classifier's input such that its output is changed. It has recently been discovered (Tsipras et al., 2019), that networks that were trained to be more robust against these kinds of perturbations, seem to exhibit the added benefit of having highly structured *saliency maps*, i.e. gradients. In this chapter, this is explained by considering the alignment between the input image and its respective saliency map. The connection between these is shown both theoretically and empirically.

Chapter 5 is centered around the topic of tumor (sub-)typing using MALDI mass spectra. A domain-adapted neural network architecture named *IsotopeNet* is presented, which yields a higher accuracy on two datasets, beating both a linear classifier based on separately extracted features as well as a standard neural network architecture. On a different, multi-center dataset however, the classification with neural networks results in an unsatisfactory performance. Using *layerwise relevance propagation*, a method of assessing the relevant parts of the input, this is explained through measurement artifacts. Based on this, a sparsifying penalty term on said relevance estimation is applied, which results in a big increase of classification accuracy.

Publications by the Author

This thesis is based on four publications by the author. The (possibly shared) first authorship is underlined.

> Christian Etmann. A Closer Look at Double Backpropagation. Manuscript submitted for publication. Preprint available at arXiv:1906.06637, 2019

Christian Etmann is the sole author of this work and responsible for all aspects of the work. This article is the basis of Chapter 3.

> Christian Etmann, Sebastian Lunz, Peter Maass, and Carola-Bibiane Schönlieb. On the Connection Between Adversarial Robustness and Saliency Map Interpretability. In *Proceedings of the 36th International Conference on Machine Learning*, 97:1823-1832, 2019a.

Christian Etmann contributed to all aspects of this work in equal contribution to Sebastian Lunz. This article serves as the foundation for Chapter 4.

> Jens Behrmann, Christian Etmann, Tobias Boskamp, Rita Casadonte, Jörg Kriegsmann, and Peter Maass. Deep Learning for Tumor Classification in Imaging Mass Spectrometry. In *Bioinformatics*, Volume 34, Issue 7, 01 April 2018, Pages 1215–1223, 97:1823-1832, 2017.

Christian Etmann contributed to all aspects of this work in equal contribution to Jens Behrmann, except for recording and annotating the data. This article is the basis of Chapter 5, along with the following paper.

> Christian Etmann, Maximilian Schmidt, Jens Behrmann, Tobias Boskamp, Lena Hauberg-Lotte, Annette Peter, Rita Casadonte, Jörg Kriegsmann, and Peter Maass. Deep Relevance Regularization: Interpretable and Robust Tumor Typing of Imaging Mass Spectrometry Data. Manuscript submitted for publication. Preprint available at arXiv:1912.05459, 2019b

Christian Etmann contributed to all aspects of this work, except for recording and annotating the data. This article is the basis of Chapter 5, along with the previous paper.

Chapter 2

Machine Learning with Neural Networks

Despite the recency of many of its modern techniques, deep learning is grounded in the same theoretical foundations as other machine learning methods, which is *statistical learning theory*. For this reason, a short introduction to the statistical learning background of machine learning is given, before turning towards specifics of neural networks. The following notation was developed for a lecture series on the mathematical foundations of machine learning, which the author designed and taught together with Jens Behrmann. It is heavily influenced by the book *Understanding Machine Learning* by (Shalev-Shwartz and Ben-David, 2014).

While the field of machine learning is very broad and the following definitions may not cover all of its subtypes, here we will provide the necessary definitions required for understanding the topics in this thesis. The area of machine learning we will focus on is predictive in nature.

2.1 Basics of Statistical Learning Theory

In the following, some notation is introduced. A set $\mathcal{X}$, which is a superset of objects one wishes to make predictions on, will be called the *domain set*. This can for instance be a vector space such as $\mathbb{R}^{256\times 256}$, in which one can represent 8-bit grayscale images of resolution 256-by-256. Usually, one is interested in a $\mathcal{X}$-valued random variable, which we will call X and whose distribution is denoted $\mathcal{D}_{\mathrm{X}}$. We write $X \sim \mathcal{D}_{\mathrm{X}}$ and call realizations of X, denoted by a lower-case x, a *feature (vector)*. If $\mathcal{X}$ is finite-dimensional, the entries of the vector x are also individually called its features. Tasks in which only realizations of X are available for the creation of a machine learning model belong to the realm of *unsupervised learning*.

In *supervised learning* on the other hand, one deals not only with features, but also with target values. Analogously, one is concerned with a random *target variable* $Y \sim \mathcal{D}_{\mathrm{Y}}$, which takes values in some set $\mathcal{Y}$. Together, these appear as feature-target-pairs $(X, Y) \sim \mathcal{D}_{\mathrm{X,Y}}$, where $\mathcal{D}_{\mathrm{X,Y}}$ is the joint distribution of X and Y.

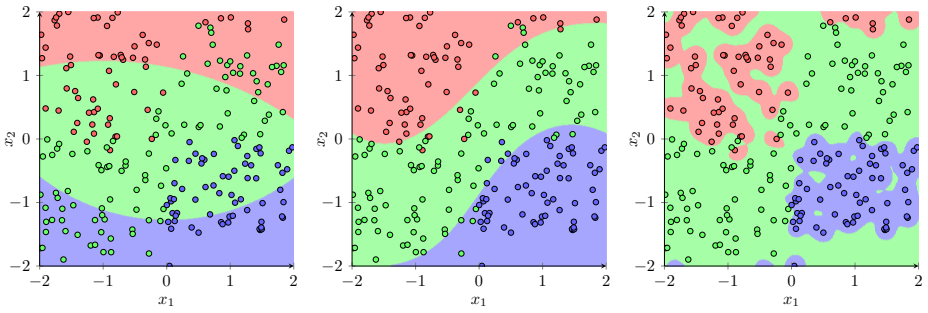


Figure 2.1: The problem of under- and overfitting exemplified on a 3-class toy problem using a kernel discriminant analysis classifier (Mika et al., 1999), taken from (Etmann, 2016). The training set is represented via colored dots, while the resulting decision boundaries are drawn in the background. The left picture represents the case of underfitting, showing a low training accurarcy. While the middle image does not exhibit perfect training accuracy, the resulting decision boundaries have a very regular shape, yielding a plausible classification model – it generalizes well. The image on the right shows a classical case of overfitting – perfect training accuracy, but implausible decision boundaries.

Two important tasks in machine learning are classification and regression, which can both be described as the search for a function $f : \mathcal{X} \to \mathcal{Y}$, such that $f(x) \approx y$ for feature-target-samples (x, y) from $\mathcal{D}_{\mathrm{X,Y}}$. Here, f is to be chosen from a pre-defined set of functions $\mathcal{H} \subset \{g \mid g : \mathcal{X} \to \mathcal{Y}\}$, the *hypothesis class*. If $\mathcal{Y} = \mathbb{R}^m$ (and the range of f has infinitely many elements), the task of finding such a function is called *regression*. If $\mathcal{Y}$ is finite, one speaks of *classification*. In this case, the classes are often encoded as $\mathcal{Y} = \{1, -1\}$ (for two-class or *binary* classification) or $\mathcal{Y} = [C] =: \{1, \dots, C\}$ (for multi-class classification). The individual realizations of Y in a classification tasks are also called *labels*.

In machine learning, the choice of f is made based on data, i.e. a finite set of realizations of random variables, the *training set*. The act of choosing f based on this training set is furthermore called *training*. However, one is not primarily interested in the quality of predictions on this training set (which would be perfectly solved by a lookup-table), but rather in the performance on previously unseen samples of the random variable. This is the central problem in supervised learning and known as *generalization*. In order to quantify this, a set of labelled *test data* is set aside and the goodness of fit is judged on this test set. For the evaluation of the model performance during training (e.g. for the choice of hyperparameters), often an additional *validation set* is also kept.

One common occurence is that of the considered model being too adapted to the training data to generalize well. This is known as *overfitting* and expresses itself via a high accuracy on the training set, but a low accuracy on a test set. The case of low accuracies both on the training and test sets is on the other hand called *underfitting*. These phenomena are exemplified in Figure 2.1.

2.1.1 Risk

A central aspect of machine learning is how to measure a model's success, i.e. how to quantify that $f(x) \approx y$. Mathematically, this concept is realized via the *loss* or *error* between the model's output and some desired value. In supervised learning tasks, this can usually be understood as the deviation of its prediction from the respective target value, e.g. the label of an image in a classification setting.

Definition 2.1.1 (Loss function)**.** *Let $\mathcal{H}$ be a hypothesis class an let Ω be some set. Then $l : \mathcal{H} \times \Omega \to \mathbb{R}_{\geq 0}$ is called a loss function, where $\mathbb{R}_{\geq 0}$ denotes the non-negative real numbers.*

The above definitions for hypothesis classes and loss functions are very general and can cover a lot of special cases from supervised and unsupervised learning. In the following, for newly-defined loss functions, the more common formulation, denoted ℓ, will be introduced along with the general form from above.

Definition 2.1.2 (Squared loss)**.** *Let $\Omega = \mathcal{X} \times \mathcal{Y}$ for some domain set $\mathcal{X}$ and $\mathcal{Y} = \mathbb{R}^m$. Let*

$$\begin{aligned} \ell_{\text{sq}} : \mathcal{Y} \times \mathcal{Y} &\to \mathbb{R}_{\geq 0} \\ (z, y) &\mapsto \|y - z\|_2^2, \end{aligned}$$

then for the hypothesis class $\mathcal{H} \subseteq \{f \mid f : \mathcal{X} \to \mathcal{Y}\}$, the function

$$l_{\text{sq}} : (f, (x, y)) \mapsto \ell_{\text{sq}}(f(x), y)$$

is called the squared loss function*.*

Since we want to judge the performance not just for a single point, but on the whole data-generating distribution, we consider the expected loss over this distribution.

Definition 2.1.3. *Let $l : \mathcal{H} \times \Omega \to \mathbb{R}_{\geq 0}$ be a loss function and let $\mathcal{D}$ be a distribution over Ω. Then for $f \in \mathcal{H}$ the expected loss over $\mathcal{D}$,*

$$\mathcal{L}_{\mathcal{D}}(f) := \mathbb{E}_{V \sim \mathcal{D}}[l(f, V)],$$

is called the risk *of f over $\mathcal{D}$ with respect to l.*

For classification tasks in particular, the 0-1-loss is suited for the evaluation of a classifier's performance.

Definition 2.1.4. *Let $\ell_{0-1} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$ be given by*

$$\ell_{0-1}(z, y) := \begin{cases} 0, & \text{if } y = z \\ 1, & \text{if } y \neq z \end{cases}$$

and call

$$l_{0-1}(f, (x, y)) := \ell_{0-1}(f(x), y)$$

the 0-1-loss function*. Then its risk over the distribution $\mathcal{D}_{\text{X,Y}}$ with respect to ℓ_{0-1} is called the* error rate *of $f : \mathcal{X} \to \mathcal{Y}$.*

The error rate as well as the *accuracy* $1 - \mathcal{L}_{\mathcal{D}_{X,Y}}(f)$ is mostly considered in classification tasks (where $\mathcal{Y}$ is finite).

The goal of training is to find a model that *generalizes* well under the data distribution of interest, i.e. has a low risk. In real-world applications, one does not have access to this distribution – otherwise one would be able to construct a classifier that is in some sense optimal, the *bayes optimal classifier*. In the presence of data however, one can apply the principle of *empirical risk minimization*, which is the minimization of an approximation of the true risk.

Definition 2.1.5. *Let $v^{(1)}, \dots, v^{(N)} \in \Omega$ be independent, identically distributed (i.i.d.) samples of a random variable over Ω. Let further $\mathcal{H}$ be some hypothesis class and $l : \mathcal{H} \times \Omega \to \mathbb{R}_{\geq 0}$ be a loss function. Then the task of minimizing*

$$\mathcal{L}_{\mathcal{T}}(f) := \frac{1}{N} \sum_{i=1}^{N} l(f, v^{(i)}) \tag{2.1}$$

over $f \in \mathcal{H}$ is called empirical risk minimization *(ERM) over $\mathcal{H}$ with respect to l and the* training set $\mathcal{T} = \{v^{(1)}, \dots, v^{(N)}\}$.

Statistical learning theory offers various stochastic guarantees for a minimizer of (2.1) to lie close to its true risk, depending on the specific assumed learning model. Interested readers are referred to (Shalev-Shwartz and Ben-David, 2014) for an overview.

Example 2.1.1 (Linear regression)**.** *Let $\mathcal{X} = \mathbb{R}^n$ and $\mathcal{Y} = \mathbb{R}^m$. Let further*

$$\mathcal{H} = \{f : \mathcal{X} \to \mathcal{Y} \mid f \text{ linear}\}$$

be a hypothesis class. By identifying $\mathcal{H}$ with the space of matrices $\mathbb{R}^{m \times n}$, the linear regression

$$\min_{W \in \mathbb{R}^{n \times m}} \frac{1}{N} \sum_{i=1}^{N} \|Wx^{(i)} - y^{(i)}\|_2^2$$

is the ERM task over $\mathcal{H}$ with respect to the squared loss over the training set $\mathcal{T} = \{(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)})\}$ of samples of a distribution over $\Omega = \mathcal{X} \times \mathcal{Y}$.

The example of linear regression demonstrates, how empirical risk minimization, i.e. a minimization over a set of functions, can often be reframed as a minimization over some parameter space. Many model classes (such as neural networks) are inherently parametric, meaning they form a hypothesis class $\mathcal{H} = \{f_\Theta \mid \Theta \in \mathcal{P}\}$, where $\mathcal{P}$ is the set of all possible parameters. A special case – which is a common setting for classification – are functions that model probabilities or probability densities. When viewing these as functions over their parameters, they are called *likelihood functions*.

2.1.2 Likelihood Models

Definition 2.1.6. *Let $f_\Theta : \Omega \to \mathbb{R}_{\geq 0}$ be either a probability mass function (pmf) or a probability density function (pdf) for all $\Theta \in \mathcal{P}$, where $\mathcal{P}$ is some set of parameters. Then for fixed $v \in \Omega$, the function*

$$\Theta \mapsto f_\Theta(v)$$

is called a likelihood function. For $v^{(1)}, \dots, v^{(N)} \in \Omega$*, we call the the product*

$$f_\Theta(v^{(1)}, \dots, v^{(N)}) := \prod_{i=1}^{N} f_\Theta(v^{(i)}) \tag{2.2}$$

the likelihood of $v^{(1)}, \dots, v^{(N)}$ *under* f_Θ*.*

Under the assumption that each $v^{(i)}$ is i.i.d. with respect to the pmf/pdf f_Θ, equation (2.2) describes the joint probability mass/density of $v^{(1)}, \dots, v^{(N)}$. Note that while it is also possible to define likelihood functions for sets of points *without* assuming independence, this assumption makes the factorization (2.2) possible.

Note that the above enables one to also define conditional likelihood functions. Let $f_\Theta(Y = y|X = x)$ describe the conditional probability that $Y = y$, given $X = x$. If $\mathcal{Y}$ is a finite set, this construction is a suitable framework for classification, where $f_\Theta(Y = y|X = x)$ describes the probability of x belonging to class y.

In applications, f_Θ is often a model for the pmf/pdf of a random variable with a true, unknown distribution $\mathcal{D}$. In specific settings, a family of models that this distribution belongs to may be known through knowledge about the underlying data-generating process. If for example, the data is known to stem from a Gaussian distribution, the task of finding the correct distribution is reduced to determining the correct mean vector and covariance matrix, which together fully parametrize a Gaussian distribution. But even in cases where such system knowledge is not available, a highly expressive model parametrized by some $\Theta \in \mathcal{P}$ may still yield a realistic model for the underlying data-generating distribution, given that one is able to make a good choice for Θ. One such parameter choice paradigm is *maximum likelihood estimation* (MLE).

Definition 2.1.7 (Maximum likelihood estimation)**.** *Let* $v^{(1)}, \dots, v^{(N)} \in \Omega$*. For all* $v \in \Omega$*, let* $\Theta \mapsto f_\Theta(v)$ *be a likelihood function. Then the task of maximizing*

$$\prod_{i=1}^{N} f_\Theta(v^{(i)})$$

with respect to Θ *is called* maximum likelihood estimation *of* $v^{(1)}, \dots, v^{(N)}$ *with respect to* f_Θ*.*

Philosophically, the idea of MLE is thus to choose Θ such that f_Θ describes the data best among the hypothesis class. In other words, one tries to find the most 'realistic' model from a hypothesis class, given only the data.

MLE with equation (2.2) has two practical disadvantages:

- For high N, numerical underflow can be a problem due to the multiplication of many small numbers.
- Often, the actual maximization is performed using derivative-based methods. This necessitates employing the product rule N times, requiring many computations.

In order to overcome these problems, it is common to instead look at the *log-likelihood*. Without loss of generality, we will always denote with log the natural logarithm. For strictly positive f_Θ, it holds that

$$\log(f_\Theta(v) \cdot f_\Theta(w)) = \log(f_\Theta(v)) + \log(f_\Theta(w)),$$

for $v, w \in \Omega$, such that equation (2.2) can be transformed to

$$\log(f_\Theta(v^{(1)}, \dots, v^{(N)})) = \sum_{i=1}^{N} \log(f_\Theta(v^{(i)})).$$

Since the logarithm is a strictly monotonically increasing function on $(0, \infty)$, the maximizer of a likelihood is the same as the maximizer of its respective log-likelihood:

$$\arg\max_{\Theta \in \mathcal{P}} \prod_{i=1}^{N} f_\Theta(v^{(i)}) = \arg\max_{\Theta \in \mathcal{P}} \sum_{i=1}^{N} \log(f_\Theta(v^{(i)})) = \arg\min_{\Theta \in \mathcal{P}} \sum_{i=1}^{N} -\log(f_\Theta(v^{(i)}))$$

Here, we furthermore made use of the fact that a maximizer of the log-likelihood is a minimizer of the *negative log-likelihood*, which proves that a maximum likelihood estimate is an empirical risk minimizer, given a specific choice of loss function.

Definition 2.1.8. *Let*

$$\mathcal{H} = \{f_\Theta : \Omega \to (0, 1] \mid f_\Theta \text{ is a pmf over } \Omega \text{ for all } \Theta \in \mathcal{P}\}$$

be a hypothesis class. Then

$$\begin{aligned} l : \mathcal{H} \times \Omega &\to \mathbb{R}_{\geq 0} \\ (f_\Theta, v) &\mapsto -\log(f_\Theta(v)) \end{aligned}$$

is called a negative log-likelihood loss function*.*

Note that the requirement that $f_\Theta \in \mathcal{H}$ is a probability *mass* function (instead of a density function) guarantees that $-\log(f_\Theta(v)) \geq 0$ for all $v \in \Omega$.

2.1.3 Classification with Likelihood Models

In the following, we will see how the above principles can be applied to the task of classification by considering classifiers built on the estimation of class membership probabilities. In the following, let $\Delta_C := \{p \in [0, 1]^c \mid \sum_{i=1}^{C} p_i = 1\}$.

Definition 2.1.9. *Let $\Omega = [M]$. Then for a vector $p \in \Delta_C$, the probability mass function over Ω*

$$k \mapsto \text{Cat}(k \mid p) := \prod_{i=1}^{C} p_i^{\delta_{ik}}$$

defines a categorical distribution *over Ω, where δ denotes the Kronecker-delta.*

Murphy (2012) also popularized the term *Multinoulli distribution*, as a generalization of the 2-class *Bernoulli distribution*.

In Definition 2.1.9, $p \in \Delta_C$ appears as a constant. If p is however given by a parametric function, e.g. $p = f_\Theta(x) \in \text{int}(\Delta_C)$, one may define a conditional likelihood function $\Theta \mapsto \text{Cat}(k \mid f_\Theta(x))$. If $y = e^{[k]}$, the k-th standard unit vector in C-dimensional Euclidean space, then the respective negative log-likelihood loss admits the representation

$$-\log(\text{Cat}(k|f_\Theta(x))) = -\sum_{i=1}^{C} y_i \log(f_\Theta(x)_i).$$

Definition 2.1.10 (Likelihood classification)**.** *Let X be a random variable over $\mathcal{X}$ and let Y be a random variable over $\mathcal{Y} = \{e^{[i]} \mid i = 1, \dots, C\}$. For a hypothesis class $\mathcal{H} = \{f_\Theta : \mathcal{X} \to \operatorname{int}(\Delta_C) \mid \Theta \in \mathcal{P}\}$, the loss*

$$\begin{aligned} l_{\mathrm{NLL}} : \mathcal{H} \times (\mathcal{X} \times \mathcal{Y}) &\to \mathbb{R}_{\geq 0} \\ f_\Theta, (x, y) &\mapsto -\sum_{i=1}^{C} y_i \log\left(f_\Theta(x)_i\right), \end{aligned}$$

is called the negative log-likelihood (NLL) loss function. For a model $f_\Theta \in \mathcal{H}$ obtained using ERM over a training set of samples from $\mathcal{D}_{\mathrm{X,Y}}$ with respect to the NLL loss, the classifier F_Θ given by

$$\begin{aligned} F_\Theta : \mathcal{X} &\to [C] \\ x &\mapsto \arg\max_i f_\Theta(x)_i, \end{aligned}$$

is called a likelihood based classifier*.*

In summary, in order to obtain a prediction model, one can employ the principle of empirical risk minimization, where the average loss over a training dataset is minimized with respect to models from a pre-defined hypothesis class. While there are many different loss functions to choose from, there is a more or less canonical choice for likelihood-based models: By using the negative log-likelihood loss function, one applies the principle of maximum likelihood estimation. This is for example the case, when a model outputs estimated class membership probabilities for classification problems. In the following section, a set of hypothesis classes that can be used for classification will be defined in neural networks.

2.2 Introduction to Neural Networks

While providing an overarching definition for *all* types of neural networks is a difficult task, they can generally be described as non-linear, parametric functions which consist of a multitude of very simple non-linear, possibly parametric functions. These 'building blocks' – also called *layers* – are typically coupled via function composition. As this thesis is focused on classification using *feedforward neural networks*, we narrow down the definition of neural networks to accommodate this view. The term *feedforward* reflects the view that the 'information' within a feedforward neural network flows only in one direction. This is in contrast to *recurrent neural networks*, which incorporate a sort of feedback.

Definition 2.2.1 (Feedforward neural network)**.** *Let $\mathcal{X}_0, \dots, \mathcal{X}_L$ be vector spaces and let $f_i : \mathcal{X}_{i-1} \to \mathcal{X}_i$ for $i \in [L]$, where at least one function f_i is parametrized by a vector $\Theta_i \in \mathcal{P}_i$, where the vector space $\mathcal{P}_i$ is called a* parameter space*. Then*

$$f_\Theta := f_L \circ \cdots \circ f_1 : \mathcal{X}_0 \to \mathcal{X}_L$$

is called a feedforward neural network *with L layers and the f_i are called the* layers *of this network. L is also called the* depth *of the network. For some $x \in \mathcal{X}_0$, the value $(f_i \circ \cdots \circ f_1)(x)$ is called an* activation *of the i-th layer. The network's parameter space $\mathcal{P}$ is defined to be the cartesian product of all layers' parameter spaces, where $\Theta \in \mathcal{P}$.*

What exactly counts as a layer is highly subjective, since any sequence of consecutive layers can also be regarded as a single layer – or even as a full-fledged feedforward neural network. The exact combination of layers that constitute such a network is also referred to as the *architecture* of this neural network.

While we defined layers abstractly as functions between vector spaces, the overwhelming majority of layers in the literature map between finite-dimensional, real vector spaces. The exact type of data typically determines, which representation is the most useful.

- A vector from $\mathbb{R}^n$ may represent unstructured tabular data as well as data along an axis. The latter case is examined in Chapter 5, where each coordinate of a vector from this space represents the abundance of ions measured at a certain mass-to-charge-ratio on a mass spectrum.
- Digital grayscale images can be described by the amount of brightness in each pixel. As such, they can be viewed as elements of $\mathbb{R}^{n_1 \times n_2}$ (with appropriately chosen $n_1, n_2 \in \mathbb{N}$).
- Color images have an additional channel axis, which – depending on the representation – may describe different things. A standard representation is that of RGB images, where each channel describes the amount of either red, green or blue colored light. This view readily generalizes to hyperspectral data with more than three channels. Such images can thus be represented in $\mathbb{R}^{n_1 \times n_2 \times n_\gamma}$, where n_γ denotes the number of channels ($n_\gamma = 3$ for RGB images).
- This concept generalizes to higher-order data, such as multi-channel 3D volumes. Such higher-order structures can be understood as multidimensional arrays and are often called *tensors* in the neural network literature (cf. e.g. LeCun et al. (2015)).

We point out that in reality, data is often restricted to a specific subset of the above vector spaces. The pixel values for an image may for example be scaled such that they each lie in $[0, 1]$ or in $\{0, \dots, 255\}$.

Although elements from $\mathbb{R}^{n_1 \times \cdots \times n_d}$ can equivalently be represented in $\mathbb{R}^{n_1 \cdot \ldots \cdot n_d}$, the first representation often allows for a more perspicuous definition of layers between these spaces. This is particular evident in the case of image data, where certain layer types operate 'locally', i.e. on a certain geometrical neighborhood of a pixel. This is in particular of importance for convolutional layers, which will be defined shortly.

2.2.1 Layers

The most basic of all layers is the *dense layer* or *fully-connected layer*.

Definition 2.2.2 (Dense layer)**.** *Let*

$$\begin{aligned} f : \mathbb{R}^n &\to \mathbb{R}^m \\ x &\mapsto \varphi(Wx + b), \end{aligned}$$

where $W \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $\varphi : \mathbb{R}^m \to \mathbb{R}^m$ is a non-linear function. Then f is called a dense layer *with* weight matrix W, bias vector b *and* activation function φ.

If the weight matrix has a band structure, this type of layer is also referred to as a *locally-connected layer*[1].

Neural networks that consist exclusively of dense layers are called *multilayer perceptrons*. In theory, these multilayer perceptrons are *universal approximators* (Hornik et al., 1989), meaning that they can fit any continuous function on a bounded domain arbitrarily well. This statement is valid even for networks with only 2 dense layers. In practice, however, fitting highly complicated functions (such as a labeling function for complex image datasets) would require prohibitively large weight matrices and huge datasets. For these types of problems, the theoretical guarantees offered by universal approximation unfortunately remain purely theoretical.

Two techniques in particular have emerged over the last few years, to which the good performance even on said complex image datasets can be attributed, and which are at least in part responsible for the sharp rise in the use of neural networks. One is the principle of *deep learning* (cf. LeCun et al., 2015), which is the use of many layers, motivated by a 'hierarchical' extraction of increasingly abstract features. The other aspect is the use of *convolutional layers* (LeCun et al., 1989), that employ multi-channel discrete convolutions in place of the matrix-vector multiplication of dense layers. Using convolutional filters is motivated by their use in image processing, where many classical tasks (such as blurring, edge detection or compression) involve convolutions (cf. Bredies and Lorenz, 2018).

Definition 2.2.3 (Multi-channel discrete convolution)**.** *For $x \in \mathbb{R}^{n_1 \times \cdots \times n_d \times n_\gamma}$, we define the* j-th channel of x*, denoted $x^{\langle j \rangle} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, via*

$$x^{\langle j \rangle}_{k_1,\ldots,k_d} := x_{k_1,\ldots,k_d,j}.$$

For $w \in \mathbb{R}^{q_1 \times \cdots \times q_d \times m_\gamma \times n_\gamma}$, we further define the filter between the j-th channel of the input and the i-th channel of the output*, denoted $w^{\langle i,j \rangle} \in \mathbb{R}^{q_1 \times \cdots \times q_d}$, via*

$$w^{\langle i,j \rangle}_{k_1,\ldots,k_d} := w_{k_1,\ldots,k_d,i,j}.$$

Let

$$* : \mathbb{R}^{q_1 \times \cdots \times q_d} \times \mathbb{R}^{n_1 \times \cdots \times n_d} \to \mathbb{R}^{m_1 \times \cdots \times m_d}$$

be a discrete convolution operator. Let

$$y^{\langle i \rangle} := \sum_{j=1}^{n_\gamma} w^{\langle i,j \rangle} * x^{\langle j \rangle},$$

the i-th channel of $y \in \mathbb{R}^{p_1 \times \cdots \times p_k \times m_\gamma}$*, which is given by*

$$y_{k_1,\ldots,k_d,i} := y^{\langle i \rangle}_{k_1,\ldots,k_d}.$$

With this, we define a multi-channel convolution operator

$$\circledast : \mathbb{R}^{q_1 \times \cdots \times q_k \times m_\gamma \times n_\gamma} \times \mathbb{R}^{n_1 \times \cdots \times n_d \times n_\gamma} \to \mathbb{R}^{m_1 \times \cdots \times m_d \times m_\gamma}$$

as the operator which maps (w, x) to $y =: w \circledast x$.

[1] These are often defined for more general data types, but since in this work we only need locally-connected layers for data from $\mathbb{R}^n$, we only provide this much simpler definition.

There are many different variants of convolution operators $*$ used in practice. These typically differ by how their inputs' boundaries are handled ('valid' or 'zero-padded' convolutions) and whether and how much downsampling is involved ('strided' convolutions). For a comprehensive overview over convolutional arithmetic for neural networks, see (Dumoulin and Visin, 2016). Note that most frameworks actually implement a discrete *cross-correlation* instead of a discrete convolution (cf. Abadi et al., 2015), which is equivalent up to reparametrization. With the multi-channel convolution defined, convolutional layers can finally be introduced.

Definition 2.2.4 (Convolutional layer)**.** *Let*

$$\begin{aligned} f : \mathbb{R}^{n_1 \times \cdots \times n_d \times n_\gamma} &\to \mathbb{R}^{m_1 \times \cdots \times m_d \times m_\gamma} \\ x &\mapsto \varphi(w \circledast x + b), \end{aligned}$$

where $w \in \mathbb{R}^{q_1 \times \cdots \times q_d \times m_\gamma \times n_\gamma}$, $b \in \mathbb{R}^{m_1 \times \cdots \times m_d \times m_\gamma}$ *and*

$$\varphi : \mathbb{R}^{m_1 \times \cdots \times m_d \times m_\gamma} \to \mathbb{R}^{m_1 \times \cdots \times m_d \times m_\gamma}$$

is a non-linear function. Then f is called a convolutional layer *with* filter kernel w*,* bias vector b *and* activation function φ*.*

Neural networks containing convolutional layers are also called *convolutional neural networks* (CNNs).

Apart from relying on strided convolutions for downsampling, another common method for doing so is *max pooling*.

Definition 2.2.5. *The function defined by*

$$f : \mathbb{R}^{n_1 \times \cdots \times n_d \times n_\gamma} \to \mathbb{R}^{\lfloor n_1/c \rfloor \times \cdots \times \lfloor n_d/c \rfloor \times n_\gamma}$$

where

$$(f(x))_{i_1,\ldots,i_d,i_\gamma} := \max_{\substack{(c-1)\cdot i_j < k_j \leq c \cdot i_j \\ j \in \{1,\ldots,d\}}} x_{k_1,\ldots,k_m,i_\gamma},$$

is called a max-pooling layer with downsampling factor (or stride*) $c \in \mathbb{N}$ (where it holds that $c \leq \min_i n_i$).*

2.2.2 Activation Functions

Activation functions are elements of various neural network layers such as dense layers (Def. 2.2.2) and convolutional layers (Def. 2.2.4).

One widely-used type of activation function are those that are defined in a coordinate-wise fashion.

Definition 2.2.6 (Coordinate-wise activation function)**.** *Given $\varphi : \mathbb{R} \to \mathbb{R}$, we define $\tilde{\varphi} : \mathbb{R}^{n_1 \times \cdots \times n_d} \to \mathbb{R}^{n_1 \times \cdots \times n_d}$ by*

$$\tilde{\varphi}(x)_{k_1,\ldots,k_d} = \varphi(x_{k_1,\ldots,k_d})$$

and call $\tilde{\varphi}$ a coordinate-wise activation function*.*

Table 2.1: A selection of commonly-used coordinate-wise activation functions. For leaky ReLU and the exponential linear unit, $\alpha > 0$ is a fixed hyperparameter.

Name	Mapping	Range
Sigmoid Function	$\sigma(x) = \frac{1}{1+\exp(-x)}$	$(0, 1)$
Hyperbolic Tangent	$\tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$	$(-1, 1)$
Rectified Linear Unit (Nair and Hinton, 2010)	$\mathrm{ReLU}(x) = \max(0, x)$	$[0, \infty)$
Softplus (Dugas et al., 2001)	$\mathrm{softplus}(x) = \log(1+\exp(x))$	$(0, \infty)$
Leaky ReLU (Maas et al., 2013)	$\mathrm{LeakyReLU}(x) = \max(\alpha x, x)$	$\mathbb{R}$
Exponential Linear Unit (Clevert et al., 2015)	$\mathrm{ELU}(x) = \begin{cases} x & x \geq 0 \\ \alpha \cdot (\exp(x) - 1) & x < 0 \end{cases}$	$(-\alpha, \infty)$

In the following, the extension of a scalar-domain function φ to higher-order tensors will simply also be denoted φ. In Table 2.1, some commonly-used coordinate-wise activation functions are presented.

The most ubiquitous *non*-coordinate-wise activation function on the other hand is $\mathrm{softmax} : \mathbb{R}^C \to \mathbb{R}^C$ given by

$$\mathrm{softmax}(x)_i = \frac{\exp(x_i)}{\sum_{k=1}^{C} \exp(x_k)},$$

such that $\mathrm{softmax}(x) \in \mathrm{int}(\Delta_C)$. This allows us to interpret its output as a categorical distribution. In particular, neural networks for classification typically employ a dense layer with a softmax activation function as the output layer. Training such a neural network with the negative log-likelihood loss results in a likelihood-based classifier (Definition 2.1.10).

In Figure 2.2, a typical example how such a neural network for the classification may look in practice is depicted.

2.2.3 Training of Neural Networks

The goal in training a neural network is in essence not different from any other prediction model, such that the principles of statistical learning theory (Section 2.1) still apply. After choosing a network architecture and a loss function for the task at hand, the network is trained using ERM. This means that, given the hypothesis class

$$\mathcal{H} = \{f_\Theta \mid \Theta \in \mathcal{P}\}$$

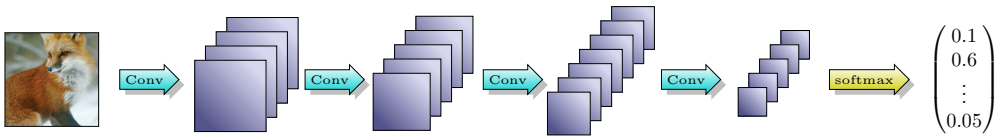


Figure 2.2: Typical example of a small CNN for classification. The 3-channel input RGB image is processed by a cascade of convolutional layers with varying number of feature maps. The resolution of each the feature map monotonically decreases over the depth of the network. In its output layer, a dense layer with a softmax activation function provides an estimation of class membership probabilities. Note that before the application of the dense layer, its input is converted to a single column vector (not depicted).

of neural networks and a loss function l, training the neural network with training data $\mathcal{T} = \{v^{(1)}, \dots, v^{(N)}\}$ amounts to solving

$$\min_{\Theta} \frac{1}{|\mathcal{T}|} \sum_{v \in \mathcal{T}} l(f_\Theta, v).$$

This minimization is then usually attempted with gradient descent and related first-order methods.

Definition 2.2.7. *In the following, let*

$$\mathcal{L}_\mathcal{T}(\Theta) = \frac{1}{|\mathcal{T}|} \sum_{v \in \mathcal{T}} l(f_\Theta, v)$$

be the average loss over $\mathcal{T}$. We call

$$\Theta^{k+1} = \Theta^k - \tau \nabla \mathcal{L}_\mathcal{T}(\Theta^k),$$

the gradient descent update rule *over $\mathcal{T}$ at step k, assuming $\mathcal{L}_\mathcal{T}$ is differentiable in Θ^k. Moreover, we call $\tau > 0$ the* learning rate.

Θ_0 is usually randomly initialized according to a distribution depending on the exact architecture of the network. Two of the most popular initialization schemes are given in (Glorot and Bengio, 2010) and (He et al., 2015).

In practice, gradient descent over the whole training set $\mathcal{T}$ is rarely used. This is because for high $N = |\mathcal{T}|$ (which can run in the millions), calculating N gradients for a single gradient descent step imposes an unnecessarily high computational burden on the training. Just like $\mathcal{L}_\mathcal{T}(\Theta)$ is simply an unbiased estimate of the true risk, for any randomly chosen subset $\mathcal{B}$ of $\mathcal{T}$, $\mathcal{L}_\mathcal{B}(\Theta)$ also provides an unbiased estimate of the true risk[2].

The idea of *(batch) stochastic gradient descent* is thus to randomly choose a smaller subset $\mathcal{B}$ of $\mathcal{T}$ in each step, and perform gradient descent over $\mathcal{B}$. These small subsets are called *(mini-)batches*, whose *batch size* yields a trade-off between optimization speed and estimation variance. Instead of choosing a random subset in every step, the training set is often randomly partitioned into batches and the gradient descent steps are performed

[2]Under the assumption of i.i.d. samples of the true distribution.

over these batches. A full iteration over the training set is called an *epoch*, after which the training set is newly randomly partitioned (cf. Goodfellow et al., 2016). It should be noted that there are many variants of stochastic gradient, of which e.g. Adam (Kingma and Ba, 2014) is a popular one.

General convergence guarantees and rates can be given for restrictive settings such as (strictly) convex losses and Lipschitz continuous gradients, which are not met in the case of neural networks. Apart from the general non-convexity of neural network losses, using activation functions such as (leaky) ReLU results in discontinuous gradients, such that this convergence theory is not applicable. In this non-differentiable case, even the condition $\nabla\mathcal{L}_{\mathcal{T}}(\Theta^*) = 0$ is not a necessary condition for Θ^* being a minimizer. See (Bottou et al., 2018) for an overview over the theoretical guarantees for convergence.

The learning rate τ may be fixed, controlled by a decay scheme or dynamically adapted. Learning rate control schemes for neural networks are an active area of research and seem to contradict the conventional wisdom from optimization, see for example (Smith, 2017).

An open question at this point is still how to actually calculate the gradients $\nabla\mathcal{L}_{\mathcal{B}}(\Theta)$. Since feedforward neural networks (Def. 2.2.1) are compositions of functions, calculating the gradients requires the chain rule. An efficient application of this chain rule (using reverse-mode automatic differentiation) is *backpropagation*, which can be attributed to Linnainmaa (1970), see for example (Schmidhuber, 2015). The following chapter will focus on an extension called *double backpropagation*. The 'standard case' of backpropagation is furthermore derived as a special case.

Chapter 3

The Mathematics of Double Backpropagation

In this chapter, a description of double backpropagation is provided in a coordinate-free Hilbert space setting. An optimized double backpropagation algorithm for a special case is derived. Furthermore, a 'pseudo-smoothing' effect on the loss landscape of ReLU networks using double backpropagation is shown empirically.

This chapter is based on the following article:

> Christian Etmann. A Closer Look at Double Backpropagation. Manuscript submitted for publication. Preprint available at arXiv:1906.06637, 2019

3.1 Introduction

Lately, an increasing number of works have suggested using penalty terms involving derivatives with respect to the neural network input. So far, no valid and general description of the backpropagation procedure for these cases exists. While (Drucker and Le Cun, 1992) derive the double backpropagation formulas for a multilayer perceptron with one hidden layer only, (Sokolić et al., 2017) provide only a high-level description for ReLU (Nair and Hinton, 2010) networks. While automatic differentiation (AD) methods have made the calculation of the error terms and their respective weight gradients trivial to implement, they do not lend themselves to providing any theoretical insights. However, as we will show here, the specific choice of architecture and activation function can have a large impact on the optimization, for which a precise understanding of the involved backpropagation is essential. Furthermore, as we will show here, one can improve both the training time and memory requirements of the involved training procedures over the naïve utilization of AD in many real-world scenarios.
While it is straightforward to derive the backpropagation terms of neural networks which do not encompass derivate-based regularization terms, the situation looks very different when these are included. This stems from an intricate interdepence of the various involved terms, which needs to be accounted for.

3.1.1 Contributions

We derive backpropagation rules for large classes of derivative-based regularization terms in the very general framework of Hilbert spaces, which covers everything from standard neural networks up to more theoretical networks in function spaces along the lines of (Bruna and Mallat, 2013; Wiatowski and Bölcskei, 2017). We thereby offer a different perspective on backpropagation, which is usually understood as an operation on a computational graph.
In neural network literature, the derivatives are most often given in coordinate-form for specific examples of layers, e.g. fully-connected layers. The coordinate-free view in Hilbert spaces offers a unifying view using Fréchet derivatives, that is readily applicable to a wide range of problems. For this, we view the linear portion of e.g. fully-connected, convolutional and locally-connected layers as specific instances of continuous, bilinear operations between the parameters and the activations and extend the standard theory of adjoints of continuous linear operators in real Hilbert spaces to continuous bilinear operators.

We furthermore analyze the runtimes of different variants of double backpropagation and are able to provide adapted algorithms for various scenarios depending on the exact setup, including a reduction by up to a third for certain Jacobian penalties. We additionally explore the induced loss landscapes of the common special case of (leaky) ReLU neural networks, which induces jump discontinuities. We demonstrate that batch optimization procedures can alleviate concerns about instabilities caused by these discontinuities.

3.1.2 Applications of Derivative-Based Loss Terms

Double backpropagation comes into play, whenever one uses derivative-based optimization on loss functions which contain derivatives with respect to the input of the network. There is a variety of applications and model types that employ losses of this type. One example is 'classical' double backpropagation (Drucker and Le Cun, 1992), where the loss for one feature-label-pair (x, y) and forward-mapping f is

$$\ell(f(x), y) + \lambda \cdot \|\nabla_x \ell(f(x), y)\|_2^2,$$

with loss function ℓ. The motivation behind this is that input features that are close to one another should result in similar losses. In other words, one enforces a small local Lipschitz constant of the model in the domain set. One possible application is robustification to adversarial attacks (Simon-Gabriel et al., 2019), which will also be applied in Chapter 4. Instead of the loss, one may also penalize derivatives of logits or class predictions. In (Sokolić et al., 2017), the penalty term takes the form $\|J_f\|_F^2$, the squared Frobenius norm of the Jacobian of the output with respect to the input. Through this penalty term, one can effectively enlargen the model's margin in order to improve its generalization. Another instance of this type of penalty is found in contractive autoencoders (Rifai et al., 2011), where the Jacobian is calculated on the encoder's output, which is intended to assign similar codes to similar inputs. If one chooses the spectral norm instead of the Frobenius norm, one idea is to instead use $\|J_f v\|_2^2$, where v is a random unit vector. This is equivalent to one iteration of the power method, as proposed e.g. in (Anil et al., 2018). For applications where a ground-truth function to be approximated is known (e.g. model compression), Sobolev training (Czarnecki et al., 2017) aims to make the model close to the ground-truth in higher-order Sobolev norms,

which entails the input's derivatives.
Flow-based generative models like normalizing flows (Rezende and Mohamed, 2015), GLOW (Kingma and Dhariwal, 2018), FFJORD (Grathwohl et al., 2018) and invertible residual networks (Behrmann et al., 2019) generate a point x via $x = f^{-1}(z)$ by sampling z from a simple base distribution. Here, f can be a neural network. These models seek to maximize the data-likelihood, resulting in a loss function which contains the log-determinant of the Jacobian J_f, for whose evaluation various strategies exist.
Another instance of generative models requiring double backpropagation are certain types of generative adversarial networks (GANs) like (Roth et al., 2017), which enforce convergence through gradient-based penalty terms. In a similar vein, (Gulrajani et al., 2017) softly enforce 1-Lipschitzness of the critic of Wasserstein-GANs (Arjovsky et al., 2017), an idea which is picked up by (Lunz et al., 2018) for learning a regularization functional for inverse problems.

3.2 Mathematical Preliminaries

We introduce continuous, bilinear operators as a very general, yet simple tool for defining the affine linear portion of many different layer types. This encompasses dense, convolutional, locally-connected layers and invertible down-sampling. If we take a dense layer as an example, then the pre-activation $Wx + b$ with matrix W and bias b contains an expression that is linear *both* in x and in W. We can thus write $K(W, x) = Wx$ and realize that K is a bilinear operator.
Similarly, for convolutional layers we have $K(w, x) = w \circledast x$ with the multi-channel convolution operator $\circledast$. A typical example for image data would be $x \in \mathbb{R}^{256\times256\times3}$ and $w \in \mathbb{R}^{5\times5\times16\times3}$, which represents the convolution of 256-by-256 RGB image with a 5-by-5 kernel onto 16 feature maps. The theoretical setting allows us to work directly in these spaces, *without* reordering the entries into column vectors and representing the Fréchet derivatives as Jacobians.

3.2.1 Properties of Bilinear Operators

In the following, let $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{P}$ always be Hilbert spaces over the field of real numbers. Let $A : \mathcal{X} \to \mathcal{Y}$ be a continuous linear operator. We denote by A^* the adjoint of A, i.e. the (unique) continuous linear operator $A^* : \mathcal{Y} \to \mathcal{X}$ for which

$$\langle Ax, y\rangle_{\mathcal{Y}} = \langle x, A^*y\rangle_{\mathcal{X}}$$

for all $(x, y) \in (\mathcal{X} \times \mathcal{Y})$, where the $\langle \bullet, \bullet \rangle$ signify the respective inner products. If $\mathcal{X} = \mathbb{R}^n$ and $\mathcal{Y} = \mathbb{R}^m$, then $A \in \mathbb{R}^{m\times n}$ and its adjoint is just the transposed matrix A^T (up to isomorphism).

We now extend the concept of an adjoint of a *linear* continuous operator on real Hilbert spaces to *bilinear* continuous operators on Hilbert spaces.

Definition 3.2.1. *Let*

$$\begin{aligned} K : \mathcal{P} \times \mathcal{X} &\to \mathcal{Y} \\ (\theta, x) &\mapsto K(\theta, x) \end{aligned}$$

be a bilinear, continuous operator between real Hilbert spaces $\mathcal{P}, \mathcal{X}, \mathcal{Y}$*, such that*

$$\begin{aligned} K(\theta, \bullet) : \mathcal{X} &\to \mathcal{Y} \\ x &\mapsto K(\theta, x) \end{aligned}$$

and

$$\begin{aligned} K(\bullet, x) : \mathcal{P} &\to \mathcal{Y} \\ \theta &\mapsto K(\theta, x) \end{aligned}$$

are continuous linear operators for some fixed values of θ *and* x *respectively. Let* $K^T(\theta, \bullet)$ *be the adjoint of* $K(\theta, \bullet)$ *and* $K^\square(x, \bullet)$ *be the adjoint of* $K(\bullet, x)$*. These are* linear *operators. We then define the* bivariate *operators*

$$\begin{aligned} K^T : \mathcal{P} \times \mathcal{Y} &\to \mathcal{X} \\ (\theta, y) &\mapsto K^T(\theta, \bullet) \cdot y \end{aligned}$$

and

$$\begin{aligned} K^\square : \mathcal{X} \times \mathcal{Y} &\to \mathcal{P} \\ (x, y) &\mapsto K^\square(x, \bullet) \cdot y, \end{aligned}$$

which we call adjoint operators *of* K*.*

Note that the order of arguments in the definition of the adjoint operators above is arbitrary. One could analogously define two more adjoints, where the order of arguments is switched. This means that while the adjoint of a linear operator is unique, the two adjoints (as defined above) of a bilinear operator are only unique up to the order of arguments. In this thesis, we adopt the convention that for a bilinear, continuous operator $K : \mathcal{P} \times \mathcal{X} \to \mathcal{Y}$, its adjoints' order of arguments (from left to right) is $\mathcal{P} \to \mathcal{X} \to \mathcal{Y}$. This is of particular importance when we later define adjoint operators of K^T and $K^\square$.

For convenience, we will call K^T the *adjoint of* $K(\theta, x)$ *in* x and $K^\square$ the *adjoint of* $K(\theta, x)$ *in* θ. While this is a slight abuse of notation, it makes discussing the relationship among these objects much simpler.

Corollary 3.2.1. *According to Definition 3.2.1,*

$$\langle K(\theta, x), y \rangle_{\mathcal{Y}} = \langle x, K^T(\theta, y) \rangle_{\mathcal{X}}$$

and

$$\langle K(\theta, x), y \rangle_{\mathcal{Y}} = \langle \theta, K^\square(x, y) \rangle_{\mathcal{P}}$$

for all $(x, y, \theta) \in \mathcal{X} \times \mathcal{Y} \times \mathcal{P}$*.*

Remark 3.2.2. *It follows that* K *is the adjoint of* $K^T(\theta, y)$ *in* y *as well as the adjoint of* $K^\square(x, y)$ *in* x*.*

The author of this thesis is not aware of any other work, where two adjoint operators in the above sense are defined. However, Arens (1951) proposes a similar framework for bilinear operators between *normed spaces*, which is presented in detail in Appendix A. Therein, a (single) bilinear operator is constructed between dual and primal spaces, which the author calls *the* adjoint. The repeated application of this procedure leads to the same (two) adjoint operators as defined above, if all involved spaces are *Hilbert spaces*

and one identifies the (bi-)dual and primal spaces via the Riesz representation theorem. The restriction to Hilbert spaces allows for a great simplification of the involved theory and the derivation of very useful properties.

Example 3.2.3. *The most basic example for the above setup is that of matrix-vector-multiplication in finite-dimensional real vector spaces, equipped with the respective standard inner products. This can be phrased as bilinear, continuous operator* $K : \mathbb{R}^{m\times n} \times \mathbb{R}^n \to \mathbb{R}^m$ *given by* $K(W, x) = Wx$*. Then clearly,* $K^T(W, y) = W^T y$*, because* $\langle Wx, y\rangle_{\mathbb{R}^m} = \langle x, W^T y\rangle_{\mathbb{R}^n}$*. In order to determine* $K^\square$*, it needs to solve the equation* $\langle K(W, x), y\rangle_{\mathbb{R}^m} = \langle W, K^\square(x, y)\rangle_{\mathbb{R}^{m\times n}}$*, where*

$$\langle W, V\rangle_{\mathbb{R}^{m\times n}} = \sum_i^m \sum_j^n W_{ij}V_{ij} = \operatorname{trace}(W^T V)$$

is the standard inner product in matrix space. A simple calculation shows that

$$\langle Wx, y\rangle_{\mathbb{R}^m} = \sum_i^m \sum_j^n W_{ij}x_j y_i = \langle W, K^\square(x, y)\rangle_{\mathbb{R}^{m\times n}}$$

such that $K^\square(x, y) = yx^T$*, the outer product between* y *and* x*.*

Remark 3.2.4. *In the context of neural networks, where* $K(\theta, x)$ *represents the application of a 'weight'* θ *to a 'feature'* x*, we call* K^T *the* transposed operator of K*. This is in accordance not only with the nomenclature for matrix-vector multiplication, but also with the convention in other publications which call the respective transposed operator of the convolution the 'transposed convolution'. We further call* $K^\square$ *the* weight-adjoint operator *of* K*.*

The author is not aware of a name for the weight-adjoint operator in the literature. In Tensorflow (Abadi et al., 2016) for example, the weight-adjoint of the 2D convolution operator is called `tf.nn.conv2d_backprop_filter` due to its role in backpropagation, as will be elaborated upon in Section 3.4.

In order for the following proofs to work, two lemmata are required.

Lemma 3.2.5. *Let* U, V, W *be real Hilbert spaces. Let further* $A : U \times \mathcal{V} \to \mathcal{W}$ *and* $B : U \times \mathcal{V} \to \mathcal{W}$ *be continuous bilinear operators. Then* $A = B$ *if and only if*

$$\langle A(u, v), w\rangle_{\mathcal{W}} = \langle B(u, v), w\rangle_{\mathcal{W}}$$

for all $(u, v, w) \in U \times \mathcal{V} \times \mathcal{W}$*.*

Proof. The '$\Rightarrow$'-direction follows immediately. We now prove the converse direction. Let

$$\langle A(u, v), w\rangle_{\mathcal{W}} = \langle B(u, v), w\rangle_{\mathcal{W}}$$

for all $(u, v, w) \in U \times \mathcal{V} \times \mathcal{W}$. Then

$$\langle [A - B](u, v), w\rangle_{\mathcal{W}} = 0,$$

in particular for $w = [A - B](u, v)$, so that

$$\langle [A - B](u, v), [A - B](u, v)\rangle_{\mathcal{W}} = 0,$$

which means that (due to the definiteness of inner products),

$$[A - B]\,(u, v) = 0$$

for all $u \in U$ and $v \in \mathcal{V}$. In other words, $A - B$ is a null operator, which proves $A = B$. □

Lemma 3.2.6. *K^T and $K^\square$ are continuous, bilinear operators.*

Proof. We now show the proof for the bilinearity and continuity of K^T only. The respective proofs for the bilinearity and continuity of $K^\square$ are completely analogous.

We begin by showing the bilinearity. We already know that $K^T(\theta, y)$ is linear in y, since $K^T(\theta, \bullet)$ is defined to be the adjoint operator of $K(\theta, \cdot)$, and thus a linear operator itself. It remains to show that $K^T(\theta, y)$ is furthermore linear in θ:

$$\begin{aligned}\langle x, K^T(\theta + \psi, y)\rangle_\mathcal{X} &= \langle K(\theta + \psi, x), y\rangle_\mathcal{Y} \\ &= \langle K(\theta, x), y\rangle_\mathcal{Y} + \langle K(\psi, x), y\rangle_\mathcal{Y} \\ &= \langle x, K^T(\theta, y)\rangle_\mathcal{X} + \langle x, K^T(\psi, y)\rangle_\mathcal{X}\end{aligned}$$

It follows that

$$\langle x, K^T(\theta + \psi, y) - \big[K^T(\theta, y) + K^T(\psi, y)\big]\rangle_\mathcal{X} = 0$$

for all x, y, θ, ψ, which, analogously to the proof of Lemma 3.2.5, shows that

$$K^T(\theta + \psi, y) = K^T(\theta, y) + K^T(\psi, y),$$

meaning that K^T is linear in the first argument and thus a bilinear operator.

We now show that K^T is a continuous operator by demonstrating that there is a $c > 0$ such that

$$\|K^T(\theta, y)\|_\mathcal{X} \leq c\|\theta\|_\mathcal{P}\|y\|_\mathcal{Y}$$

for all $(\theta, y) \in \mathcal{P} \times \mathcal{Y}$ which is sufficient to show continuity (cf. Rudin, 1991, p. 55). A central tool is the *uniform boundedness principle* (UBP, also known as the *Banach-Steinhaus-Theorem*), which states that if

$$\sup_{A \in F} \|Ax\|_\mathcal{Y} < \infty$$

for all $x \in \mathcal{X}$ for a subset F of continuous operators from $\mathcal{X}$ to $\mathcal{Y}$, then

$$\sup_{A \in F} \|A\|_{B(\mathcal{X},\mathcal{Y})} < \infty$$

holds, where $\|\bullet\|_{B(\mathcal{X},\mathcal{Y})}$ signifies the operator norm for linear operators from $\mathcal{X}$ to $\mathcal{Y}$ (cf. Simon, 2015, p. 398). Since $K(\bullet, x)$ is a continuous operator, $\|K(\theta, x)\|_\mathcal{Y} \leq c\|\theta\|_\mathcal{P}$ holds for all $\theta \in \mathcal{P}$ and some $c > 0$, it follows that

$$\sup_\theta \frac{1}{\|\theta\|_\mathcal{P}} \|K(\theta, \bullet) \cdot x\|_\mathcal{Y} < \infty$$

for all $x \in \mathcal{X}$. Here, $F = \{\frac{1}{\|\theta\|_{\mathcal{P}}} K(\theta, \bullet) : \theta \in \mathcal{P}\}$ is the family of linear operators in the UBP, which now holds. Hence, due to

$$\sup_{\theta} \frac{1}{\|\theta\|_{\mathcal{P}}} \|K(\theta, \bullet)\|_{B(\mathcal{X},\mathcal{Y})} < \infty,$$

we see that furthermore

$$\sup_{\theta} \frac{1}{\|\theta\|_{\mathcal{P}}} \|K^T(\theta, \bullet)\|_{B(\mathcal{Y},\mathcal{X})} < \infty,$$

because the operator norms of linear operators and their adjoints coincide. The above implies that $\sup_\theta \frac{1}{\|\theta\|_{\mathcal{P}}} \|K^T(\theta, y)\|_{\mathcal{X}} \leq c\|y\|_{\mathcal{Y}}$ for some $c > 0$ and for any $y \in \mathcal{Y}$. In particular, for any $\theta \in \mathcal{P}$, it holds that

$$\|K^T(\theta, y)\|_{\mathcal{X}} \leq c\|\theta\|_{\mathcal{P}}\|y\|_{\mathcal{Y}},$$

which implies continuity of K^T. □

Since K^T and $K^\square$ are bilinear operators, if they are continuous in both arguments, there exist two adjoint operators for each of them as well (if the order of arguments is fixed). Two of these four operators were already identified in Remark 3.2.2. We clarify their connection with the following theorem, which will be essential for the calculation of the double backpropagation rules.

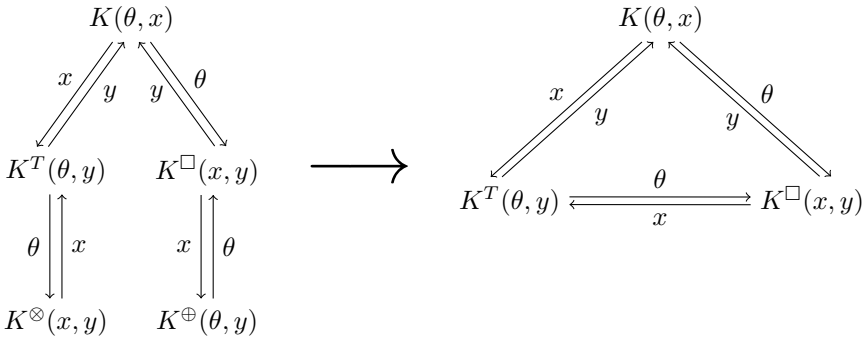


Figure 3.1: Visualization of Theorem 3.2.7 and the setup of its proof. An arrow indicates in which variable the adjoint is taken.

Theorem 3.2.7 (Commutativity of adjoints)**.** *Let K^T be the adjoint of $K(\theta, x)$ in x and let $K^\square$ be the adjoint of $K(\theta, x)$ in θ. Then the adjoint of $K^T(\theta, y)$ in θ exists and coincides with $K^\square$ and the adjoint of $K^\square(x, y)$ in x exists and coincides with K^T.*

Proof. According to Lemma 3.2.6, K^T and $K^\square$ are continuous and bilinear, which implies the existence of their adjoints. Let $K^\otimes : (x, y) \mapsto K^T(\bullet, y)^* \cdot x$ be the adjoint of $K^T(\theta, y)$ in θ and let $K^\oplus : (\theta, y) \mapsto K^\square(\bullet, y)^* \cdot \theta$ be the adjoint of $K^\square(x, y)$ in x. Then

$$\begin{aligned}\langle K(\theta, x), y\rangle_{\mathcal{Y}} &= \langle \theta, K^\square(x, y)\rangle_{\mathcal{P}} = \langle K^\oplus(\theta, y), x\rangle_{\mathcal{X}} \\ &= \langle x, K^T(\theta, y)\rangle_{\mathcal{X}} = \langle K^\otimes(x, y), \theta\rangle_{\mathcal{P}}\end{aligned}$$

for all $(\theta, x, y) \in \mathcal{P} \times \mathcal{X} \times \mathcal{Y}$. From Lemma 3.2.5 and the fact that inner products of spaces over the field of real numbers are symmetric, we can infer that $K^T = K^{\oplus}$ and $K^{\square} = K^{\otimes}$. The proof is visualized in Figure 3.1. □

The above theorem indicates that the weight-adjoint of a transposed operator is the same as the weight-adjoint for its primal operator. While this is easy to see for special cases like matrix-vector multiplication or convolutions, this theorem guarantees this for every bilinear, continuous operator.

Remark 3.2.8. *Theorem 3.2.7 immediately generalizes to multilinear operators. For real Hilbert spaces $\mathcal{X}_1, \ldots, \mathcal{X}_{n+1}$, which we uniquely identify by their indices*[1]*, let*

$$K : \mathcal{X}_1 \times \cdots \times \mathcal{X}_n \to \mathcal{X}_{n+1}$$

be a multilinear, continuous operator and let

$$K^{\mathcal{X}_i} : \mathcal{X}_1 \times \cdots \times \mathcal{X}_{i-1} \times \mathcal{X}_{i+1} \times \cdots \times \mathcal{X}_{n+1} \to \mathcal{X}_i$$

be the adjoint of K in its i-th argument, which is a multilinear operator. If $K^{\mathcal{X}_i}$ is continuous, then

$$\left(K^{\mathcal{X}_i}\right)^{\mathcal{X}_j} = K^{\mathcal{X}_j}$$

for all j. This is because one can view all other arguments as fixed and thus reduce this multilinear operator to a bilinear operator. Hence, only the final argument with respect to which one takes the adjoint determines the resulting operator.

3.2.2 Fréchet Calculus

Here, we will provide short definitions and theorems for derivatives in Hilbert spaces, which are just generalizations of familiar terms in $\mathbb{R}$. We will always assume the involved spaces to be vector spaces over the field $\mathbb{R}$. The following definitions and theorems are standard and can e.g. be found in even more generality in (Schechter, 1996).

Definition 3.2.2 (Fréchet derivative)**.** *Let $\mathcal{X}$ and $\mathcal{Y}$ be real Hilbert spaces and let $U \subset \mathcal{X}$ be an open subset. The function $f : U \to \mathcal{Y}$ is called* Fréchet differentiable in $x \in U$ *if there is a continuous linear operator $\frac{\mathrm{d}\, f(x)}{\mathrm{d}\, x} : \mathcal{X} \to \mathcal{Y}$ such that*

$$\lim_{\|h\|_{\mathcal{X}} \to 0} \frac{\|f(x+h) - f(x) - \frac{\mathrm{d}\, f(x)}{\mathrm{d}\, x} \cdot h\|_{\mathcal{Y}}}{\|h\|_{\mathcal{X}}} = 0.$$

Then $\frac{\mathrm{d}\, f(x)}{\mathrm{d}\, x}$ is called the Fréchet derivative of f in x.

Remark 3.2.9. *When it is clear from context that $y = f(x)$, we will simply write $\frac{\mathrm{d}\, y}{\mathrm{d}\, x}$ for $\frac{\mathrm{d}\, f(x)}{\mathrm{d}\, x}$.*

Definition 3.2.3 (Gradient)**.** *Let $\mathcal{X}$ be a real Hilbert space and let $U \subset \mathcal{X}$ be an open subset. Let further $f : \mathcal{X} \to \mathbb{R}$ be Fréchet differentiable in $x \in U$. We call the vector $v \in U$ for which*

$$\frac{\mathrm{d}\, f(x)}{\mathrm{d}\, x} \cdot h = \langle v, h \rangle_{\mathcal{X}}$$

for all $h \in U$ the gradient of f with respect to x *and write $\nabla_x f(x) := v$.*

[1] This is to prevent ambiguity if $\mathcal{X}_i = \mathcal{X}_j$ for some (i, j).

Remark 3.2.10. *The existence and uniqueness is guaranteed by Riesz' representation theorem.*

Theorem 3.2.11 (Sum rule)**.** *Let $\mathcal{X}$ and $\mathcal{Y}$ be real Hilbert spaces and let $U \subset \mathcal{X}$ be an open subset. Let the two functions $f : U \to \mathcal{Y}$ and $g : U \to \mathcal{Y}$ be Fréchet differentiable in $x \in U$, then $f + g$ is Fréchet differentiable in x and*

$$\frac{\mathrm{d}(f+g)(x)}{\mathrm{d}\,x} = \frac{\mathrm{d}\,f(x)}{\mathrm{d}\,x} + \frac{\mathrm{d}\,g(x)}{\mathrm{d}\,x}$$

holds.

Another important theorem is the *generalized product rule* (cf. Lang, 1993, p. 336).

Theorem 3.2.12 (Generalized product rule)**.** *Let $\mathcal{X}$, $\mathcal{Y}_1$, $\mathcal{Y}_2$ and $\mathcal{Z}$ be real Hilbert spaces. Let $f : \mathcal{X} \to \mathcal{Y}_1$ and $g : \mathcal{X} \to \mathcal{Y}_2$ be Fréchet differentiable in $x \in U \subset \mathcal{X}$ (U open in $\mathcal{X}$) and let $B : \mathcal{Y}_1 \times \mathcal{Y}_2 \to \mathcal{Z}$ be a continuous bilinear operator. Then $B(f(\bullet), g(\bullet))$ is Fréchet differentiable in x and*

$$\begin{aligned}&\frac{\mathrm{d}\,B\left(f(x), g(x)\right)}{\mathrm{d}\,x} \cdot h \\ =&B\left(\frac{\mathrm{d}\,f(x)}{\mathrm{d}\,x} \cdot h, g(x)\right) + B\left(f(x), \frac{\mathrm{d}\,g(x)}{\mathrm{d}\,x} \cdot h\right)\end{aligned}$$

Theorem 3.2.13 (Chain rule)**.** *Let $\mathcal{X}, \mathcal{Y}$ and $\mathcal{Z}$ be real Hilbert spaces. Let $f : \mathcal{X} \to Y$ and $g : U \to \mathcal{Z}$ for some open set $U \subset \mathcal{Y}$. Let f be Fréchet differentiable in x and $y := f(x) \in U$. Let further g be differentiable in y. Then $g \circ f$ is well-defined in x and Fréchet differentiable in x with*

$$\frac{\mathrm{d}\,g(y)}{\mathrm{d}\,x} = \frac{\mathrm{d}\,g(y)}{\mathrm{d}\,y}\frac{\mathrm{d}\,y}{\mathrm{d}\,x}.$$

Applying the chain rule to Definition 3.2.3 of the gradient yields the following theorem.

Theorem 3.2.14 (Gradient chain rule)**.** *Let the assumptions from Theorem 3.2.13 hold, where additionally $\mathcal{Z} = \mathbb{R}$. Then*

$$\nabla_x g(y) = \left(\frac{\mathrm{d}\,y}{\mathrm{d}\,x}\right)^* \cdot \nabla_y g(y).$$

In finite-dimensional real vector spaces, the Fréchet derivative can be represented as a matrix, the *Jacobian*, given continuity of the partial derivatives. The gradient definition in 3.2.3 leads to the familiar column vector consisting of partial derivatives. In this chapter, we will rigorously distinguish between a matrix W, which defines a linear map $f : x \mapsto Wx$, and the linear map f itself. Occasionally, the fact that they are isomorphic to one another is emphasized by the expression $W \cong f$.

3.3 Neural Network Model

3.3.1 Forward Pass

In the following, we will consider an L-layer network

$$\left.\begin{aligned} z_j &= K_j(\theta_j, x_{j-1}) + b_j \\ x_j &= \varphi_j(z_j) \end{aligned}\right\} \text{ for } j = 1, \dots, L,$$

where x_0 is the input to the network. Here, $K_j : \mathcal{P}_j \times \mathcal{X}_{j-1} \to \mathcal{X}_j$ is a continuous bilinear operator between Hilbert spaces, $\theta_j \in \mathcal{P}_j$ and $b_j \in \mathcal{X}_j$ are the j-th layer's parameters, $z_j \in \mathcal{X}_j$ and $x_j \in \mathcal{X}_j$ its respective pre-activation and activation, $\varphi_j : \mathcal{X}_j \to \mathcal{X}_j$ the activation function. Here we concentrate on networks with $\mathcal{X}_L = \mathbb{R}^C$ (with standard inner product), e.g. classifiers with $\varphi_L = \text{softmax}$, but the results naturally extend to other types of neural networks. Note that we use superscripts (x^j) to denote coordinates of vectors in this chapter.

3.3.2 Dealing with Nonlinearities

We will further assume φ_j (for $j < L$) to be a nonlinearity that is applied coordinate-wise (Definition 2.2.6), like ReLU or tanh. Assuming some coordinate representation would of course defeat the purpose of finding a coordinate-free (double) backpropagation scheme. This is why we have to find a more general characterization of these types of functions that still retains their simplicity.

Definition 3.3.1 (Coordinate-wise Fréchet differentiable)**.** *Let $\mathcal{X}$ be a real Hilbert space and $U \subset \mathcal{X}$ be an open subset. If there exists a symmetric, continuous, bilinear operator $M : \mathcal{X} \times \mathcal{X} \to \mathcal{X}$ such that $\varphi : U \to \mathcal{X}$ is Fréchet differentiable in $x \in U$ with*

$$\frac{\mathrm{d}\,\varphi(x)}{\mathrm{d}\,x} \cdot h = M(\varphi'(x), h)$$

for some function $\varphi' : U \to \mathcal{X}$ for all $h \in \mathcal{X}$, we call g coordinate-wise Fréchet differentiable *in x. If φ' is itself coordinate-wise Fréchet differentiable in x, we call g coordinate-wise twice Fréchet differentiable in x with second derivative φ''.*

The motivation behind this technical definition is the fact that for the coordinate-wise application of functions like $g = \tanh$, the Jacobian is a diagonal matrix, such that

$$\frac{\mathrm{d}\tanh(x)}{\mathrm{d}\,x} \cdot v = \tanh'(x) \odot v$$

with $M : (x, y) \mapsto x \odot y$ denoting the coordinate-wise multiplication. When appropriate, we will use the abbreviations

$$\Phi'(x) := \frac{\mathrm{d}\,\varphi(x)}{\mathrm{d}\,x} \text{ and } \Phi''(x) := \frac{\mathrm{d}\,\varphi'(x)}{\mathrm{d}\,x}, \tag{3.1}$$

which allows us to easily switch between viewing these derivatives as either linear or bilinear maps. The latter will later be essential in order to be able to apply the generalized product rule 3.2.12. The following lemma and corollaries show that these functions are self-adjoint.

Lemma 3.3.1. *Let $M : \mathcal{X} \times \mathcal{X} \to \mathcal{X}$ be a symmetric, continuous, bilinear operator. Then $M = M^{\square} = M^T$.*

Proof. First of, since $M(x, \bullet) = M(\bullet, x)$ for all x, we have

$$M^T(x, \bullet) = (M(x, \bullet))^* = (M(\bullet, x))^* = M^{\square}(x, \bullet)$$

for all $x \in \mathcal{X}$, such that $M^{\square} = M^T$. Moreover, applying Theorem 3.2.7 yields

$$M^T(x, y) = M^{\square}(\bullet, y)^* \cdot x = [M(y, \bullet)^*]^* \cdot x = M(y, x) = M(x, y)$$

for all $x, y \in \mathcal{X}$, which proves that $M = M^T = M^{\square}$. □

Corollary 3.3.2. *Let $A := M(a, \bullet)$, where $M : \mathcal{X} \times \mathcal{X} \to \mathcal{X}$ is a symmetric, continuous, bilinear operator and $a \in \mathcal{X}$. Then A is self-adjoint.*

Proof. For all $x, y \in \mathcal{X}$,

$$\langle Ax, y\rangle_{\mathcal{X}} = \langle M(a, x), y\rangle_{\mathcal{X}} = \langle x, M(a, y)\rangle_{\mathcal{X}}$$

according to Lemma 3.3.1. Furthermore,

$$\langle Ax, y\rangle_{\mathcal{X}} = \langle x, A^* y\rangle_{\mathcal{X}},$$

so that $\langle x, A^* y\rangle_{\mathcal{X}} = \langle x, M(a, y)\rangle_{\mathcal{X}}$ for all $x, y \in \mathcal{X}$, which means that $A^* = M(a, \bullet) = A$ due to Lemma 3.2.5. □

This corollary then immediately implies the self-adjointness of coordinate-wise Fréchet derivatives.

Corollary 3.3.3. *If g is twice coordinate-wise Fréchet differentiable with derivatives $\Phi'(x)$ and $\Phi''(x)$ for some $x \in \mathcal{X}$ as defined in* (3.1)*, then $\Phi'(x)$ and $\Phi''(x)$ are self-adjoint continuous linear operators.*

The restriction to coordinate-wise nonlinearities (except for the final layer) allows for a great simplification of the utilized theory, while representing the vast majority of real-world neural networks. In particular, the product rule can be readily applied. If on the other hand, one were to use general Fréchet differentiable activation functions, the used higher-order derivatives would need to be calculated on spaces of Fréchet derivatives, which demands a much more involved derivation of the backpropagation rules.

While we presume all activation functions up to the final layer to be coordinate-wise, the final layer requires separate consideration. In the following, we will assume that the output layer's activation function φ_L *also* has a self-adjoint Fréchet derivative. This is certainly true for $\varphi_L = \mathrm{softmax}$, for which a standard calculation shows that

$$\frac{\mathrm{d}\, x_L}{\mathrm{d}\, z_L} \cong \mathrm{diag}(x_L) - x_L x_L^T, \tag{3.2}$$

meaning that $\Phi'_L(x_L)$ is self-adjoint as well.

3.3.3 A Note on Differentiability

For smooth activation functions, the sum, product and chain rule guarantee the differentiability of the neural network model everywhere. The commonly used (leaky) ReLU activation functions however – locally linear functions with a non-differentiability at 0 – induce non-differentiable behavior throughout the network. Moreover, this can also be the case for the penalty function p. To be precise, ReLU networks partition the input space into polytopes in which the logit layer z_L is affine in x_0 (Raghu et al., 2017). The points of non-differentiability lie on the boundary of these polytopes. There are extensions of (Fréchet) differentiability such as different types of *subgradients and -differentials* that may apply to points on the boundary. See (Mordukhovich, 2006) for an overview of the many different concepts of subgradients. A full theoretical subdifferential treatment however is far out of the scope of this thesis, because the differentiation rules from Section 3.2.2 do not generally hold anymore. Since the boundaries of the polytopes form a null set, we do not consider these points here and assume that every derivative exists.
Most neural network frameworks still define a derivative even when a function is not classically differentiable. These are unfortunately not coherent with the differentiation rules. A recent approach by Bolte and Pauwels (2019) makes this type of differentiation precise, but is not considered in this thesis.
Whether a minimum exists for the double backpropagation loss may also depend on the applied notion of differentiability for these classically non-differentiable points. Even in the case of a coercive loss function (e.g. when using a weight decay penalty), the loss is not necessarily lower semi-continuous (see section 3.7).

3.4 Deriving Double Backpropagation Rules

Double backpropagation comes into play, whenever the loss function to be minimized contains a derivative of a function with respect to x_0. As we optimize our loss using first-order methods, our ultimate goal is to determine the gradients $\nabla_{\theta_j}\mathcal{R} \in \mathcal{P}_j$ and $\nabla_{b_j}\mathcal{R} \in \mathcal{X}_j$, where $\mathcal{R}$ denotes an expression that depends on a derivative with respect to x_0 (usually a regularization or penalty term).

3.4.1 Penalty Terms

We consider penalty terms $\mathcal{R}$ (or sums thereof) that can be written in the form

$$\mathcal{R} := p\left(\left(\frac{\mathrm{d}\, x_L}{\mathrm{d}\, x_0}\right)^* \cdot v\right), \tag{3.3}$$

where $p : \mathcal{X}_0 \to \mathbb{R}$ is differentiable almost everywhere and not locally constant and v may or may not depend on $x_L, \ldots, x_0$. The exact form of the penalty is thus determined by p and v. In the following, we will offer some examples.

Classical double backpropagation

In classical double backpropagation, we apply a penalty $\|\nabla_{x_0}\mathcal{L}\|^2_{\mathcal{X}_0}$, where $\mathcal{L} := \ell(x_L, y)$ is the network's loss (with loss function ℓ). Here $y \in \mathbb{R}^C = \mathcal{X}_L$, e.g. a one-hot encoded

label vector. By applying the gradient chain rule (Theorem 3.2.14), this yields

$$\begin{aligned}&\|\nabla_{x_0}\mathcal{L}\|^2_{\mathcal{X}_0}\\ =&\left\|\left(\frac{\mathrm{d}\,x_L}{\mathrm{d}\,x_0}\right)^* \cdot \nabla_{x_L}\ell(x_L, y)\right\|^2_{\mathcal{X}_0},\end{aligned}$$

so that $p : u \mapsto \|u\|^2_{\mathcal{X}_0}$ and $v = \nabla_{x_L}\mathcal{L}$. In the special case of the squared euclidean error

$$\ell(x_L, y) = \|x_L - y\|_2^2, \tag{3.4}$$

this results in $v = 2(x_L - y)$. When using the negative log-likelihood

$$\ell(x_L, y) = -\sum_{i=1}^{C} y^i \cdot \log\left(x_L^i\right),$$

we get $v = -y \oslash x_L$ (with $\oslash$ denoting the component-wise (*Hadamard*) division). These constitute cases where v depends on x_L and thus on all x_j with $j \leq L$.

Penalties on gradients of output nodes

Another general type of penalty is on derivatives of output nodes with respect to the input. For example, $\|\nabla_{x_0} x_L^i\|_2^2$, a squared euclidean norm penalty on the gradient of the i-th output node with respect to the input, can be represented via $v = e^{[i]}$ in (3.3), where $e^{[i]}$ denotes the i-th standard unit vector.
We can immediately obtain formulas for the (squared) Frobenius norm of the Jacobian $J_f \cong \frac{\mathrm{d}\,x_L}{\mathrm{d}\,x_0}$ by realizing that

$$\|J_f\|_F^2 = \sum_{i=1}^{C} \|\nabla_{x_0} x_L^i\|_2^2,$$

which entails C penalties of the form (3.3). This however naturally increases the time complexity of the double backpropagation roughly by a factor of C. We will later (Section 3.6) present an algorithm, with which the runtime may be reduced by up to a third, depending on the used activation functions.
If the penalties are applied on the logits (z_L) instead of the softmax-outputs (x_L), one can simply model this via $\varphi_L = \mathrm{id}$, the identity function.

Operator norm penalties

Section 3.1.2 briefly touched upon how randomized penalties can be employed in the calculation of the spectral norm of the Jacobian (more generally: operator norms of the Fréchet derivative). The operator norm can be written as

$$\begin{aligned}\left\|\frac{\mathrm{d}\,x_L}{\mathrm{d}\,x_0}\right\|_{B(\mathcal{X}_0,\mathcal{X}_L)} &:= \sup_{\|u\|_{\mathcal{X}_0}=1} \left\|\frac{\mathrm{d}\,x_L}{\mathrm{d}\,x_0} \cdot u\right\|_2\\ &= \sup_{\|v\|_2=1} \left\|\left(\frac{\mathrm{d}\,x_L}{\mathrm{d}\,x_0}\right)^* \cdot v\right\|_{\mathcal{X}_0},\end{aligned} \tag{3.5}$$

where we used the fact that the operator norms of primal and adjoint continuous operators in Hilbert spaces coincide (Rudin, 1991). By sampling $\tilde{v}$ from a normal distribution and setting $v = \tilde{v}/\|\tilde{v}\|_2$, one samples v almost surely uniformly from the unit sphere

Algorithm 1 Calculation of the penalty term

Initialize $\xi_L = v$
for $j \leftarrow L$ **to** 1 **do**
  $\zeta_j = \Phi_j'(z_j) \cdot \xi_j$
  $\xi_{j-1} = K_j^T(\theta_j, \zeta_j)$
end for
Output: $\mathcal{R} = p(\xi_0)$

$\{v : \|v\|_2 = 1\}$ (Muller, 1959). With $p = \|\bullet\|_{\mathcal{X}_0}$, we thus obtain a lower bound of the operator norm, which yields a penalty term of the form (3.3). This is equivalent to one power iteration. Better estimates of the optimal v in (3.5) are obtained by performing multiple power iterations.

3.4.2 Backward Pass: Calculating the Penalty Terms

In order to calculate $\mathcal{R}$ in the first place, we define

$$\begin{aligned}\xi_j &:= \left(\frac{\mathrm{d}\, x_L}{\mathrm{d}\, x_j}\right)^* \cdot v \\ \zeta_j &:= \left(\frac{\mathrm{d}\, x_L}{\mathrm{d}\, z_j}\right)^* \cdot v,\end{aligned}$$

which allows us to write $\xi_L = v$ and $\mathcal{R} = p(\xi_0)$. Given ξ_j, we can calculate ζ_j via

$$\begin{aligned}\zeta_j &= \left(\frac{\mathrm{d}\, x_L}{\mathrm{d}\, z_j}\right)^* \cdot v \\ &= \left(\frac{\mathrm{d}\, x_j}{\mathrm{d}\, z_j}\right)^* \cdot \left(\frac{\mathrm{d}\, x_L}{\mathrm{d}\, x_j}\right)^* \cdot v \\ &= (\Phi'(z_j))^* \cdot \xi_j \\ &= \Phi'(z_j) \cdot \xi_j,\end{aligned} \tag{3.6}$$

using the chain rule and the self-adjointness of $\Phi'(z_j)$. Given ζ_j, we can further calculate

$$\begin{aligned}\xi_{j-1} &= \left(\frac{\mathrm{d}\, x_L}{\mathrm{d}\, x_{j-1}}\right)^* \cdot v \\ &= \left(\frac{\mathrm{d}\, z_j}{\mathrm{d}\, x_{j-1}}\right)^* \cdot \left(\frac{\mathrm{d}\, x_L}{\mathrm{d}\, z_j}\right)^* \cdot v \\ &= (K_j(\theta_j, \bullet))^* \cdot \zeta_j \\ &= K_j^T(\theta_j, \bullet) \cdot \zeta_j \\ &= K_j^T(\theta_j, \zeta_j),\end{aligned} \tag{3.7}$$

where we applied the chain rule and used the fact that the adjoint of $K_j(\theta_j, u)$ in u is the transposed operator of K_j. In summary, the penalty is calculated via the recursion given in Algorithm 1.

Here, the difficulty in calculating $\nabla_{\theta_j}\mathcal{R}$ and $\nabla_{b_j}\mathcal{R}$ for all j becomes visible: While ξ_{j-1} depends directly on θ_j, it also depends on ζ_j, which itself directly depends on z_j, which

in turn depends on θ_j. Furthermore, ζ_j depends on ξ_j, which implicitly depends on θ_j as well, since it is a result of the backward pass. In other words, due to the edges from the upper half to the lower half of the graph, ζ_j depends on every variable except for $\xi_0, \dots, \xi_{j-1}, \zeta_1, \dots, \zeta_{j-1}, x_L, \mathcal{R}$ and $\mathcal{L}$. For the calculation of the weight-gradients, one hence has to untangle these complicated functional relationships. The complete interdependence of all involved variables is displayed in the dependency graph in Figure 3.2.

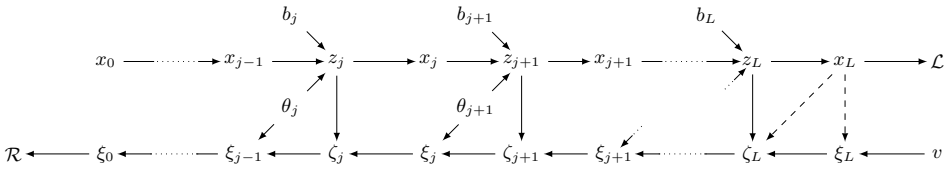


Figure 3.2: Dependency graph of the quantities in the derivative-regularized network according to Sections 3.3 and 3.4. An edge from node A to node B signifies B being a function of A. This implies that B is also a function of every node that A is a function of etc. The input nodes are x_0 and v, while the output nodes are $\mathcal{L}$ and $\mathcal{R}$. Dashed lines symbolize *possible* dependencies, which depends on the exact loss function used. See Section 3.5 for information about this.

3.4.3 Standard Backpropagation

We can easily recover the standard backpropagation rules (without any derivative-based penalty terms) for the loss $\ell(x_L, y)$ from the above setup by setting $v = \nabla_{x_L} \ell(x_L, y)$. Then

$$\begin{aligned}
\nabla_{\theta_j} \ell(x_L, y) &= \left(\frac{\mathrm{d}\, z_j}{\mathrm{d}\, \theta_j} \right)^* \cdot \nabla_{z_j} \ell(x_L, y) \\
&= \left(\frac{\mathrm{d}\, K_j(\theta_j, x_{j-1}) + b_j}{\mathrm{d}\, \theta_j} \right)^* \cdot \zeta_j \\
&= (K_j(\,\bullet\,, x_{j-1}))^* \cdot \zeta_j \\
&= K_j^{\square}(x_{j-1}, \zeta_j)
\end{aligned}$$

and

$$\begin{aligned}
\nabla_{b_j} \ell(x_L, y) &= \left(\frac{\mathrm{d}\, z_j}{\mathrm{d}\, b_j} \right)^* \cdot \nabla_{z_j} \ell(x_L, y) \\
&= \left(\frac{\mathrm{d}\, K_j(\theta_j, x_{j-1}) + b_j}{\mathrm{d}\, b_j} \right)^* \cdot \zeta_j \\
&= \mathrm{id}^* \cdot \zeta_j \\
&= \zeta_j,
\end{aligned}$$

which provide the well-known weight-gradients, that are needed for each iteration of a first-order optimization scheme of the network's loss, in the general framework of continuous bilinear operators.
This also demonstrates that one is able to 'reuse' the values ζ_i and ξ_i for standard backpropagation *and* for classical double backpropagation, unlike for all other penalty terms.

3.4.4 Backward-Backward Pass

Networks that require double backpropagation can be viewed as extended neural networks, where the forward pass (FP) and the backward pass (BP) are concatenated to form the forward pass of a neural network with twice the depth. Through this lens, double backpropagation is nothing but backpropagation through the extended network, where the gradients first pass through the BP of the original network, then the FP of the original network (which was already recognized in (Drucker and Le Cun, 1992)). We therefore call the procedures with which we calculate these gradients the *backward-backward pass* and the *forward-backward pass*.

Much like in standard backpropagation, our goal is to calculate $\nabla_{\theta_j}\mathcal{R}$ and $\nabla_{b_j}\mathcal{R}$, while keeping the dependency graph (Fig. 3.2) in mind. Due to

$$\begin{aligned}\nabla_{\theta_j}\mathcal{R} &= \left(\frac{\mathrm{d}\,\xi_{j-1}}{\mathrm{d}\,\theta_j}\right)^* \cdot \nabla_{\xi_{j-1}}\mathcal{R} \\ \nabla_{b_j}\mathcal{R} &= \left(\frac{\mathrm{d}\,\zeta_j}{\mathrm{d}\,b_j}\right)^* \cdot \nabla_{\zeta_j}\mathcal{R}\end{aligned} \tag{3.8}$$

we are interested in

$$\begin{aligned}q_j &:= \nabla_{\xi_j}\mathcal{R} \\ h_j &:= \nabla_{\zeta_j}\mathcal{R},\end{aligned}$$

for which propagation rules are needed. From equation (3.7) we can infer that

$$\frac{\mathrm{d}\,\xi_{j-1}}{\mathrm{d}\,\zeta_j} = \left(\frac{\mathrm{d}\,z_j}{\mathrm{d}\,x_{j-1}}\right)^*,$$

such that

$$\begin{aligned}h_j &= \nabla_{\zeta_j}\mathcal{R} \\ &= \left(\frac{\mathrm{d}\,\xi_{j-1}}{\mathrm{d}\,\zeta_j}\right)^* \cdot \nabla_{\xi_{j-1}}\mathcal{R} \\ &= K_j^T(\theta_j, \bullet)^* \cdot q_{j-1} \\ &= K_j(\theta_j, q_{j-1})\end{aligned}$$

holds. Moreover, according to equation (3.6)

$$\frac{\mathrm{d}\,\zeta_j}{\mathrm{d}\,\xi_j} = \left(\frac{\mathrm{d}\,x_j}{\mathrm{d}\,z_j}\right)^*,$$

with which

$$\begin{aligned}q_j &= \nabla_{\xi_j}\mathcal{R} \\ &= \left(\frac{\mathrm{d}\,\zeta_j}{\mathrm{d}\,\xi_j}\right)^* \cdot \nabla_{\zeta_j}\mathcal{R} \\ &= \Phi'(z_j)^* \cdot h_j \\ &= \Phi'(z_j) \cdot h_j\end{aligned}$$

follows. This results in the iteration scheme summarized in Algorithm 2.

At this point, we still cannot evaluate equations (3.8). As visible in Figure 3.2, the linear operators

$$\frac{\mathrm{d}\,\xi_{j-1}}{\mathrm{d}\,\theta_j}$$

Algorithm 2 Calculation of the backwards-backwards terms

Initialize $q_0 = \nabla_{\xi_0} p(\xi_0)$
for $j \leftarrow 1$ **to** L **do**
  $h_j = K_j(\theta_j, q_{j-1})$
  $q_j = \Phi'(z_j) \cdot h_j$
end for

and

$$\frac{\mathrm{d}\,\zeta_j}{\mathrm{d}\,b_j}$$

make us consider the functional relationship between the upper and lower half of the dependency graph. This happens through the forward-backward pass.

3.4.5 Fordward-Backward Pass

We continue to try to evaluate equations (3.8). First off, we note that calculating the bias-gradients

$$\begin{aligned} \nabla_{b_j} \mathcal{R} &= \left(\frac{\mathrm{d}\,z_j}{\mathrm{d}\,b_j}\right)^* \cdot \left(\frac{\mathrm{d}\,\zeta_j}{\mathrm{d}\,z_j}\right)^* \cdot \nabla_{\zeta_j} \mathcal{R} \\ &= \mathrm{id}^* \cdot \left(\frac{\mathrm{d}\,\zeta_j}{\mathrm{d}\,z_j}\right)^* \cdot h_j \\ &= \left(\frac{\mathrm{d}\,\zeta_j}{\mathrm{d}\,z_j}\right)^* \cdot h_j \end{aligned} \tag{3.9}$$

requires evaluating

$$\eta_j := \left(\frac{\mathrm{d}\,\zeta_j}{\mathrm{d}\,z_j}\right)^* \cdot h_j.$$

Note that due to $h_j = \nabla_{\zeta_j} \mathcal{R}$, one has $\eta_j = \nabla_{z_j} \mathcal{R}$. Right now, we do not have a way of evaluating η_j yet, but we will derive an expression for it later.
Similarly to the bias-gradients, we express the gradients of the linear weights as

$$\begin{aligned} \nabla_{\theta_j} \mathcal{R} &= \left(\frac{\mathrm{d}\,\xi_{j-1}}{\mathrm{d}\,\theta_j}\right)^* \cdot \nabla_{\xi_{j-1}} \mathcal{R} \\ &= \left(\frac{\mathrm{d}\,\xi_{j-1}}{\mathrm{d}\,\theta_j}\right)^* \cdot q_{j-1}. \end{aligned} \tag{3.10}$$

Since $\xi_{j-1} = K_j^T(\theta_j, \zeta_j)$ and because K_j^T is a continuous bilinear operator, we can harness the generalized product rule (Theorem 3.2.12) for equation (3.10):

$$\begin{aligned} \frac{\mathrm{d}\,\xi_{j-1}}{\mathrm{d}\,\theta_j}^* &= \left(K_j^T\left(\frac{\mathrm{d}\,\theta_j}{\mathrm{d}\,\theta_j}\bullet, \zeta_j\right) + K_j^T\left(\theta_j, \frac{\mathrm{d}\,\zeta_j}{\mathrm{d}\,\theta_j}\bullet\right)\right)^* \\ &= K_j^T(\bullet, \zeta_j)^* + K_j^T\left(\theta_j, \frac{\mathrm{d}\,\zeta_j}{\mathrm{d}\,\theta_j}\bullet\right)^* \\ &= K_j^{\square}(\bullet, \zeta_j) + \left(\frac{\mathrm{d}\,\zeta_j}{\mathrm{d}\,\theta_j}\right)^* \cdot K_j\left(\theta_j, \bullet\right), \end{aligned}$$

where we used the anti-distributivity of adjoint operators. According to Figure 3.2, ζ_j only depends on θ_j through z_j. We hence can apply the chain rule

$$\frac{\mathrm{d}\,\zeta_j}{\mathrm{d}\,\theta_j} = \frac{\mathrm{d}\,\zeta_j}{\mathrm{d}\,z_j} \cdot \frac{\mathrm{d}\,z_j}{\mathrm{d}\,\theta_j} = \frac{\mathrm{d}\,\zeta_j}{\mathrm{d}\,z_j} \cdot K_j(\bullet, x_{j-1})$$

Plugging this into equation (3.10) yields

$$\begin{aligned}
\nabla_{\theta_j}\mathcal{R} =& K_j^{\square}(q_{j-1}, \zeta_j)^* + \left(\frac{\mathrm{d}\,\zeta_j}{\mathrm{d}\,\theta_j}\right)^* \cdot K_j\left(\theta_j, q_{j-1}\right) \\
=& K_j^{\square}(q_{j-1}, \zeta_j) + \left(\frac{\mathrm{d}\,z_j}{\mathrm{d}\,\theta_j}\right)^* \cdot \left(\frac{\mathrm{d}\,\zeta_j}{\mathrm{d}\,z_j}\right)^* \cdot h_j \\
=& K_j^{\square}(q_{j-1}, \zeta_j) + K(\bullet, x_{j-1})^* \cdot \zeta_j \\
=& K_j^{\square}(q_{j-1}, \zeta_j) + K_j^{\square}\left(x_{j-1}, \eta_j\right),
\end{aligned}$$

which means that for both $\nabla_{b_j}\mathcal{R}$ and $\nabla_{\theta_j}\mathcal{R}$, we need a way of evaluating $\eta_j = \nabla_{z_j}\mathcal{R}$.

Here, we may finally make use of the fact that we assumed φ_j to be coordinate-wise Fréchet differentiable (Def. 3.3.1) for all $j < L$ (with some symmetric, bilinear operator M_j). This allows us to write

$$\begin{aligned}
\frac{\mathrm{d}\,\zeta_j}{\mathrm{d}\,z_j} =& \frac{\mathrm{d}\,\Phi_j'(z_j) \cdot \xi_j}{\mathrm{d}\,z_j} = \frac{\mathrm{d}\,M_j(\varphi_j'(z_j), \xi_j)}{\mathrm{d}\,z_j} \\
=& M_j\left(\frac{\mathrm{d}\,\varphi_j'(z_j)}{\mathrm{d}\,z_j}\bullet, \xi_j\right) + M_j\left(\varphi_j'(z_j), \frac{\mathrm{d}\,\xi_j}{\mathrm{d}\,z_j}\bullet\right)
\end{aligned} \tag{3.11}$$

Applying the adjoint

$$\frac{\mathrm{d}\,\zeta_j}{\mathrm{d}\,z_j}^* = \Phi_j''(z_j) \cdot M_j\left(\bullet, \xi_j\right) + \left(\frac{\mathrm{d}\,\xi_j}{\mathrm{d}\,z_j}\right)^* \cdot \Phi'(z_j),$$

with

$$\left(\frac{\mathrm{d}\,\xi_j}{\mathrm{d}\,z_j}\right)^* = \left(\frac{\mathrm{d}\,x_j}{\mathrm{d}\,z_j}\right)^* \cdot \left(\frac{\mathrm{d}\,\xi_j}{\mathrm{d}\,x_j}\right)^*$$

to h_j as in equations (3.10) and (3.9) yields

$$\begin{aligned}
\eta_j =& M_j\left(\Phi_j''(z_j) \cdot h_j, \xi_j\right) + \Phi_j'(z_j) \cdot \nabla_{x_j}\mathcal{R} \\
=& M_j\left(\Phi_j''(z_j) \cdot h_j, \xi_j\right) + \Phi_j'(z_j) \cdot \gamma_j
\end{aligned} \tag{3.12}$$

with

$$\gamma_j := \nabla_{x_j}\mathcal{R}.$$

While we do not have an expression for γ_j yet, we can recursively calculate it according to the formula

$$\begin{aligned}
\gamma_{j-1} =& \nabla_{x_{j-1}}\mathcal{R} \\
=& \left(\frac{\mathrm{d}\,z_j}{\mathrm{d}\,x_{j-1}}\right)^* \cdot \nabla_{z_j}\mathcal{R} \\
=& K_j^T(\theta_j, \eta_j)
\end{aligned} \tag{3.13}$$

Algorithm 3 Calculation of the forward-backward pass and the weight-gradients

for $j \leftarrow L$ **to** 1 **do**
  if $j = L$ **then**
    Initialize η_L according to Section 3.5
  else
    $\eta_j = M_j\left(\Phi_j''(z_j) \cdot h_j, \xi_j\right) + \Phi_j'(z_j) \cdot \gamma_j$
  end if
  $\nabla_{\theta_j}\mathcal{R} = K_j^{\square}(q_{j-1}, \zeta_j) + K_j^{\square}(x_{j-1}, \eta_j)$
  $\nabla_{b_j}\mathcal{R} = \eta_j$
  if $j > 1$ **then**
    $\gamma_{j-1} = K_j^T(\theta_j, \eta_j)$
  end if
end for

for which the initial value η_L is required, which depends on the exact penalty term (cf. the subsequent Section 3.5). Equipped with this, the weight gradients

$$\begin{aligned}\nabla_{\theta_j}\mathcal{R} &= K_j^{\square}(q_{j-1}, \zeta_j) + K_j^{\square}(x_{j-1}, \eta_j)\\ \nabla_{b_j}\mathcal{R} &= \eta_j\end{aligned}$$

can finally be calculated. This procedure is summarized in Algorithm 3.

3.5 Initial Values η_L

In section 3.4.1, a generalization of the different penalty terms to the form

$$\mathcal{R} := p\left(\left(\frac{\mathrm{d}\, x_L}{\mathrm{d}\, x_0}\right)^* \cdot v\right) \tag{3.14}$$

was introduced. The initial value η_L, which is needed in order to initialize Algorithm 3, depends on the output layer's activation function φ_L and $v = \xi_L$. Typical special cases for the activation function include $\varphi_L = \mathrm{softmax}$ (for classification problems) or $\varphi_L = \mathrm{id}$ (non-categorical targets like in regression or if one wants to apply penalties to derivatives of logits). Because softmax is not a coordinate-wise activation function, we cannot harness equation (3.11).
For v, we can identify two particular special cases: Those where v is independent of the network and $v = \nabla_{x_L}\mathcal{L} = -y \oslash x_L$ for classical double backpropagation.
For these reasons, η_L needs to be calculated explicitly for the cases above.

3.5.1 Softmax Penalties

Here, we derive η_L for penalty terms of the form (3.14), where v is independent of the neural network's input and $\varphi_L = \mathrm{softmax}$. According to equation (3.2),

$$\frac{\mathrm{d}\, x_L}{\mathrm{d}\, z_L} \cong \mathrm{diag}(x_L) - x_L x_L^T$$

is self-adjoint (symmetric). We hence have

$$\begin{aligned}\zeta_L &= \left(\frac{\mathrm{d}\, x_L}{\mathrm{d}\, z_L}\right)^* \cdot v \\ &= x_L \odot v - \langle x_L, v\rangle \cdot x_L\end{aligned}$$

and particular, for $v = e^{[i]}$ and $x_L = (x_L^1, \ldots, x_L^C)^T$, this can be written as

$$\begin{aligned}\left(\frac{\mathrm{d}\, x_L}{\mathrm{d}\, z_L}\right)^* \cdot e^{[i]} &= x_L^i e^{[i]} - x_L^i \cdot x_L \\ &= x_L^i \cdot (e^{[i]} - x_L).\end{aligned}$$

Since v does not depend on x_L, we can treat v as a constant, which yields

$$\begin{aligned}\frac{\mathrm{d}\, \zeta_L}{\mathrm{d}\, x_L} &\cong \left(\frac{\partial \zeta_L^i}{\partial x_L^j}\right)_{ij} \\ &= \left(\frac{\partial \delta_{ij} \cdot x_L^i \cdot v^i}{\partial x_L^j}\right)_{ij} - \left(\langle x_L, v\rangle \cdot \frac{\partial x_L^i}{\partial x_L^j} + x_L^i \cdot \frac{\partial \langle x_L, v\rangle}{\partial x_L^j}\right)_{ij} \\ &= \left(\delta_{ij} \cdot v^j\right)_{ij} - \left(\langle x_L, v\rangle \cdot \delta_{ij}\right)_{ij} - \left(x_L^i \cdot v^j\right)_{ij} \\ &= \operatorname{diag}(v) - \langle x_L, v\rangle \cdot I - x_L v^T\end{aligned}$$

(with identity matrix I). This leads to the following expression for η_L:

$$\begin{aligned}\eta_L &= \left(\frac{\mathrm{d}\, x_L}{\mathrm{d}\, z_L}\right)^* \left(\frac{\mathrm{d}\, \zeta_L}{\mathrm{d}\, x_L}\right)^* \cdot h_L \\ &= \left(\frac{\mathrm{d}\, x_L}{\mathrm{d}\, z_L}\right)^* \left(\operatorname{diag}(v) - \langle x_L, v\rangle \cdot I - x_L v^T\right)^T \cdot h_L \\ &= \left(\frac{\mathrm{d}\, x_L}{\mathrm{d}\, z_L}\right)^* \left(\operatorname{diag}(v) - \langle x_L, v\rangle \cdot I - v x_L^T\right) \cdot h_L \\ &= \left(\frac{\mathrm{d}\, x_L}{\mathrm{d}\, z_L}\right)^* (v \odot h_L - \langle x_L, v\rangle \cdot h_L - \langle x_L, h_L\rangle \cdot v) \\ &= \left(\operatorname{diag}(x_L) - x_L x_L^T\right) \cdot (v \odot h_L - \langle x_L, v\rangle \cdot h_L - \langle x_L, h_L\rangle \cdot v) \\ &= x_L \odot v \odot h_L - \langle x_L, v\rangle \cdot x_L \odot h_L - \langle x_L, h_L\rangle \cdot x_L \odot v \\ &\quad - x_L \cdot \langle x_L, v \odot h_L\rangle + \langle x_L, v\rangle \langle h_L, x_L\rangle x_L + \langle x_L, h_L\rangle \langle v, x_L\rangle x_L\end{aligned}$$

As the last two summands are equal, we conclude that

$$\begin{aligned}\eta_L = x_L \odot v \odot h_L &- \langle x_L, v\rangle \cdot x_L \odot h_L - \langle x_L, h_L\rangle \cdot x_L \odot v \\ &- x_L \cdot \langle x_L, v \odot h_L\rangle + 2\langle x_L, v\rangle \langle h_L, x_L\rangle x_L.\end{aligned} \tag{3.15}$$

3.5.2 Softmax with Non-Negative Log-Likelihood Loss

The following deals with classical double backpropagation, where a penalty of the form

$$p(\nabla_{x_0} \ell(x_L, y))$$

is applied, with ℓ denoting the non-negative log likelihood loss function as defined in equation (3.4). As detailed in section 3.4.1, this penalty term can be written in the general form (3.3) with $\varphi_L = \text{softmax}$ and $v = \nabla_{x_L}\ell(x_L, y) = -y \oslash x_L$.

$$\begin{aligned}\zeta_L &= \left(\frac{\mathrm{d}\, x_L}{\mathrm{d}\, z_L}\right)^* \cdot \nabla_{x_L}\mathcal{L} \\ &= \left(\operatorname{diag}(x_L) - x_L x_L^T\right) \cdot (-y \oslash x_L) \\ &= -y + \left(\sum_{i=1}^{C} (x_L^i y^i)/x_L^i\right) x_L \\ &= x_L - y,\end{aligned}$$

where we used that $\sum_i y^i = 1$, where $y = (y^1, \ldots, y^C)^T$. Thus, for classic double backpropagation the formula for the initial value of the forward-backward pass

$$\begin{aligned}\eta_L &= \left(\frac{\mathrm{d}\, x_L}{\mathrm{d}\, z_L}\right)^* \cdot \left(\frac{\mathrm{d}\, \zeta_L}{\mathrm{d}\, x_L}\right)^* \cdot h_L \\ &= \left(\operatorname{diag}(x_L) - x_L x_L^T\right) \cdot (\mathrm{id})^* \cdot h_L \\ &= x_L \odot h_L - x_L \langle x_L, h_L \rangle\end{aligned}$$

holds.

3.5.3 Identity Function

If one applies a penalty to the derivatives of the logit-layers (z_L), one can still represent this case as in equation (3.3) by modelling φ_L as an identity map, so that

$$\frac{\mathrm{d}\, x_L}{\mathrm{d}\, z_L} = \mathrm{id}$$

and $\zeta_L = \xi_L = v$. Since ζ_L is constant in z_L, we have $\eta_L = \left(\frac{\mathrm{d}\, \zeta_L}{\mathrm{d}\, z_L}\right)^* \cdot h_L = 0$.

3.6 Runtimes

In the last section, the double backpropagation rules were derived. For most networks (in particular in convolutional neural networks), the most time-consuming portion of the network lies in the calculation of the forward and transposed operators K_j and K_j^T. Here, we will consider the runtimes of different penalty functionals in terms of these operators and offer optimized implementations of some.

3.6.1 The General Case

In the general case, the forward, backward and backward-backward pass each require L evaluations of the (transposed) operators. The forward-backward pass however does not require to evaluate $\gamma_0 = K_1^T(\theta_1, \eta_1)$, which is why in this case only $L - 1$ transposed operations need to be performed. This results in a time complexity of

$$4L - 1$$

operations for the full double backpropagation.
If the full loss term is $\mathcal{L} + \lambda p(\xi_0)$, but ξ_0 is *not* $\nabla_{x_0}\mathcal{L}$ (as in classical double backpropagation), one needs to perform another $L-1$ operations, because $\nabla_{\theta_j}\mathcal{L}$ and $\nabla_{b_j}\mathcal{L}$ are needed, whereas some values can be reused in classical double backpropagation (as detailed in section 3.4.3). In summary, these cases require

$$5L - 2$$

linear operations, compared to the $2L - 1$ operations of a network without a penalty term of the type (3.3).

3.6.2 Locally Linear Activation Functions

If the activations $x_j = \varphi_j(z_j)$ are not only coordinate-wise twice Fréchet differentiable in z_j, but also locally linear (as with ReLU or leaks ReLU) in z_j, the double backpropagation takes a simpler form:
As $\Phi''(z_j)$ is the null operator (*almost everywhere*, for every z_j for which $\varphi_j(z_j)$ is twice Fréchet differentiable) and because M_j is linear in both arguments, equation (3.12) reduces to

$$\eta_j = \Phi_j'(z_j) \cdot \gamma_j.$$

In general, this however does not reduce the amount of linear operations K_j and K_j^T.

3.6.3 Linear Output Nodes and Locally Linear Activation Functions

If the penalty terms are applied on the derivatives of linear output nodes (i.e. $\varphi_L = \mathrm{id}$, for example when $\mathcal{R} = \|\nabla_{x_0} z_L^i\|_{\mathcal{X}_0}^2$), then $\eta_L = 0$, as demonstrated in Section 3.5. While this only reduces the amount of linear operations by 1 (through $\gamma_{L-1} = K_L^T(\theta_j, 0) = 0$), the effect cascades when also a locally linear activation function is used. This is because in that case, $\eta_j = \gamma_j = 0$ for all j, according to equations (3.12) and (3.13). As a result, the weight gradients reduce to $\nabla_{b_j}\mathcal{R} = 0$ for all j and $\nabla_{\theta_j}\mathcal{R} = K_j^{\square}(q_{j-1}, \zeta_j)$, which means that one does not need to perform the forward-backward pass at all. All in all, the reduced number of linear operations for the penalty term is then $3L$, and $4L - 1$ for the full loss term $\mathcal{L} + \lambda\mathcal{R}$ (the same as for the classical double backpropagation loss $\mathcal{L} + \lambda p(\nabla_{x_0}\mathcal{L})$).

3.6.4 Jacobian Penalties

If the penalty term is

$$\mathcal{R} = \sum_{i=1}^{C} \mathcal{R}^{[i]} :\equiv \sum_{i=1}^{C} \|\nabla_{x_0} x_L^i\|_2^2$$

(equivalent to the squared Frobenius norm of the Jacobian), the double backpropagation scheme needs to be performed C times. Note that the forward pass has to be performed only once. The total number of linear operations is thus $L + C(3L-1)$, plus another $L-1$ if one needs $\nabla_{\theta_j}\mathcal{L}$ and $\nabla_{b_j}\mathcal{L}$ (which results in $2L-1+C(3L-1)$ linear operations) in general.

We now present an optimized double backpropagation algorithm for this scenario, that applies if only locally linear activation functions like ReLU (up to the final softmax layer)

are employed, which allows one to abuse a certain linearity. This algorithm reduces the number of performed linear operations by roughly a third, while keeping the required memory roughly the same as with a single conventional double backpropagation. We will now index variables such as η_j that relate to a certain $\mathcal{R}^{[i]}$ via $\eta_j^{[i]}$. We note the following:

1. We can write $\nabla_{\theta_j}\mathcal{R} = \hat{\theta}_j^1 + \hat{\theta}_j^2$, where

$$\hat{\theta}_j^1 = \sum_{i=1}^{C} K_j^{\square}(q_{j-1}^{[i]}, \zeta_j^{[i]}) \text{ and } \hat{\theta}_j^2 = \sum_{i=1}^{C} K_j^{\square}\left(x_{j-1}, \eta_j^{[i]}\right).$$

2. When looping over i, $\hat{\theta}_j^1$ can be calculated by an update scheme via 'initializing' $\hat{\theta}_j^1$ as 0 and adding $K_j^{\square}(q_{j-1}^{[i]}, \zeta_j^{[i]})$ after every backward-backward pass. This way, only the accumulated $\hat{\theta}_j^1$ needs to be kept in memory, compared to every summand.

3. $K_j^{\square}(x_{j-1}, \bullet)$ is linear, such that

$$\sum_i K_j^{\square}(x_{j-1}, \eta_j^{[i]}) = K_j^{\square}(x_{j-1}, \textstyle\sum_i \eta_j^{[i]}),$$

   which means that we can accumulate $\hat{\eta}_j = \sum_i \eta_j^{[i]}$.

4. $\hat{\eta}_L$ can be calculated (similarly to $\hat{\theta}_j^1$) via initialization as 0 and updating after every backward-backward pass by adding $\hat{\eta}_L^{[i]}$.

5. Since $\Phi_j''(z_j)$ is the null operator, $\eta_j^{[i]}$ depends *linearly* on $\gamma_j^{[i]}$, as explained in Section 3.6.2. As a result,

$$\hat{\eta}_j = \Phi_j'(z_j) \cdot \left(\textstyle\sum_i \gamma_j^{[i]}\right) = \Phi_j'(z_j) \cdot \hat{\gamma}_j$$

   with $\hat{\gamma}_j = \sum_i \gamma_j^{[i]}$.

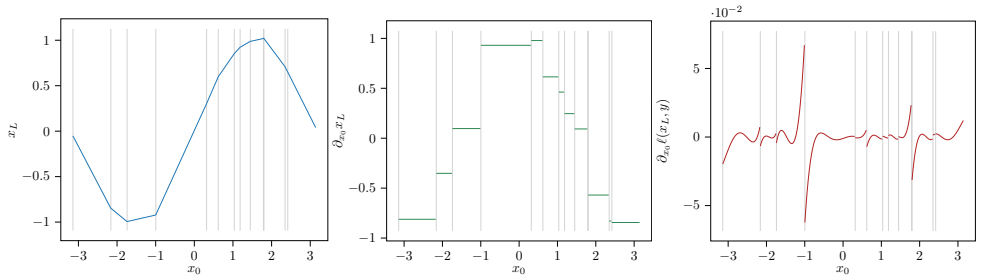
6. $\hat{\eta}_j$ and consequently $\hat{\theta}_j^2 = K_j^{\square}(x_{j-1}, \hat{\eta}_j)$ are linear in $\hat{\eta}_L$ and can be calculated recursively from $\hat{\eta}_L$. This way, only one forward-backward pass needs to be performed, compared to the C passes that normally need to be performed.

7. By erasing variables from memory once they are no longer needed, this optimized algorithm does not require more memory than a single conventional double backpropagation procedure.

We end up with $2L - 1 + 2CL$ linear operations (L for the forward pass, L for each of the C backward and backward-backward passes and $L - 1$ for the forward-backward pass). As the naïve implementation requires $L + 3CL - C$ linear operations, about a third of the linear operations are saved (because $2CL$ respectively $3CL$ represent the bulk of the operations). The algorithm is presented in detail in Algorithm 4.

Algorithm 4 For a neural network with locally linear activation functions (e.g. ReLU) and softmax output, the weight gradients of the penalty term $\mathcal{R} = \sum_{i=1}^{C} \mathcal{R}^{[i]} = \sum_{i=1}^{C} \|x_L^i\|_{\mathcal{X}_0}^2$ can be calculated using an algorithm with only $2CL + 2L - 1$ linear operations, compared to the naïve implementation with $3CL + L - C$. Additionally, the memory requirements are $\mathcal{O}(1)$ in C.

forward pass
Initialize x_0
for $j \leftarrow 1$ **to** L **do**
  $z_j = K_j(\theta_j, x_{j-1}) + b_j$
  $x_j = \varphi_j(z_j)$
  $a_j = \varphi_j'(z_j)$, **delete** z_j
end for
Initialize $\hat{\theta}_j^1 = 0$, $\hat{\eta}_j = 0$
for $i \leftarrow 1$ **to** C **do**
  # i-th backward pass
  Initialize $\xi_L^{[i]} = e^{[i]}$
  for $j \leftarrow L$ **to** 1 **do**
    $\zeta_j^{[i]} = M_j(a_j, \xi_j^{[i]})$, **delete** $\xi_j^{[i]}$
    $\xi_{j-1}^{[i]} = K_j^T(\theta_j, \zeta^{[i]})$
  end for
  # i-th backward-backward pass
  Initialize $q_0^{[i]} = \nabla_{\xi_0^{[i]}} \|\xi_0^{[i]}\|_{\mathcal{X}_0}^2 = 2\xi_0^{[i]}$
  for $j \leftarrow 1$ **to** L **do**
    $\hat{\theta}_j^1 \leftarrow \hat{\theta}_j + K_j^{\square}(q_{j-1}^{[i]}, \zeta_j^{[i]})$, **delete** $\zeta_j^{[i]}$
    $h_j^{[i]} = K_j(\theta_j, q_{j-1}^{[i]})$, **delete** $q_{j-1}^{[i]}$
    $q_j^{[i]} = M_j(a_j, h_j^{[i]})$
    if $j \neq L$ **then delete** $h_j^{[i]}$ **end if**
  end for
  $\eta_L^{[i]} \leftarrow$ according to equation (3.15) with $v = e^{[i]}$
  $\hat{\eta}_L \leftarrow \hat{\eta}_L + \eta_L^{[i]}$
  delete $h_L^{[i]}, \eta_L^{[i]}$
end for
delete x_L
# cumulated forward-backward passes
for $j \leftarrow L$ **to** 1 **do**
  $\nabla_{\theta_j} \mathcal{R} = K_j^{\square}(q_{j-1}, \zeta_j) + K_j^{\square}(x_{j-1}, \eta_j)$
  $\nabla_{b_j} \mathcal{R} = \eta_j$
  if $j > 1$ **then**
    $\hat{\gamma}_{j-1} = K_j^T(\theta_j, \hat{\eta}_j)$
    delete $\hat{\eta}_j$
    $\hat{\eta}_{j-1} = M_{j-1}(a_{j-1}, \hat{\gamma}_{j-1})$
    delete $a_{j-1}, \hat{\gamma}_{j-1}$
  end if
end for



(a) Neural network output (b) Gradient of output node (c) Gradient of loss

Figure 3.3: Neural network trained to approximate a sine-curve. Vertical grey lines symbolize the boundaries of locally affine regions.

3.7 Loss Landscapes for (leaky) ReLU networks

In the following, we will only consider finite-dimensional networks. The loss landscapes of (leaky) ReLU networks represent special cases due to jump discontinuities in their derivatives.

3.7.1 Loss Landscape in the Inputs

As mentioned earlier, (leaky) ReLU networks partition the input space into polytopes in which the logit layer z_L is affine in x_0 (Raghu et al., 2017). This in turn means that the operator

$$\frac{\mathrm{d}\, z_L}{\mathrm{d}\, x_0} \tag{3.16}$$

is constant in x_0 in the interior of each polytope and in turn locally constant almost everywhere. Since the penalty term is given by

$$\mathcal{R} = p\left(\left(\frac{\mathrm{d}\, z_L}{\mathrm{d}\, x_0}\right)^* \cdot \zeta_L\right), \tag{3.17}$$

this implies that $\mathcal{R}$ is locally constant in x_0 within each polytope if (and only if) ζ_L is locally constant in this region as well. Nevertheless, a jump discontinuity may occur when for some z_j, the activation $x_j = \varphi_j(z_j)$ enters a different locally linear region of the (leaky) ReLU nonlinearity. This is the case when an entry of the vector z_j switches between $(-\infty, 0]$ and $(0, \infty)$.

3.7.2 Loss Landscape in the Parameters

The above considerations lead to the question, whether $\mathcal{R}$ may also be locally constant almost everywhere in the parameter space. If that were the case, any derivative-based optimization algorithm (like stochastic gradient descent) would instantly fail, because then automatically $\nabla_\Theta \mathcal{R} = 0$. However, for fixed x_0 in the interior of a polytope, the operator (3.16) is locally affine in the linear weights θ_j and locally constant in the biases b_j. As z_j depends (locally affine) on θ_k and (locally constant) on b_k (for $k \leq j$), this means that $\mathcal{R}$ is luckily not locally constant almost everywhere in Θ. The exact functional dependence then hinges on whether and how ζ_L depends on the weights.

However, $\mathcal{R}$ thus also 'inherits' the jump discontinuities from $\frac{\mathrm{d}\, z_L}{\mathrm{d}\, x_0}$, which may introduce numerical problems when using derivative-based optimization.

In reality, the problem of jump discontinuities may however be not as severe as it may seem at first glance: Usually, the optimization methods are applied not on a penalty term for a single point x_0 (with label y), but on the average value of $\mathcal{R}$ for a whole batch $\{(x_0^{(i)}, y^{(i)})\}_{i=1,\dots,M}$. While the number of jump discontinuities adds up over the number of samples in this batch, the averaging process introduces a 'smoothing' effect on the loss landscape. These phenomena are empirically demonstrated on a simple toy example.

3.7.3 Experiments

For the following extremely simple toy example, we created a dataset of 1500 points $\{(x_0^{(i)}, y^{(i)})\}_{i=1,\dots,1500}$, where $x_0^{(i)} \in [-\pi, \pi]$ and $y^{(i)} = \sin(x^{(i)})$. We then fitted a small multilayer perceptron with 2 hidden ReLU layers (with 8 respectively 5 neurons) and a linear output layer (with 1 neuron and $\varphi_L = \mathrm{id}$) to this dataset, using the squared loss. This architecture was chosen, because experimentally it yielded a good approximation to the sine-curve, while still exhibiting visibly locally linear regions. The goal of this section is not to add to the vast empirical evidence of the viability of double backpropagation (see the many examples from Section 3.1.2), but rather to explain *why* it works from the standpoint of optimization by considering the loss landscape of the penalty term (3.17). We start by considering the loss landscape in the inputs. In Figure 3.3a, the resulting approximating neural network is depicted. As expected, the neural network creates a locally affine, continuous output. Since the network maps real numbers to real numbers, we can identify the operator (3.16) with the partial derivative $\partial_{x_0} x_L = \partial_{x_0} z_L \in \mathbb{R}$ (where this equality holds due to $\varphi_L = \mathrm{id}$). As displayed in Figure 3.3b, this derivative exhibits locally constant regions separated by the locations of non-differentiability. As a consequence, any penalty term

$$\mathcal{R}_{\text{node}} := p\left(\partial_{x_0} z_L\right)$$

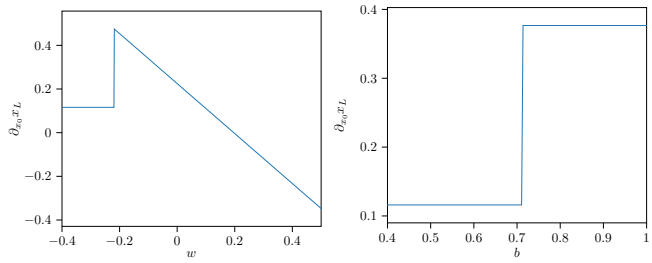
for some a.e. differentiable p would necessarily be locally constant in x_0 as well (not depicted here). The loss landscape in x_0 for the classical double backpropagation penalty

$$\mathcal{R}_{\text{cdb}} := \|\partial_{x_0} \ell(x_L, y)\|_2^2 = \left(\left(\partial_{x_0}(x_L - y)^2\right)\right)^2$$

is depicted in Figure 3.3c and shows the expected jump discontinuities.

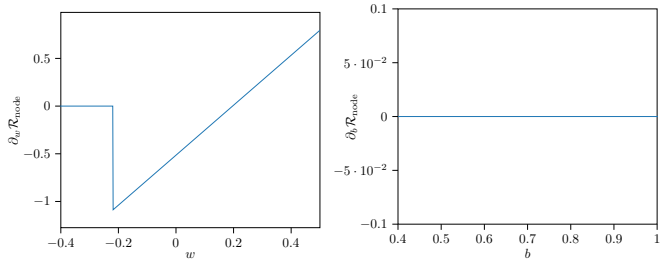
Since even a neural network as small as this one has 61 parameters, one cannot feasibly depict the loss landscape over *all* parameters. Therefore, we fix the weights of the trained network and vary only one parameter of each the weight matrix and bias vector of the second hidden layer. We will call these parameters w and b. In Figure 3.4, for fixed $x_0 \approx 1.022$, the dependence of $s := \partial_{x_0} z_L \in \mathbb{R}$ on w respectively b is shown. As predicted in section 3.7.2, s is locally affine in w and exhibits jump discontinuities. Furthermore, s as a function of b is locally constant and exhibits jump discontinuities.

For the actual optimization, the properties of interests are the derivatives of the *penalty terms*. For $\mathcal{R}_{\text{node}}$, we choose $p : s \mapsto s^2$ and visualize $\partial_w \mathcal{R}_{\text{node}}$ and $\partial_b \mathcal{R}_{\text{node}}$ in Figure 3.5. While $\partial_w \mathcal{R}_{\text{node}}$ exhibits a piecewise linear behavior (including a locally constant portion) with a jump discontinuity, $\partial_b \mathcal{R}_{\text{node}}$ is constant 0 (as a consequence of $\mathcal{R}_{\text{node}}$ being locally



(a) Dependence on linear weight w (b) Dependence on bias b

Figure 3.4: Dependence of $\partial_{x_0} z_L$ on w and b.



(a) Dependence on linear weight w (b) Dependence on bias b

Figure 3.5: Derivatives of the penalty term $\mathcal{R}_{\text{node}}$ with respect to w and b.

constant due to $\varphi_L = \text{id}$, as explained in section 3.7.2). This demonstrates how first-order optimization of $\mathcal{R}_{\text{node}}$ for a single example x_0 may suffer from instabilities, whenever a neuron switches between the locally linear regions of the (leaky) ReLU nonlinearity.
We perform a similar analysis for the classical double backpropagation penalty $\mathcal{R}_{\text{cdb}}$ and display our results in Figure 3.6. While the jump discontinuities appear in the same spots, the non-constant portion of $\partial_w \mathcal{R}_{\text{cdb}}$ exhibits nonlinear behavior due to the (nonlinear) choice of p and the dependence of ξ_0 on w. A central difference in the behavior of the bias derivative $\partial_b \mathcal{R}_{\text{cdb}}$ compared to $\partial_b \mathcal{R}_{\text{node}}$ lies in the fact that this derivative is *not* constant 0. This is because classical double backpropagation in general yields $\eta_L \neq 0$.

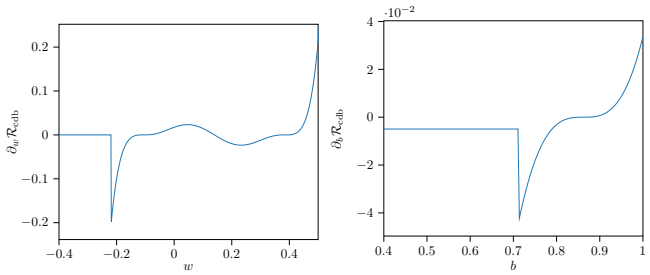
As we will show now, the feared instabilities can be reduced by batchwise optimization, which is standard practice. To visualize this, we randomly pick a batch $\mathfrak{B} = \{(x_0^{(i)}, y^{(i)})\}_{i=1,\dots,M}$ for $x^{(i)} \in [-\pi, \pi]$, $y^{(i)} = \sin(x^{(i)})$ with batch size $M = 256$ and visualize the averaged penalty terms

$$\mathcal{R}_{\text{node}}^{\mathfrak{B}} := \frac{1}{M} \sum_{i=1}^{M} \left(\partial_{x_0^{(i)}} x_L^{(i)} \right)^2$$

and

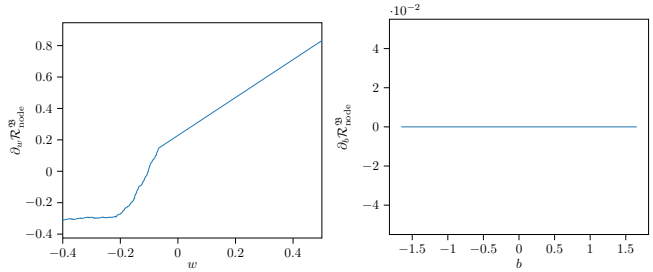
$$\mathcal{R}_{\text{cdb}}^{\mathfrak{B}} := \frac{1}{M} \sum_{i=1}^{M} \left\| \partial_{x_0^{(i)}} \ell(x_L^{(i)}, y^{(i)}) \right\|_2^2$$

in Figures 3.7 and 3.8. While jump discontinuities of the averaged penalty terms are still



(a) Dependence on linear weight w (b) Dependence on bias b

Figure 3.6: Derivatives of the penalty term $\mathcal{R}_{\text{cdb}}$ on w and b.



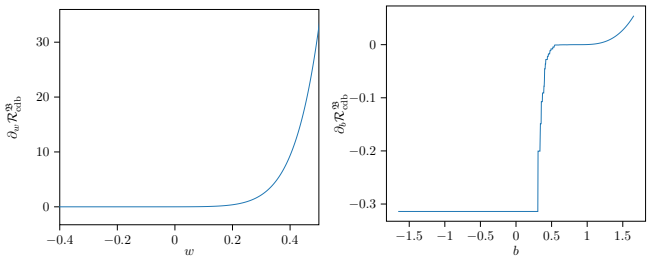
(a) Dependence on linear weight w (b) Dependence on bias b

Figure 3.7: Derivatives of the penalty term $\mathcal{R}^{\mathfrak{B}}_{\text{node}}$ on w and b. The averaging over the batch $\mathfrak{B}$ creates a 'smoothed' landscape over w.

visible, the fact that the individual discontinuities lie close together in parameter space creates the effect of 'almost smooth' loss landscapes. Due to this smoothing effect, the optimization using batch optimization is much less impaired by the discontinuities than for a single example x_0, which explains their success in the applications listed in section 3.1.2, even when using (leaky) ReLU activation functions.

3.8 Conclusion & Outlook

In this chapter, an in-depth description of 'double backpropagation' procedures was provided, which come into play whenever a loss function contains derivatives of output nodes with respect to input nodes. We offer a unified perspective for a large class of such loss functions and describe the derivatives in the general framework of Fréchet derivatives on Hilbert spaces. For this, we developed a theory of adjoint operators for continuous, bilinear operators, which covers many common layer types. The obtained description of the involved derivatives allows us to present optimized double backpropagation schemes for some networks, which reduces the time complexity by roughly a third in this case. Furthermore, we provided a description for the (discontinuous) loss landscape for derivative-based losses of (leaky) ReLU networks both in the inputs as well as the parameters. We further demonstrate that training in batches introduces a 'pseudo-smoothing' effect to the loss landscape, which may help against instabilities due to the aforementioned discontinuities.



(a) Dependence on linear weight w (b) Dependence on bias b

Figure 3.8: Derivatives of the penalty term $\mathcal{R}_{\text{cdb}}^{\mathfrak{B}}$ on w and b. The averaging over the batch $\mathfrak{B}$ creates a 'smoothed' landscape.

While the theoretical framework introduced in this chapter is quite general, minor extensions are needed in order to cover almost all widely-used architectures.
One particularly common design paradigm of feedforward neural networks are *residual connections* introduced by He et al. (2016), which cannot be represented with the model defined in Section 3.3. Residual connections can be represented as functions

$$f : x \mapsto x + \nu(x),$$

where ν represents a subnetwork consisting of multiple layers. The fact that

$$\frac{\mathrm{d}\, f(x)}{\mathrm{d}\, x} = \mathrm{id} + \frac{\mathrm{d}\, \nu(x)}{\mathrm{d}\, x}$$

demonstrates how residual connections in the forward pass persists even in the backward pass and thus also in the backward-backward and forward-backward passes.

Another classical building block of neural networks are *max pooling layers* (Definition 2.2.5). In max pooling, the largest entry of a set of entries (e.g. pixels) is kept, while the rest is discarded. As an example, the max pooling layer over 2 pixels with a window size of 2, $\mu : \mathbb{R}^2 \to \mathbb{R}$, can be written as

$$\begin{aligned}\mu(x) &= \begin{cases} x_1 & \text{if} \quad x_1 \geq x_2 \\ x_2 & \text{if} \quad x_1 < x_2 \end{cases} \\ &= \begin{cases} (1 \;\; 0) \cdot x \text{ if } x_1 \geq x_2 \\ (0 \;\; 1) \cdot x \text{ if } x_1 < x_2, \end{cases}\end{aligned}$$

such that $\mu(x) = A(x) \cdot x$. Here, $A(x)$ is a suitable matrix with entries from $0, 1$, which is locally constant in x. This means in particular that

$$\frac{\mathrm{d}\, \mu(x)}{\mathrm{d}\, x} = \frac{\mathrm{d}\, A(x) \cdot x}{\mathrm{d}\, x} = A(x)$$

almost everywhere, which means that in the derivation of (double) backpropagation rules, max pooling can 'almost' be treated like a linear operator.

From a mathematical point of view, it may be of interest to generalize the presented theory from Hilbert spaces to more general structures, e.g. Banach spaces or normed

spaces. This concerns both the differentiation itself for the double backpropagation rules, as well as the required theory for bilinear mappings. In the following, we will try to draw some connections to related work and point towards possible extensions.
In Section 3.2.1 (and elaborated upon in Appendix A), one such generalization for the latter was presented in the form of the Arens-adjoint (Arens, 1951), which is defined for bilinear, continuous function between normed spaces. This results in mappings involving both primal and dual spaces. If the primal operator is between Hilbert spaces, these can then be identified with the transposed and weight-adjoint operators, such that up to isomorphism, the Arens-adjoint can be viewed as a generalization of the construction presented here. In normed spaces, this is generally not possible and one has to work out the theory in the dual space itself. At least for reflexive spaces (like L_p-spaces), an identification of the bidual space with the primal space is possible, which *might* allow for some simplifications. The question remains, to which extent the Ahrens-adjoint(s) are even useful or required in more general contexts than the very well-behaved Hilbert space setting, in which bilinear adjoints come into play when applying the gradient chain rule in the derivation of the (double) backpropagation rules.

Fréchet derivatives and their sum, product and chain rule each generalize readily to normed spaces, whereas Definition 3.2.3 of the gradient is however not well-defined anymore. This is because the Fréchet derivative of a scalar-valued function (which is a continuous, linear form) can now not be uniquely identified with a vector – the gradient. This is owed to the fact that the Riesz representation theorem does not hold in general for (non-Hilbert) normed spaces. Still, one can calculate $\frac{\partial \mathcal{L}}{\partial \theta^j}$ and similar terms with the chain rule for Fréchet derivatives. Despite the lack of gradients, there are still various steepest descent approximations in Banach spaces (Schöpfer et al., 2006; Bonesky et al., 2008; Barbu and Precupanu, 2012). The employed Banach analogues of the Riesz isomorphism in this setting are the set-valued *duality mappings*. Adding certain restrictions (such as smoothness, reflexivity and convexity of the spaces) allows for reducing the gap to the Hilbert space setting.

While the generalization to Hilbert spaces made it possible to derive a general, coordinate-independent representation of double backpropagation, this step to more general spaces probably does not have a direct benefit for most real-world deployed neural networks. It may still be useful for functional analytic network models such as (Bruna and Mallat, 2013; Wiatowski and Bölcskei, 2017) or unrolled optimization schemes in general function space settings such as *analytic deep priors* (Dittmer et al., 2019).

Chapter 4

Adversarial Robustness and Saliency Map Interpretability

This chapter examines the relationship between a neural network's robustness to adversarial attacks and how structured its saliency maps are.

It is based on the following article:

> Christian Etmann, Sebastian Lunz, Peter Maass, and Carola-Bibiane Schönlieb. On the Connection Between Adversarial Robustness and Saliency Map Interpretability. In *Proceedings of the 36th International Conference on Machine Learning*, 97:1823-1832, 2019a.

4.1 Introduction

Despite impressive results in a variety of classification tasks (LeCun et al., 2015), even highly accurate neural network classifiers are plagued by a vulnerability to so-called *adversarial perturbations* (Szegedy et al., 2014). These adversarial perturbations are small, often visually imperceptible perturbations to the network's input, which however result in the network's classification decision being changed. Such vulnerabilities may pose a threat to real-world deployments of automated recognition systems, especially in security-critical applications such as autonomous driving or banking. This has sparked a large number of publications related to both the creation of adversarial attacks (Goodfellow et al., 2014; Kurakin et al., 2016; Moosavi-Dezfooli et al., 2016) as well as defenses against these (see (Schott et al., 2018) for an overview). Apart from the application-focused viewpoint, the observed adversarial vulnerability offers non-obvious insights into the inner workings of neural networks.

One particular method of defense is *adversarial training* (Madry et al., 2018), which aims to minimize a modified training objective. While this method – like all known approaches of defense – decreases the accuracy of the classifier, it is also successful in increasing the robustness to adversarial attacks, i.e. the perturbations need to be larger on average in order to change the classification decision.

(Tsipras et al., 2019) also notice that networks that are robustified in this way show

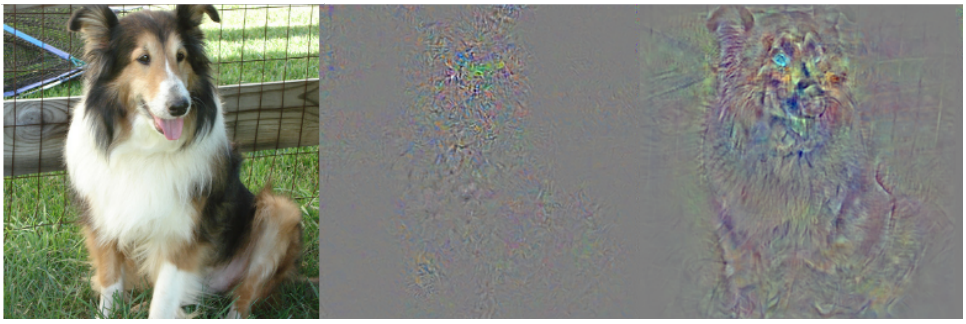


Figure 4.1: An image of a dog (left), the saliency maps of a highly non-adversarially-robust neural network (middle) and of a more robust network (right). We observe that the robust network gives a much clearer indication of what the classifier deems to be discriminative features. Details about saliency and the robustification are given in section 4.4. Most figures are best viewed on a screen.

interesting phenomena, which so far could not be explained. Neural networks usually exhibit very unstructured *saliency maps* (gradients of a classifier score with respect to the network's input (Simonyan et al., 2013)) which barely relate to the input image. On the other hand, saliency maps of robustified classifiers tend to be far more interpretable, in that structures in the input image also emerge in the corresponding saliency map, as exemplified in Figure 4.1. (Tsipras et al., 2019) describe this as an 'unexpected benefit' of adversarial robustness. In order to obtain a semantically meaningful visualization of the network's classification decision in non-robustified networks, the saliency map has to be aggregated over many different points in the vicinity of the input image. This can be achieved either via averaging saliency maps of noisy versions of the image (Smilkov et al., 2017) or by integrating along a path (Sundararajan et al., 2017). Other approaches typically employ modified backpropagation schemes in order to highlight the discriminative portions of the image. Examples of this include *guided backpropagation* (Springenberg et al., 2015) and *deep Taylor decomposition* (Montavon et al., 2017a).
In this work, we show that the interpretability of the saliency maps of a robustified neural network is not only a side-effect of adversarial training, but a general property enjoyed by networks with a high degree of robustness to adversarial perturbations. We first demonstrate this principle for the case of a linear, binary classifier and show that the 'interpretability' is due to the image vector and the respective image gradient aligning. For the more general, non-linear case we empirically show that while this relationship is true on average, the linear theory and the non-linear reality do not always agree. We empirically demonstrate that the more linear the model is, the stronger the connection between robustness and alignment becomes.

4.2 Adversarial Robustness and Saliency Maps

Since adversarial perturbations are small perturbations that change the predicted class of a neural network, it makes sense to define the robustness towards adversarial perturbations via the distance of the unperturbed image to its nearest perturbed image, such that the classification is changed.

Definition 4.2.1. *Let $F : \mathcal{X} \to \mathcal{Y}$ (with $\mathcal{Y}$ finite) be a classifier over the normed vector*

space $(\mathcal{X}, \|\cdot\|)$. *We call*

$$\rho(x) = \inf_{\delta \in \mathcal{X}} \{\|\delta\| : F(x+\delta) \neq F(x)\} \tag{4.1}$$

the (adversarial) robustness of F in the point x.

Put differently, the robustness of a classifier in a point is nothing but the distance to its closest decision boundary. Margin classifiers like support vector machines (Cortes and Vapnik, 1995) seek to keep this distance large for the training set, usually in order to avoid overfitting. (Sokolić et al., 2017) and (Elsayed et al., 2018) also apply this principle to neural networks via regularization schemes. We point out that our definition of adversarial robustness does not depend on the ground truth class label and – given feasible computability – can approximately be calculated even on unlabelled data. Furthermore, it should be noted that from an application standpoint, one is typically less interested in the robustness *in a point*, but rather in statistical quantities such as the average robustness of a classifier over a dataset.

For the rest of the chapter, we will always assume $\mathcal{X} = \mathbb{R}^n$ and $\|\cdot\|$ to be the 2-norm.

4.2.1 A Motivating Toy Example

We consider the toy case of a linear binary classifier $F(x) = \operatorname{sgn}(\Psi(x))$ with the so-called score function $\Psi(x) = \langle x, z\rangle$ and fixed $z \neq 0$, where $\langle\cdot,\cdot\rangle$ denotes the standard inner product on $\mathbb{R}^n$.

The adversarial robustness is then given by the distance of x to the separating hyperplane $H = \{p \in \mathcal{X} \mid \langle p, z\rangle = 0\}$. According to the projection theorem, this distance is given by the distance between x and its orthogonal projection onto H, given by

$$P_H(x) = x - \frac{\langle x, z\rangle \cdot z}{\|z\|^2},$$

which yields

$$\rho(x) = \frac{|\langle x, z\rangle|}{\|z\|} = \frac{|\langle x, \nabla\Psi(x)\rangle|}{\|\nabla\Psi(x)\|}.$$

Unless stated otherwise, in this chapter we will always denote with ∇ the gradient with respect to x. Note that $\rho(x) = \|x\| \cdot |\cos(\omega)|$, where ω is the angle between the vectors x and $\nabla\Psi(x)$. This implies that $\rho(x)$ grows with the alignment of x and z and is maximized if and only if x and z are collinear.
This motivates the following definition.

Definition 4.2.2 (Alignment)**.** *Let the binary classifier*

$$F : \mathcal{X} \to \{-1, 1\}$$

be defined a.e. by $F(x) = \operatorname{sgn}(\Psi(x))$, *where* $\Psi : X \to \mathbb{R}$ *is differentiable in* x. *We then call* $\nabla\Psi$ *the* saliency map of F with respect to Ψ in x *and*

$$\alpha(x) := \frac{|\langle x, \nabla\Psi(x)\rangle|}{\|\nabla\Psi(x)\|},$$

the alignment with respect to Ψ *in* x.

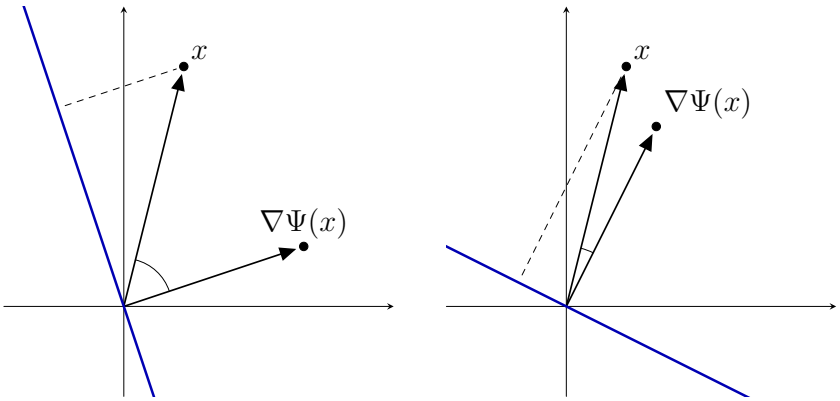


Figure 4.2: In the case of a binary, linear classifier, the alignment between x and the saliency map $\nabla\Psi(x)$ naturally increases with the robustness.

The alignment is a measure of how similar the input image x and the saliency map $\nabla\Psi(x)$ are. If $\|x\| = 1$, and x and $\nabla\Psi(x)$ are zero-centered, this coincides with the absolute value of their Pearson correlation. For a linear binary classifier, the alignment – by construction – increases with the robustness of the classifier.

Generalizing from the linear to the affine case leads to a classifier of the form $F(x) = \operatorname{sgn}(\langle x, z\rangle + b)$, whose robustness in x is

$$\rho(x) = \frac{|\langle x, z\rangle + b|}{\|z\|}.$$

In this case the robustness and alignment do not coincide anymore. In order to connect these two diverging concepts, we offer two alternative viewpoints. On the one hand, we can trivially bound the robustness via the triangle inequality

$$\rho(x) \leq \alpha(x) + \frac{|b|}{\|z\|}. \tag{4.2}$$

This is particularly meaningful if $|b|/\|z\|$ is small in comparison to $\alpha(x)$. Alternatively, one can connect the robustness to the alignment at a different point

$$\xi := x + \frac{b}{\|z\|}\frac{z}{\|z\|},$$

leading to the relation

$$\rho(x) = \alpha(\xi). \tag{4.3}$$

In the affine case this approach simply amounts to a shift of the data that is uniform over all data points x. We will see how these two viewpoints lead to different bounds in the non-linear case later.

4.2.2 The General Case

We now consider the general, C-class case.

Definition 4.2.3 (Alignment, Multi-Class Case)**.** *Let*

$$\Psi = (\Psi^1, \dots, \Psi^C) : \mathcal{X} \to \mathbb{R}^C$$

be differentiable in x. Then for an C-class classifier defined a.e. by

$$F(x) = \arg\max_i \Psi^i(x), \tag{4.4}$$

we call $\nabla \Psi^{F(x)}$ the saliency map of F*. We further call*

$$\alpha(x) := \frac{|\langle x, \nabla \Psi^{F(x)}(x) \rangle|}{\|\nabla \Psi^{F(x)}(x)\|}, \tag{4.5}$$

the alignment with respect to Ψ in x*.*

Linearized Robustness

In general the distance to the decision boundary $\rho(x)$ can be unfeasible to compute. However, for classifiers built on locally affine score functions – such as most neural networks using ReLU or leaky ReLU activations – $\rho(x)$ can easily be computed, provided the locally affine region is sufficiently large. To quantify this, define the radius of the locally affine component of F around x as

$$l(x) := \sup\{r \mid \forall i : \Psi^i \text{ affine in } B_r(x)\},$$

where $B_r(x)$ is the open ball of radius r around x with respect to the Euclidean metric.

Lemma 4.2.1. *Let F be a classifier with locally affine score function Ψ. Assume $l(x) \geq \rho(x)$. Then*

$$\rho(x) = \min_{j \neq i^*} \frac{\Psi^{i^*}(x) - \Psi^j(x)}{\|\nabla \Psi^{i^*}(x) - \nabla \Psi^j(x)\|},$$

for $i^* := F(x)$ the predicted class at x.

Proof. As $l(x) \geq \rho(x)$, we can take the infimum in (4.1) over all perturbations in the local affine component, i.e. δ with $\|\delta\| \leq l(x)$ only. This allows us to reformulate

$$\begin{aligned} & F(x+\delta) \neq F(x) \\ \Leftrightarrow\ & \exists j \neq i^* : \Psi^j(x+\delta) > \Psi^{i^*}(x+\delta) \\ \Leftrightarrow\ & \exists j \neq i^* : \langle \nabla \Psi^j(x) - \nabla \Psi^{i^*}(x), \delta \rangle > \Psi^{i^*}(x) - \Psi^j(x). \end{aligned}$$

The infimum over $\|\delta\|$ is achieved by choosing δ as a multiple of $\nabla \Psi^j(x) - \nabla \Psi^{i^*}(x)$. A direct computation then finishes the proof. □

Similar identities were previously also independently derived in (Hein and Andriushchenko, 2017), (Elsayed et al., 2018) and (Jakubovitz and Giryes, 2018).

Note that while nearly all state-of-the art classification networks are piecewise affine, the condition $l(x) \geq \rho(x)$ is typically violated in practice. However, the lemma can still hold *approximately* as long as the linear approximation to the network's score functions is sufficiently good in the relevant neighbourhood of x. This motivates the definition of the *linearized (adversarial) robustness* $\tilde{\rho}$.

Definition 4.2.4 (Linearized Robustness)**.** *Let $\Psi(x)$ be the differentiable score vector for the classifier F in x. We call*

$$\tilde{\rho}(x) := \min_{j \neq i^*} \frac{\Psi^{i^*}(x) - \Psi^j(x)}{\|\nabla\Psi^{i^*}(x) - \nabla\Psi^j(x)\|}, \tag{4.6}$$

the linearized robustness *in x, where $i^* := F(x)$ is the predicted class at point x.*

We later show that the two notions lead to very similar results, even if the condition $l(x) \geq \rho(x)$ is violated.

Reducing the Multi-Class Case

In this section, we introduce a toolset which helps bridge the gap between the alignment and the linearized robustness of a multi-class classifier.

Definition 4.2.5 (Classifier binarization)**.** *Let F be a classifier as given in Definition 4.2.3. For fixed $x \in \mathbb{R}^n$, let $i^* := F(x)$ and j^* be the minimizer in* (4.6)*. We define the* binarized classifier $F_x^\dagger$ *of F in x by*

$$F_x^\dagger(y) := \operatorname{sgn}\left(\Psi_x^\dagger(y)\right),$$

where $\Psi_x^\dagger(y) := \Psi^{i^*}(y) - \Psi^{j^*}(y)$. We call the saliency map of $F_x^\dagger$ in x (according to Definition 4.2.2), given by $\nabla\Psi_x^\dagger(x) = \nabla_y \Psi_x^\dagger(y)|_{y=x}$, the binarized saliency map *of F. Furthermore, the respective alignment of $F_x^\dagger$, given by*

$$\alpha^\dagger(x) = \frac{|\langle x, \nabla(\Psi^{i^*} - \Psi^{j^*})(x)\rangle|}{\|\nabla(\Psi^{i^*} - \Psi^{j^*})(x)\|},$$

is called the binarized alignment *of F in x.*

Note that the linearized robustness of $F_x^\dagger$ in x is the same as for F. The technique of binarization offers an alternative, natural perspective of our considerations about robustness and alignment. This is because for classifiers as defined in (4.4), the actual score values do not necessarily carry any information about the classification decision, whereas the score differences do. While, roughly speaking, $\nabla\Psi^{i^*}$ tells us what F 'thinks' makes x a member of its predicted class, $\nabla\Psi_x^\dagger(x)$ carries information what sets x apart from its closest neighboring class (according to linearization(!)).

We point out that there are actually two different, possible definitions for binary classifiers and their saliency maps and alignments in this chapter, which can be unified using said binarization principle. On the one hand, Definition 4.2.3 defines a binary classifier via

$$F : x \mapsto \arg\max_{i \in \{1,2\}} \Psi^i(x).$$

When applying Definition 4.2.2 on the other hand, a binary classifier G is defined by

$$G : x \mapsto \operatorname{sgn}(\Gamma(x)),$$

where Γ is a scalar-valued function. While F maps to classes from $\{1, 2\}$, G maps to classes from $\{1, -1\}$.
It is always possible to convert each of these two classifiers into the respective other one by e.g. defining either $\Psi^i(x) := c \cdot (-1)^i \Gamma(x)$ or conversely, $\Gamma(x) := c \cdot (\Psi^1(x) - \Psi^2(x))$,

where c is either 1 or -1, depending on how the classes are encoded between these two classifiers. While the robustnesses of both classifiers are naturally the same in each point, the same is not true for their alignments. However, for F and G constructed this way, the alignment of G coincides with the binarized alignment of F. When applying this idea to linear, binary classifiers, we can always define these via $\Psi(x) = (\langle x, z\rangle, -\langle x, z\rangle)$ for some non-zero $z \in \mathbb{R}^n$, such that

$$\alpha(x) = \alpha^\dagger(x) = \rho^\dagger(x) = \rho(x)$$

for almost all $x \in \mathbb{R}^n$ in this case. This highlights that the Definition 4.2.3 of the alignment α for multi-class problems *as well as* the binarized alignment $\alpha^\dagger$ from Definition 4.2.5 are in some sense strict generalizations of the alignment originally defined in 4.2.2.

So how do these distinct concepts diverge for more general models than the toy example of binary, linear classifiers? For linear, multi-class classifier, it still holds that

$$\rho(x) = \tilde{\rho}(x) = \alpha^\dagger(x),$$

for almost all x, according to Lemma 4.2.1 (when considering linear functions Ψ). A further generalization to nonlinear score functions Ψ will generally additionally yield

$$\rho(x) \neq \tilde{\rho}(x).$$

In other words, for the simple toy case of binary, linear classifiers all of the different concepts coincide, whereas they may diverge for more general classifiers. In the following, we will try to quantify connections between these, which hold even in very general models.

4.3 Decompositions and Bounds for Neural Networks

4.3.1 Homogeneous Decomposition

In the previous section we have seen that in the case of binary classifiers, the robustness and binarized alignment coincide for linear score functions. However, requiring Ψ to be linear is a stronger assumption than necessary to deduce the result: It is in fact sufficient for Ψ to be *positive one-homogeneous*. Any such function satisfies $\Psi(ax) = a\Psi(x)$ for all $a > 0$ and x.

Lemma 4.3.1 (Euler's Homogeneous Function Theorem)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a positive one-homogeneous function that is continuously differentiable on $\mathbb{R}^n \backslash \{0\}$. Then*

$$f(x) = \langle \nabla f(x), x\rangle$$

Proof. First note that

$$\begin{aligned}\partial_i f(ax) &= \lim_{t\to 0} \frac{f(ax + te_i) - f(ax)}{t} \\ &= \lim_{t\to 0} \frac{f(ax + ate_i) - f(ax)}{at} = \partial_i f(x).\end{aligned}$$

Hence

$$f(x) = \int_0^1 \langle \nabla f(tx), x\rangle \, \mathrm{dt} = \langle \nabla f(x), x\rangle$$

□

Corollary 4.3.2. *Consider a classifier F with positive one-homogeneous score functions. Then*

$$\tilde{\rho}(x) = \alpha^{\dagger}(x)$$

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

Proof. Direct consequence of 4.3.1. □

In particular, most feedforward neural networks with (leaky) ReLU activations *without biases* are positive one-homogeneous. This observation motivates decomposing neural networks into a homogeneous term and the corresponding remainder, leading to the following decomposition result.

Definition 4.3.1 (Neural networks of class $\mathcal{N}$)**.** *Define the class of neural networks $\mathcal{N}$ to be any network built on learnable affine transforms (convolutional layers, dense layers) with linear weights θ and biases b and positive one-homogeneous activation functions. The network can include arbitrary skip-connections, batch-normalization layers and max or average pooling layers of arbitrary window size.*

This class of neural network includes many state-of-the-art classification networks.

Lemma 4.3.3 (Homogenized Networks)**.** *For fixed x, consider the logit $\Psi^i_{\theta,b}(x)$ of a network $\Psi_{\theta,b} \in \mathcal{N}$, where θ denotes the linear weights and b is the compiled bias vector of the network. Then the function*

$$f : y \mapsto \Psi^i_{\theta, b\frac{\|y\|}{\|x\|}}(y),$$

is positive one-homogeneous and it holds that $f(x) = \Psi^i_{\theta,b}(x)$.

Proof. Consider first a network consisting of a single layer with linear transform A and bias b with ReLU non-linearity. The associated network function is hence given by $\Psi_{A,b}(x) = \varphi(Ax+b)$, where φ is a positive one-homogeneous function, e.g. (leaky) ReLU. For this network, we compute for x fixed and any y and $a > 0$ as

$$\begin{aligned} f(ay) &= \varphi\left(A(ay) + b\frac{\|ay\|}{\|x\|}\right) \\ &= \varphi\left(a \cdot Ay + a \cdot b\frac{\|y\|}{\|x\|}\right) = af(y). \end{aligned}$$

A single layer is hence positive one-homogeneous. A function consisting of compositions of positive one-homogeneous functions is positive one-homogeneous itself as well, the function f associated to a network consisting of affine transforms and ReLU activations is positive one-homogeneous. All of the operations skip-connections, batch-normalization layers and max or average pooling are positive one-homogeneous as well, thus proving the claim. □

Theorem 4.3.4. *Let $\Psi^i_{\theta,b}$ be any logit of a neural network of class $\mathcal{N}$. Denote by θ the linear filters and by b the bias terms of the network. Then*

$$\begin{aligned} \Psi^i_{\theta,b}(x) &= \langle x, \nabla_x \Psi^i_{\theta,b}(x)\rangle + \langle b, \nabla_b \Psi^i_{\theta,b}(x)\rangle \\ &= \langle x, \nabla_x \Psi^i_{\theta,b}(x)\rangle + \sum_k b_k \partial_{b_k} \Psi^i_{\theta,b}(x), \end{aligned}$$

given all derivatives exist.

Proof. Let f be the functions associated with the network $\Psi^i_{\theta,b}$ as in Lemma 4.3.3. Then by Lemma 4.3.1 we can compute the value of f at the point x via

$$f(x) = \langle x, \nabla_y f(y)|_{y=x} \rangle.$$

Note that by construction $f(x) = \Psi^i_{\theta,b}(x)$. We compute the gradient of f at the point x explicitly as

$$\nabla_y f(y)|_{y=x} = \nabla_x \Psi^i_{\theta,b}(x) + \frac{x}{\|x\|^2} \langle b, \nabla_b \Psi^i_{\theta,b}(x) \rangle.$$

Combining these results shows

$$\begin{aligned} f(x) &= \langle x, \nabla_x \Psi^i_{\theta,b}(x) + \frac{x}{\|x\|^2} \langle b, \nabla_b \Psi^i_{\theta,b}(x) \rangle \rangle \\ &= \langle x, \nabla_x \Psi^i_{\theta,b}(x) \rangle + \langle b, \nabla_b \Psi^i_{\theta,b}(x) \rangle, \end{aligned}$$

proving the statement. □

Note that the above vector b includes the running averages of the means for batch normalization. For (leaky) ReLU networks, the remainder term $\beta^i(x) := \langle b, \nabla_b \Psi^i_{\theta,b}(x) \rangle$ is locally constant, because it changes only when x enters another locally affine region. For such locally affine score functions $\Psi_{\theta,b}$, the homogeneous decomposition thus simply reduces to the Taylor decomposition. For ease of notation, we will now again drop the subscripts θ and b.

4.3.2 Pointwise Bounds

In section 4.2.1, we introduced two different viewpoints for locally affine, binary classifiers which connect the robustness to the alignment. In a similar vein to inequality (4.2) and equality (4.3), upper bounds to the linearized robustness depending on the alignment can be given for neural networks. In the following, we will write $\overline{v} := v/\|v\|$ for $v \neq 0$. Again, in the following we fix x and write $i^* = F(x)$ and j^* for the minimizer in j from equation (4.6).

Theorem 4.3.5. *Let $g := \nabla \Psi^{i^*}(x)$. Furthermore, let $g^\dagger := \nabla(\Psi^{i^*} - \Psi^{j^*})(x)$ and $\beta^\dagger := \beta^{i^*}(x) - \beta^{j^*}(x)$. Then*

$$\begin{aligned} \tilde{\rho}(x) &\leq \alpha^\dagger(x) + \frac{|\beta^\dagger|}{\|g^\dagger\|} \\ &\leq \alpha(x) + \|x\| \cdot \|\overline{g}^\dagger - \overline{g}\| + \frac{|\beta^\dagger|}{\|g^\dagger\|} \end{aligned} \tag{4.7}$$

Proof. We have

$$\begin{aligned} \tilde{\rho}(x) &= \frac{\Psi^{i^*}(x) - \Psi^{j^*}(x)}{\|\nabla \Psi^{i^*}(x) - \nabla \Psi^{j^*}(x)\|} \\ &= \frac{\langle x, \nabla \Psi^{i^*}(x) - \nabla \Psi^{j^*}(x) \rangle + \beta^{i^*}(x) - \beta^{j^*}(x)}{\|\nabla \Psi^{i^*}(x) - \nabla \Psi^{j^*}(x)\|} \\ &= \left| \langle x, \overline{g}^\dagger \rangle + \frac{\beta^\dagger}{\|g^\dagger\|} \right| \leq \alpha^\dagger(x) + \frac{|b^\dagger|}{\|g^\dagger\|}, \end{aligned}$$

using the decomposition theorem and the triangle inequality. Further,

$$\begin{aligned}
&\alpha^\dagger(x) + \frac{|b^\dagger|}{\|g^\dagger\|} \\
&= \left|\langle x, \overline{g}^\dagger\rangle\right| + \frac{|b^\dagger|}{\|g^\dagger\|} \\
&= \left|\langle x, \overline{g}^\dagger - \overline{g} + \overline{g}\rangle\right| + \frac{|b^\dagger|}{\|g^\dagger\|} \\
&\leq |\langle x, \overline{g}\rangle| + \left|\langle x, \overline{g}^\dagger - \overline{g}\rangle\right| + \frac{|b^\dagger|}{\|g^\dagger\|} \\
&\leq \alpha(x) + \|x\| \cdot \|\overline{g}^\dagger - \overline{g}\| + \frac{|b^\dagger|}{\|g^\dagger\|},
\end{aligned}$$

using the Cauchy-Schwarz inequality. □

Distances on the unit sphere (such as $\|\overline{g}^\dagger - \overline{g}\|$) can be converted to angles through the law of cosines. For the above inequalities to be reasonably tight, the angle between g and $g^\dagger$ needs to be small and $|\beta^\dagger|/\|g^\dagger\|$ needs to be small in comparison to $\alpha^\dagger(x)$. In this case, the alignment should roughly increase with the linearized robustness.

Theorem 4.3.6. *Let* $\xi := x + \frac{\beta^\dagger}{\|g^\dagger\|}\frac{g^\dagger}{\|g^\dagger\|}$ *and* $\gamma := \nabla\Psi^{i^*}(\xi)$*, with* $g^\dagger$ *and* $\beta^\dagger$ *defined as in the previous theorem. Then*

$$\tilde{\rho}(x) \leq \frac{|\langle \xi, \gamma\rangle|}{\|\gamma\|} + \|\xi\| \cdot \|\overline{g}^\dagger - \overline{\gamma}\|, \tag{4.8}$$

and if additionally $F(x) = F(\xi)$*, then*

$$\tilde{\rho}(x) \leq \alpha(\xi) + \|\xi\| \cdot \|\overline{g}^\dagger - \overline{\gamma}\|.$$

Proof. We have

$$\begin{aligned}
\tilde{\rho}(x) &= \frac{\langle x, g^\dagger\rangle + \beta^\dagger\langle \frac{g^\dagger}{\|g^\dagger\|^2}, g^\dagger\rangle}{\|g^\dagger\|} \\
&= \frac{\langle x + \frac{\beta^\dagger}{\|g^\dagger\|}\frac{g^\dagger}{\|g^\dagger\|}, g^\dagger\rangle}{\|g^\dagger\|} \\
&= \langle \xi, \overline{g}^\dagger\rangle = \langle \xi, \overline{g}^\dagger - \overline{g} + \overline{g}\rangle \\
&\leq |\langle \xi, \overline{\gamma}\rangle| + \|\xi\| \cdot \|\overline{g}^\dagger - \overline{\gamma}\|,
\end{aligned}$$

using the Cauchy-Schwarz inequality in the same way as in the last theorem. □

Depending on the sign of $\beta^\dagger$, the shifted image ξ can either be understood as a gradient ascent or descent iterate for maximizing/minimizing $\Psi^{i^*} - \Psi^{j^*}$. This theorem assimilates $\beta^\dagger(x)$ into x, providing an upper bound to $\tilde{\rho}(x)$ that depends on $\alpha(\xi)$. The sensibility of this hinges on ξ being reasonably close to x and γ having a low angle with $g^\dagger$.

If the error terms in inequalities (4.7) and (4.8) are small, these inequalities thus provide a simple illustration why more robust networks yield more interpretable saliency maps.

Nevertheless, the right-hand side may be much larger than $\tilde{\rho}(x)$, if the inner product between an image and its respective saliency map are almost orthogonal. This is because the Cauchy-Schwarz inequality provides a large upper bound in this case. The inequalities rather serve as an explanation of how the various terms of alignment may deviate from the linearized robustness in the case of a neural network.

4.3.3 Alignment and Interpretability

The above considerations demonstrate how an increase in robustness may induce an increase in the alignment between an input image and its respective saliency map. The initial observation – which was previously described as an increase in *interpretability* – may thus be ascribed to this phenomenon. This is especially true in the case of natural images, as exemplified in Figure 4.1. There, what a human observer would deem an increase in interpretability, expresses itself as discriminative portions of the original image reappearing in the saliency map, which naturally implies a stronger alignment. The concepts of alignment and interpretability should however not be conflated completely: In the case of quasi-binary image data like MNIST, 0-regions of the image render the inner product in equation (4.5) invariant with respect to the saliency map in this region, even if the saliency map e.g. assigns relevance to the absence of a feature in this region. Note however that the saliency map in this region still influences the alignment term through the division by its norm. Additionally, the alignment is also not invariant to the images' representation (color space, shifts, normalization etc.). Still, for most types of image data an increase in alignment in discriminative regions should coincide with an increase in interpretability.

4.4 Experiments

The goal of this section is to validate our hypothesis of the connection between adversarial robustness and image-gradient-alignment. This is achieved by training models with varied adversarial robustness on both MNIST (LeCun et al., 1990) and ImageNet (Deng et al., 2009) using double backpropagation (Drucker and Le Cun, 1992).

As presented in the previous chapter 3, for a neural network f_Θ with a softmax output layer, this amounts to minimizing the modified loss

$$\frac{1}{N}\sum_{i=1}^{N}\left[\ell(f_\Theta(x^{(i)}), y^{(i)}) + \lambda \cdot \|\nabla \ell(f_\Theta(x^{(i)}), y^{(i)})\|^2\right]$$

over the parameters Θ. Here, $\{(x^{(i)}, y^{(i)})\}_{i=1,\dots,N}$ is the training set and ℓ denotes the negative log-likelihood error function. The hyperparameter $\lambda \geq 0$ determines the strength of the regularization. Note that this penalizes the local Lipschitz constant of the loss. As (Simon-Gabriel et al., 2018) demonstrate, double backpropagation makes neural networks more resilient to adversarial attacks. The reasoning is that for $\mathcal{L}(x) := \ell(f_\Theta(x), y)$, a Taylor expansion shows that approximately

$$\sup_{\delta:\|\delta\|\leq\varepsilon} |\mathcal{L}(x+\delta) - \mathcal{L}(x)| \approx \sup_{\delta:\|\delta\|\leq\varepsilon} |\langle \delta, \nabla\mathcal{L}(x)\rangle| = \varepsilon\|\nabla\mathcal{L}(x)\|, \tag{4.9}$$

such that the increase of the loss function can approximately be related to the norm of the loss gradient. By keeping this norm small, the robustness to adversarial attacks

Table 4.1: Architecture of the neural network used for training the MNIST model. Dropout refers to the regularization technique presented in (Srivastava et al., 2014).

Layer	Kernel size	Stride	Size
Input layer	-	-	$28 \times 28 \times 1$
Conv layer (ReLU)	3	1	$28 \times 28 \times 32$
Max Pooling	-	2	$14 \times 14 \times 32$
Conv layer (ReLU)	3	1	$14 \times 14 \times 64$
Max Pooling	-	2	$7 \times 7 \times 64$
Conv layer (ReLU)	3	1	$7 \times 7 \times 128$
Max Pooling	-	2	$3 \times 3 \times 128$
Dense layer (ReLU)	-	-	128
Dropout	-	-	128
Dense layer (softmax)	-	-	10

should increase. It should be pointed out that, however, this is only valid up to first-order approximation and does not take non-linear effects into account. Note that (4.9) uses the fact that the 2-norm is dual to itself.

By varying λ, we can easily create models of different adversarial robustness for the same dataset, whose properties we can then compare. (Anil et al., 2018) previously noted that Lipschitz constrained networks exhibit 'interpretable' saliency maps (without an explanation), which can be regarded as a side-effect of the increase in adversarial robustness.

For the MNIST experiments, we trained each of our 16 models on an NVIDIA 1080Ti GPU with a batch size of 100 for 200 epochs, covering the regularization hyperparameter range from 10 to 180,000, before the models start to degenerate. The used architecture is found in Table 4.1.
For the experiments on ImageNet, we fine-tuned the pre-trained ResNet50 model from (He et al., 2016) over 35 epochs on 2 NVIDIA P100 GPUs with a total batch size of 32. We used stochastic gradient descent with a learning rate of 0.0001 and momentum of 0.99. The learning rate was divided by 10 whenever the error stopped improving. For the regularization parameter, we chose $\lambda = 10^4, 10^{4.5}, \ldots, 10^7$ for the individual models. The experiments were implemented in Tensorflow (Abadi et al., 2015).

4.4.1 Robustness and Alignment

For checking the relation between the alignment and robustness of a neural network, we created 1000 adversarial examples per model on the respective validation set. This was realized using the python library `Foolbox` (Rauber et al., 2017), which offers predefined adversarial attacks, three of which we used in this work. In the following, let $D = [a_1, b_1] \times \cdots \times [a_n, b_n]$ denote the set of admissable adversarial examples, i.e. images contained within the allowed pixel range (e.g. $D = [0, 1]^{784}$ for MNIST). Let $\mathcal{L}(x)$ denote the non-negative likelihood loss $\ell(f_\Theta(x), y)$ of image-label-pair (x, y).

1. The `GradientAttack` performs a line search for the closest adversarial example in Euclidean metric along the direction of the loss gradient. In other words, one searches for the smallest $\tau > 0$ such that $\tilde{x}_\tau := x + \tau \cdot \nabla\mathcal{L}(x)$ is contained in $\mathcal{D}$ and $F(x) \neq F(\tilde{x}_\tau)$, where x is the original, unperturbed image.

2. The `L2BasicInterativeAttack` implements the projected gradient (ascent) attack from (Kurakin et al., 2016) for the Euclidean metric. Let

$$P_D : x \mapsto \arg\min_{y \in D} \|x - y\|_2$$

be the projection onto D. For the box constraints of images, this simply amounts to clipping the coordinates to their admissable range. Then

$$\tilde{x}_k = P_D(\tilde{x}_{k-1} + \tau \cdot \nabla \mathcal{L}(\tilde{x}_{k-1}))$$

is the k-th iteration of the projected gradient attack.

3. The `CarliniWagnerL2Attack` (CW-attack) is the attack introduced in (Carlini and Wagner, 2017) suited for finding the closest adversarial example in Euclidean distance. For notational simplicity, we assume that the images' pixel range is in $[0, 1]$, such that $D = [0, 1]^n$. It is straightforward to generalize this method to general box constraints by applying an affine transformation. The idea of the CW-attack is to relax the optimization task of finding the closest adversarial example, i.e.

$$\begin{aligned} \min_{\tilde{x}} \quad & \|x - \tilde{x}\|_2^2 \\ \text{s.t.} \quad & \tilde{x} \in D \\ & F(\tilde{x}) = t, \end{aligned}$$

where t is a target class (in `Foolbox`'s implementation, this is the class of the second-highest logit value[1]). To construct this relaxation, a function s is chosen such that $s(\tilde{x}) \leq 0$ if and only if $F(\tilde{x}) = t$. Furthermore, the constraint $\tilde{x} \in [0, 1]^n$ is met by parametrizing $\tilde{x}_w := \frac{1}{2}(\tanh(w) + 1) \in (0, 1)^n \subset D$, such that the constraints are automatically met when optimizing over $w \in \mathbb{R}^n$. This constrained optimization problem is then posed as a penalized, unconstrained optimization problem

$$\min_w \|x - \tilde{x}_w\|_2^2 + \mu \cdot s(\tilde{x}_w) \tag{4.10}$$

over w, for some value of $\mu > 0$. The utilized function s is here given by

$$s(\tilde{x}_w) = \max_i \text{ReLU}(\Psi^i(\tilde{x}_w) - \Psi^t(\tilde{x}_w)),$$

which is 0 if and only if $F(\tilde{x}_w) = t$ (ignoring the edge case of equal logits). The optimal parameter μ is determined to be the one which yields the closest adversarial example after solving (4.10) with Adam (Kingma and Ba, 2014), according to a line search within a pre-determined range.

4. Additionally, we calculated the linearized robustness $\tilde{\rho}(x)$, which entails calculating C gradients per image for an C-class problem.

[1] An interesting application of Theorem 4.2.1 could be to instead choose the class with the closest decision boundary according to linearization. This, however, requires the calculation of C gradients.

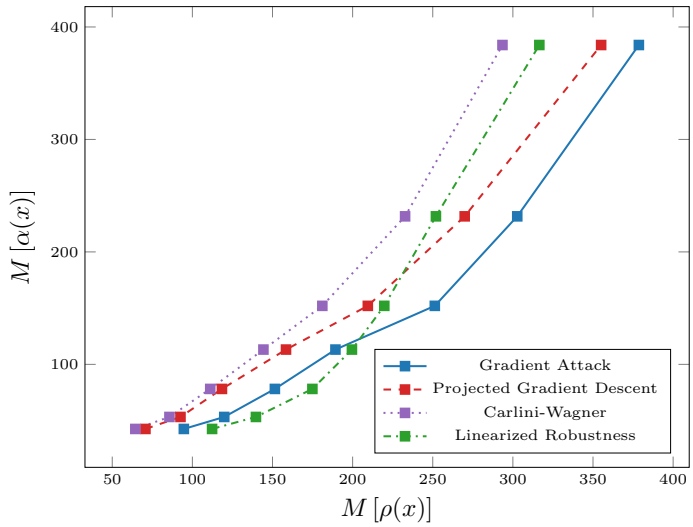


Figure 4.3: The median alignment increases with the median robustness of the model on ImageNet. Furthermore, the more elaborate attacks consistently find smaller adversarial perturbations than the simple gradient attack. The linearized robustness estimator provides a rather realistic estimation of the algorithmically calculated robustness.

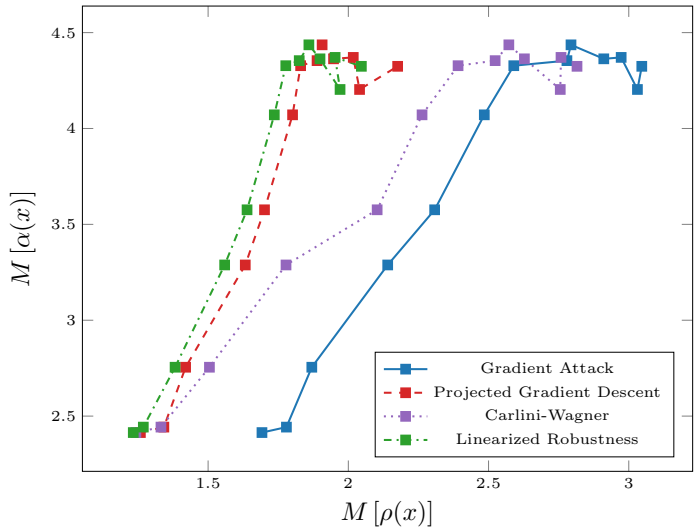


Figure 4.4: Similar to Figure 4.3, the median alignment increases with the median robustness of the model on MNIST. Towards the end, some saturation effects are visible.

In Figures 4.3 and 4.4, we investigate how the median alignment depends on the medians of the different conceptions of robustness. We opted in favor of the median (denoted M) instead of the arithmetic mean due to its increased robustness to outliers, which occurred especially when using the gradient attack. In the case of ImageNet (Figure 4.3), an increase in median alignment with the median robustness is clearly visible for all three estimates of the robustness. On the other hand, the alignment for the MNIST

data increases with the robustness as well, but seems to saturate at some point. We will offer an explanation for this phenomenon later.

We now consider the pointwise connection between robustness and alignment. In Figure 4.5 the two variables are highly-correlated for a model trained on MNIST, pointing towards the fact that the network behaves very similarly to a positive one-homogeneous function. There is however no visible correlation between them on the ImageNet model, which is a consistent behavior throughout the whole experiment cohort. We will later analyse the source of this behavior. The increase in median alignment for ImageNet, $M\left[\alpha(x)\right] = M\left[|\langle x, \overline{g}\rangle|\right]$, can still be explained by a statistical argument: If $M\left[\langle x, \overline{g}\rangle\right] = 0$, as approximately true in our ImageNet model, then $M\left[\alpha(x)\right]$ is the median absolute deviation of $\langle x, \overline{g}\rangle$. In other words, the graph for ImageNet in Figure 4.5 depicts the dispersion of $\langle x, \overline{g}\rangle$. The above observations also hold well for the binarized alignment. In Figure 4.6 a tight correlation between $\tilde{\rho}(x)$ and $\rho(x)$ becomes evident. Here, the latter has been calculated using the CW-attack. The linearized robustness model $\tilde{\rho}$ is hence an adequate approximation of the actual robustness ρ, even for the highly non-linear neural network models used on ImageNet. Finally note that all used attacks lead to the same general behavior of all quantities investigated (see Figures 4.3 and 4.4).

4.4.2 Explaining the Observations

In the last section, we observed some commonalities between the experiments on ImageNet and MNIST, but also some very different behaviors. In particular, two aspects stand out: Why does the median alignment steadily increase for the observed ImageNet experiments, whereas on MNIST this stagnates at some point (Figures 4.3 and 4.4)? Furthermore, why are $\tilde{\rho}(x)$ and $\alpha(x)$ so highly-correlated on MNIST but almost uncorrelated on ImageNet (Figure 4.5)? We turn to Theorems 4.3.5 and 4.3.6 for answers.
Theorem 4.3.5 states that

$$\tilde{\rho}(x) \leq \alpha^{\dagger}(x) + \frac{|\beta^{\dagger}|}{\|g^{\dagger}\|}, \tag{4.11}$$

where $\beta^{\dagger}$ is the locally constant term and $g^{\dagger}$ is the saliency map of the binarized classifier and $\overline{v} = v/\|v\|$ for $v \neq 0$. In Figure 4.7, we check how strongly the right-hand side of inequality (4.11) is dominated by $\alpha^{\dagger}(x)$, i.e. how large the influence of the locally linear term is in comparison to the locally constant term. For ImageNet, this ratio increases from below 0.55 to almost 0.85, pointing towards a model increasingly governed by its linearized part. On MNIST, this ratio strongly decreases over the robustness's range. Note however that in the weakly regularized MNIST models, the right hand side is extremely dominated by the median alignment in the first place.

A similar analysis can be performed for the second inequality from Theorem 4.3.5,

$$\tilde{\rho}(x) \leq \alpha(x) + \|x\| \cdot \|\overline{g}^{\dagger} - \overline{g}\| + \frac{|\beta^{\dagger}|}{\|g^{\dagger}\|}, \tag{4.12}$$

which additionally makes a step from binarized alignment to (conventional) alignment.

This leads to an additional error term, making the bound significantly less tight than in the previous case. In particular, the proportion of the alignment α on the right-hand side diminishes, confirming our prediction from section 4.3.2. Nevertheless, the qualitative behaviors is similar to the previous case, with the $\alpha(x)$ taking up an increasing fraction of the right-hand with increasing robustness. For MNIST data, the ratio varies little

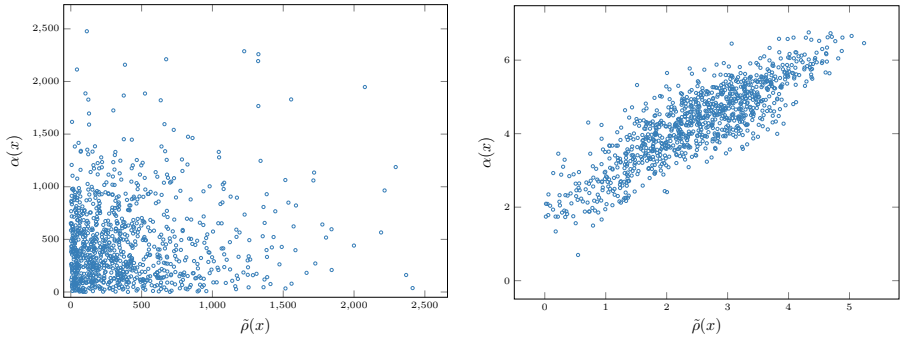


Figure 4.5: The pointwise relationship between $\tilde{\rho}(x)$ and $\alpha(x)$, exemplified on a model trained on ImageNet (left) and MNIST (right). While the two properties are well-correlated on MNIST (fitting the 'averaged' view from Figure 4.4), there is no visible correlation in the case of ImageNet.

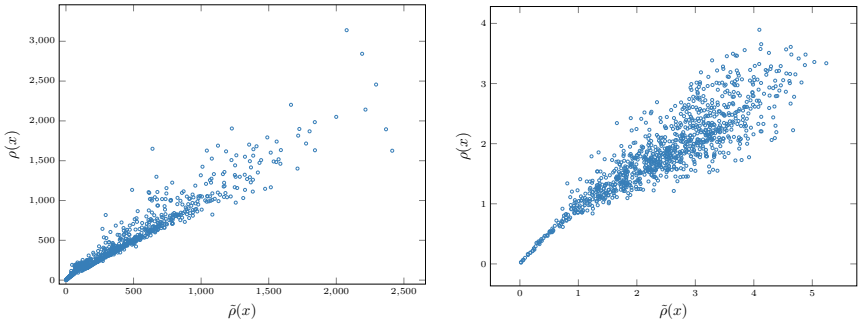


Figure 4.6: The pointwise relationship between $\tilde{\rho}(x)$ and $\rho(x)$, each calculated for 1000 validation points on a model trained on ImageNet (left) and MNIST (right). $\rho(x)$ was approximately calculated using the CW-attack. In both cases, the correlation is high.

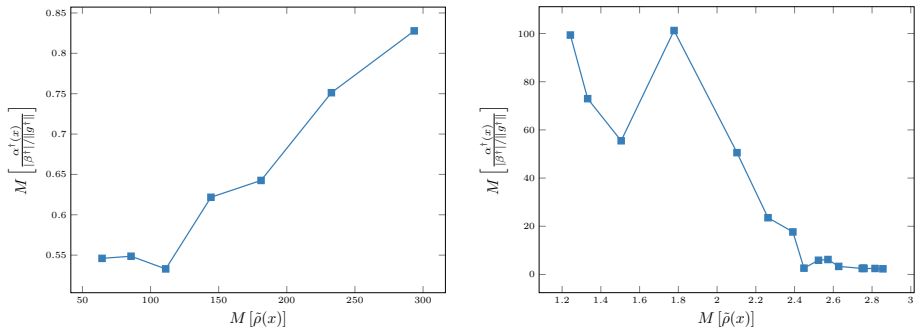


Figure 4.7: Comparing the size of the summands of inequality (4.11) for the various experiments. In the case of ImageNet (left), $\alpha^\dagger(x)$ takes up an increasing fraction of the right-hand side of the inequality. For MNIST (right), this portion tends to strongly decrease with the robustness. Note however that in this case, $\alpha^\dagger(x)$ starts out vastly dominating the right-hand side.

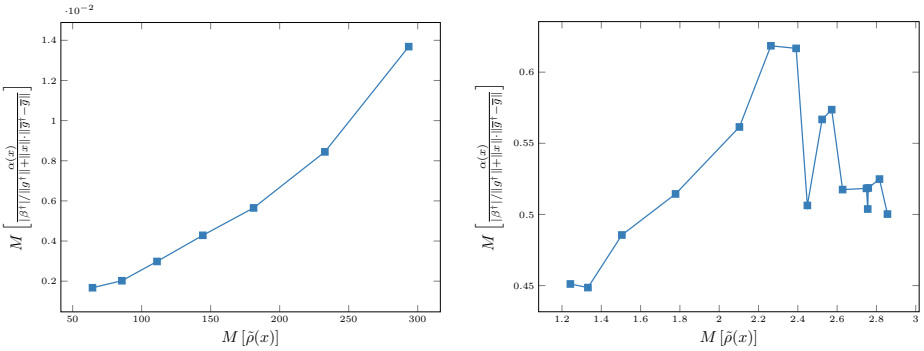


Figure 4.8: Comparing the size of the summands of inequality (4.12) for the various experiments. For the ImageNet experiments (left), the portion of $\alpha(x)$ of the right-hand side of the inequality increases roughly 7-fold. For MNIST (right), this portion stays roughly constant compared to the variation from Figure 4.7.

compared to the ratio from the last inequality. This indicates that the remainder term $\|\overline{g}^\dagger - \overline{g}\|$ does not change too strongly over the set of MNIST experiments compared to $\alpha(x)$. We thus deduce that the qualitative relationship between robustness and alignment is fully governed by the error term introduced in (4.11), i.e. the locally constant term of the logit.

We now do the same for the inequality in Theorem 4.3.6, which states that

$$\tilde{\rho}(x) \leq \frac{|\langle \xi, \gamma \rangle|}{\|\gamma\|} + \|\xi\| \cdot \|\overline{g}^\dagger - \overline{\gamma}\| \tag{4.13}$$

for $\xi = x + \frac{b^\dagger}{\|g^\dagger\|}\frac{g^\dagger}{\|g^\dagger\|}$ and $\gamma = \nabla\Psi^{F(x)}(\xi)$, which gets rid of the additive term $|\beta^\dagger|/\|g^\dagger\|$ from (4.11). Again, in the case of ImageNet $|\langle \xi, \overline{\gamma} \rangle|$ grows more quickly in comparison to $\|\overline{g}^\dagger - \overline{\gamma}\|$, the distance of the normalized gradients, whereas their ratio is approximately constant for MNIST data, see Figure 4.9.

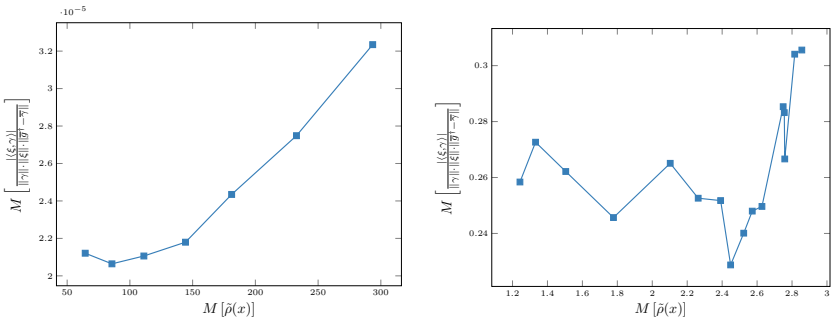


Figure 4.9: Comparing the size of the summands of inequality (4.13) for the various experiments. In the case of ImageNet (left), the alignment in ξ takes up an increasingly large portion of the right-hand side of the inequality. For MNIST (right), this portion stays roughly constant.

To conclude, we have seen that the upper bounds from Theorems 4.3.5 and 4.3.6 provide valuable information in which ways both the experiments on ImageNet and MNIST are influenced by the respective terms. In the case of ImageNet, we consistently see the alignment terms growing more quickly than the other terms. This might indicate that the growth in alignment stems not only from the growth in the robustness alone, but also from the model becoming increasingly similar to our idealized toy example. In other words, not only does the robustness make the alignment grow, but moreover the connection between these two properties becomes stronger in the case of ImageNet. This is in agreement with the seemingly superlinear growth of the median alignment in Figure 4.3.
It is not surprising that a classifier for a problem as complex as ImageNet is highly non-linear, which makes the (pointwise) connection between alignment and robustness rather loose. We hence conjecture that the imposed regularization increasingly restricts the models to be more linear (and thus more homogeneous), thereby making them more similar to our initial toy example.

For MNIST, the regularization seems to have the opposite effect: As seen in Figure 4.7, the binarized alignment initially dwarfs the correction term $|\beta^{\dagger}|/\|g^{\dagger}\|$ introduced by the locally constant portion of the binarized logit $\Psi_x^{\dagger}(x)$. As the network becomes more robust, $\Psi_x^{\dagger}(x)$ is apparently not dominated by the linear terms anymore, while the influence of the locally constant terms (i.e. $\beta^{\dagger}$) increases. This hypothesis seems sensible, considering MNIST is a very simple problem which we tackled with a comparatively shallow network. This can be expected to yield a model with a low degree of non-linearity. The penalization of the local Lipschitz constant here seems to have the effect of requiring larger locally constant terms $|\beta^{\dagger}|$, in contrast to the models trained on ImageNet.

We check the validity of these claims by tracking the median size of $|\langle x, g^{\dagger}\rangle|$ against the median size of $|\Psi_x^{\dagger}(x)|$ in Figure 4.10. On MNIST, $M\left[|\langle x, g^{\dagger}\rangle|\right]$ starts out at approximately 40% of $M\left[|\Psi_x^{\dagger}(x)|\right]$ and at the end rises to almost 100%. Note that this does not indicate that β^i is typically close to 0 for all i, just that $\beta^{\dagger}$ is, compared to $\langle x, g^{\dagger}\rangle$.
On MNIST, this ratio is close to 1 up until $M\left[\tilde{\rho}(x)\right] \approx 2.4$, when it suddenly and quickly falls below 0.5. This drop is consistent with what we see in Figure 4.4: At around the

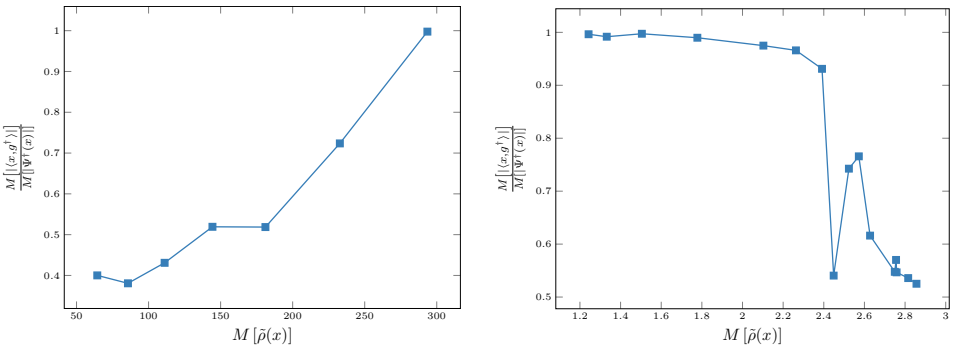


Figure 4.10: On the ImageNet experiments, the linear (i.e. homogeneous) term $|\langle x, g^\dagger\rangle|$ takes up an increasing portion of the binarized score $\Psi^\dagger(x)$. In the case of MNIST, $\Psi^\dagger(x)$ is completely dominated by the linear term, before its influence decreases sharply at $M[\tilde{\rho}(x)] \approx 2.4$.

same point this drop occurs, the alignment starts to saturate. While an increase in the model's median robustness should imply an increase in the model's median alignment, the deviation from linearity weakens the connection between robustness and alignment, such that the two effects roughly cancel out.

In Figure 4.11, we provide examples for the different gradient concepts we introduced in Theorems 4.3.5 and 4.3.6, both for the most robust and non-robust network from our experiment cohort.

4.5 Towards Explaining SmoothGrad

SmoothGrad (Smilkov et al., 2017) was mentioned in the introduction to this chapter as a method for obtaining highly structured saliency maps by averaging saliency maps under Gaussian noise. In Figure 4.12, this is exemplified for an ImageNet-trained model. In the following, we will outline how the theory presented in this chapter may explain, why these methods yield these highly 'interpretable' saliency maps. We achieve this by proving that SmoothGrad can be seen as the gradient of a certifiably adversarially robust classifier. Note that (Levine et al., 2019) consider the related question of the stability of SmoothGrad saliency maps.

4.5.1 Connecting SmoothGrad and Adversarial Robustness

In the following, let $f = \text{softmax} \circ \Psi$ be a neural network with a softmax layer. In the case of SmoothGrad, the averaged saliency map under Gaussian noise is an estimator of

$$\gamma_\sigma^i(x) := \int_{\mathbb{R}^n} \phi_\sigma(\delta) \nabla f^i(x+\delta) d\delta,$$

where ϕ_σ is the probability density function of $\mathcal{N}(0, \sigma^2 I)$. Since the partial derivatives of $f^i(x)$ are bounded (for neural networks with activation functions with bounded derivatives), one can apply Leibniz' integral rule and thus exchange the integral sign and

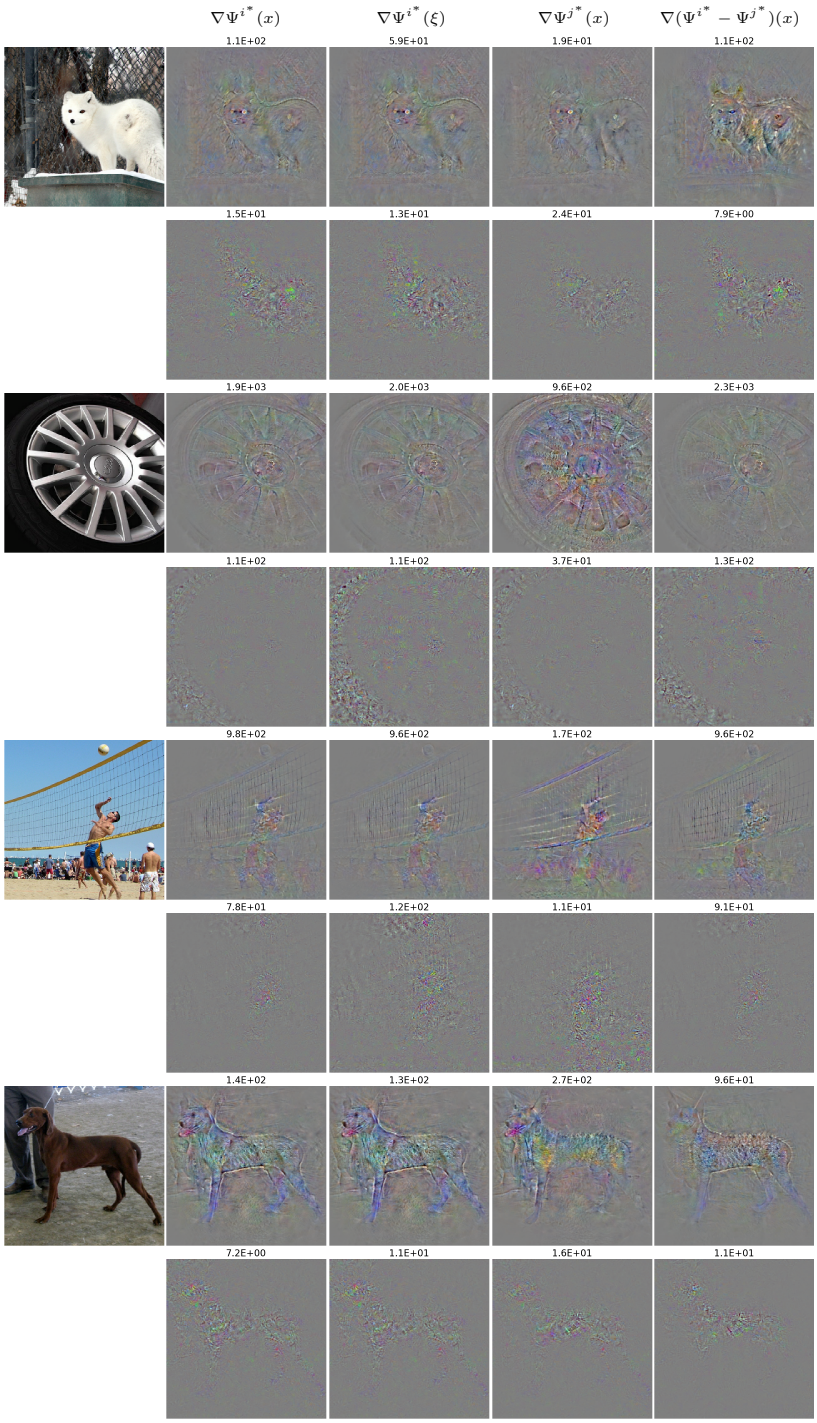


Figure 4.11: Selected examples from the ImageNet validation set of the different gradients and their respective alignments with x, respectively ξ. The odd rows are generated with the most robust ImageNet classifier, whereas the even rows are generated by the least robust classifier. The gradient images are individually scaled to fit the color range $[0, 255]$.

the gradient sign. This means that SmoothGrad is an estimator for the gradient of

$$\hat{f}^i_\sigma(x) := \int_{\mathbb{R}^n} \phi_\sigma(\delta) f^i(x+\delta) d\delta,$$

which defines a classifier $\hat{F}_\sigma(x) := \arg\max_i \hat{f}^i_\sigma(x)$, such that $\gamma^i_\sigma(x) = \nabla \hat{f}^i_\sigma(x)$. Robustifying a classifier with this technique is called *randomized smoothing* (Kaur et al., 2019; Cohen et al., 2019). The question remains, whether the thus obtained classifier $\hat{F}_\sigma$ exhibits an increased adversarial robustness.
This question is answered by (Kaur et al., 2019, Theorem 1):

Theorem 4.5.1 (Kaur, Cohen, Lipton)**.** *Let $\hat{f}_\sigma$ and $\hat{F}_\sigma$ be defined as above. Let y_1 and y_2 be the indices of the largest and second-largest entries of $\hat{f}_\sigma(x)$. Then $\hat{F}_\sigma(x+\delta) = \hat{F}_\sigma(x)(= y_1)$ for all δ with*

$$\|\delta\| \leq \frac{\sigma}{2}\left(\Phi^{-1}(\hat{f}_\sigma(x)_{y_1}) - \Phi^{-1}(\hat{f}_\sigma(x)_{y_2})\right),$$

where Φ is the cumulative Gaussian distribution function.

As a direct consequence[2] of this theorem, for the robustness of $\hat{F}_\sigma$, denoted $\hat{\rho}$, it holds that

$$\frac{\sigma}{2}\left(\Phi^{-1}(\hat{f}_\sigma(x)_{y_1}) - \Phi^{-1}(\hat{f}_\sigma(x)_{y_2})\right) \leq \hat{\rho}(x),$$

such that one can lower-bound the robustness. In other words, Theorem 4.5.1 offers a *robustness certificate*. One can thus interpret the apparent increase in 'interpretability' for SmoothGrad saliency maps as a side-effect of this increased robustness. This also explains the phenomenon pictured in Figure 4.12, where a too large value of σ starts to degrade the quality of the saliency map. It can be assumed that such a classifier is 'too robust' in the sense that it lacks the expressibility in order to capture the data characteristics – i.e. it is not a good classifier.

4.5.2 Gaps in Theory

To conclude, here we explain why the alignment between an image and its respective saliency map of a (a.e. differentiable) classifier increases, when the adversarial robustness of said classifier increases. Since SmoothGrad saliency maps are the gradients of a classifier with increased adversarial robustness (as proven in Theorem 4.5.1), the highly-structured SmoothGrad saliency maps can be seen as a consequence of this increase in adversarial robustness.
However, one should take note that the theoretical statements in this chapter applied to this setting can only hold approximately, for several reasons. Since the above randomized smoothing is applied on the (not locally affine) softmax-probabilites instead of the (locally affine) logits, there is in general no neighborhood in which the linearized robustness is exactly equal to the actual robustness, no matter how small the actual robustness is. Even if one manages to prove a similar statement to Theorem 4.5.1 for randomized smoothing of the locally affine *logit layer*, randomized smoothing acts as a mollifier and 'destroys' locally affine regions. An exception are linear classifiers, as they are invariant

[2] To see this, note that the robustness for a classifier G in a point x can equivalently be defined as the supremum of all c, such that for all δ with $\|\delta\| < c$ it holds that $G(x) = G(x+\delta)$.

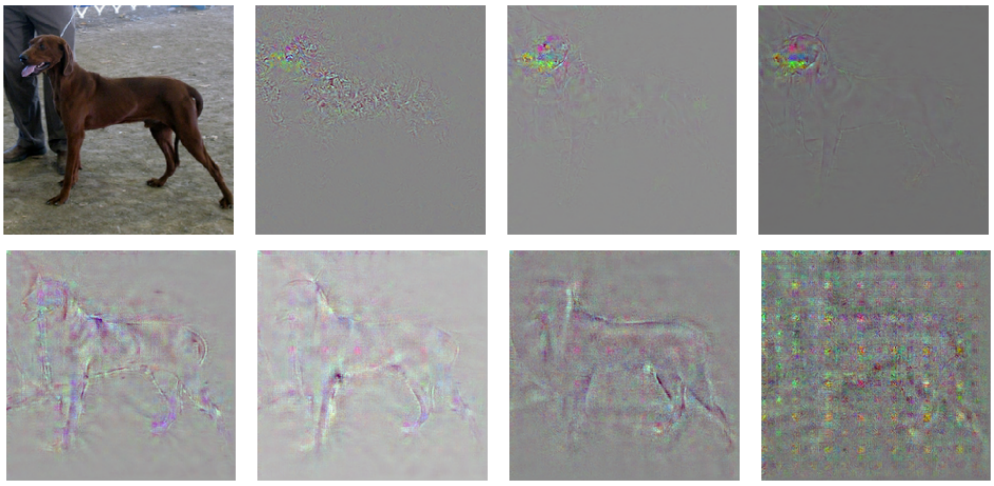


Figure 4.12: An image from the ImageNet validation set and its SmoothGrad saliency maps of a ResNet-50 (He et al., 2016) with respect to the correct class 'Redbone'. SmoothGrad was calculated based on 512 samples perturbed by Gaussian noise. The values for σ are (from left to right, top to bottom): 0, 10, 25, 50, 75, 100, 200. For $\sigma = 0$ (top row, second from left), this is equivalent to the conventional saliency map. One can observe, that the alignment apparently increases with σ, but seems to drop off when using a too high value. The saliency maps are scaled to the pixel range.

with respect to randomized smoothing (cf. Cohen et al., 2019) – but in that case there is also no increase in robustness. Moreover, Theorems 4.3.5 and 4.3.6 rely on the the homogeneous decomposition theorem (Theorem 4.3.4). This can however be remedied by instead using a first-order Taylor approximation, which should result in very similar upper bounds (up to a remainder term).

4.6 Conclusion and Outlook

In this work, we investigated the connection between a neural network's robustness to adversarial attacks and the interpretability of the resulting saliency maps. Motivated by the binary, linear case, we defined the alignment α as a measure of how much a saliency map matches its respective image. We hypothesized that the perceived increase in interpretability is due to a higher alignment and tested this hypothesis on models trained on MNIST and ImageNet. While on average, the proposed relation holds well, the connection is much less pronounced for individual points, especially on ImageNet. Using some upper bounds for the robustness of a neural network, which we derived using a decomposition theorem, we arrived at the conclusion that the strength of this connection is strongly linked with how similar to a homogeneous model the neural network is locally. As ImageNet is a comparatively complex problem, any sufficiently accurate model is bound to be very non-linear (and thus very non-homogeneous), which explains the difference to MNIST.

4.6.1 Outlook

While this work shows the general link between robustness and alignment, there are still some open questions. Since we only used one specific robustification method, further experiments should determine the influence of this method. One could explore, whether a different choice of norm leads to different observations. However, there is a growing body of evidence (Kaur et al., 2019; Kim et al., 2019; Noack et al., 2019) which supports our finding that the alignment between images and their saliency map does not depend on the robustification method. Another future direction of research could be to investigate the degree of (non-)linearity and (non-)homogeneity and their connection to this topic. While Theorems 4.3.5 and 4.3.6 illustrate how the pointwise linearized robustness and alignment may diverge, depending on terms like g, $g^\dagger$, γ and $\beta^\dagger$, a more in-depth look should focus on *why* and *when* these terms have a certain relationship to each other. In order to further isolate the various, interplaying effects from one another, it could be of of interest to train purely positively 1-homogeneous networks by omitting bias vectors from the neural architecture. While their expressiveness is limited (e.g. one always has $\Psi(0) = 0$), for simple datasets such as MNIST their performance may still be satisfactory. By thus eliminating $\beta^\dagger$, the linearized robustness now coincides with the binarized alignment.

From a methodological standpoint, the discovered connection may also serve as an inspiration for new adversarial defenses, where not only the robustness but also the alignment is taken into account. These can help answer the question, whether the inverse of the question considered in this work is also true: 'Does an increase in alignment also lead to an increase in robustness?'
For this, various penalty terms similar to

$$1 - \frac{\langle x, \nabla\Psi^i(x)\rangle^2}{\|x\|^2\|\nabla\Psi^i(x)\|^2 + \varepsilon}$$

(which is bounded from below by 0 via the Cauchy-Schwarz inequality) with $\varepsilon > 0$ and true label i were tried out in early experiments, but yielded unstable training procedures. Another, similar way of increasing the alignment directly would be by penalizing

$$\|x\|^2\|\nabla\Psi^i(x)\|^2 - \langle x, \nabla\Psi^i(x)\rangle^2.$$

Any robustifying effects of the induced increase in alignment may however be confounded with the local Lipschitz-penalty that the first summand effectively introduces, which necessitates a careful experimental evaluation.

Chapter 5

Robust Tumor Typing with Deep Relevance Regularization

This chapter is concerned with the classification of MALDI mass spectra for tumor typing. It is based on the following two articles:

> Jens Behrmann, Christian Etmann, Tobias Boskamp, Rita Casadonte, Jörg Kriegsmann, and Peter Maass. Deep Learning for Tumor Classification in Imaging Mass Spectrometry. In *Bioinformatics*, Volume 34, Issue 7, 01 April 2018, Pages 1215–1223, 97:1823-1832, 2017.
>
> Christian Etmann, Maximilian Schmidt, Jens Behrmann, Tobias Boskamp, Lena Hauberg-Lotte, Annette Peter, Rita Casadonte, Jörg Kriegsmann, and Peter Maass. Deep Relevance Regularization: Interpretable and Robust Tumor Typing of Imaging Mass Spectrometry Data. Manuscript submitted for publication. Preprint available at arXiv:1912.05459, 2019b

The first article establishes neural networks as effective tools for tumor classification on two datasets (for lung tumor subtyping and lung/pancreas tumor typing). This is achieved by designing a CNN architecture which is specifically adapted to the properties of mass spectra. Furthermore, a scaled saliency map is used to retrace the classifier's classification decision in order to check the biological plausibility of the model.
In the second article, this work is extended by considering a regularization scheme using double backpropagation. This is able to overcome low accuracies on a multi-center dataset for ovarian and breast cancer by incorporating prior knowledge about the data structure.

Since the second article is in parts an update to the first, the methodology differs to a certain degree between the two studies. We will give justifications for these decisions when appropriate.

5.1 Data Analysis for Mass Spectrometry

In recent years, there has been a growing interest in Imaging Mass spectrometry (IMS) for spatially resolved molecular analysis of small to large molecules. In general, mass spectrometry uses the ionization of chemical compounds in order to separate the resulting ions according to their mass-to-charge ratio (m/z), measured in Dalton (Da). While there are various types of ionization methods, the particular method employed here is MALDI (*Matrix Assisted Laser Desorption/Ionization*), which ionizes the chemical compounds by help of a laser pulse. In order to facilitate ionization, a crystalline *matrix* is applied to the samples, which absorbs the photons emitted by the laser and passes this energy on to the compound (Zenobi and Knochenmuss, 1998). Since the ionization is performed with a laser, it is possible to record mass spectra in a spatially resolved manner, which gives rise to the technique of MALDI-IMS for biochemical imaging applications (Caprioli et al., 1997).
Given a thin tissue section (such as slices of tumor tissue collected during surgery), 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) abundances and thus offer insights into the chemical composition of a region within the tissue, see e.g. (Stoeckli et al., 2001). In particular, this technique is applicable to formalin-fixed paraffin-embedded (FFPE) tissue (a common tissue storage solution in pathology), on which tryptic digestion can be performed, resulting in characteristic peptide patters for proteins of interest. Hence, MALDI-IMS has a high potential for many applications in pathology, as discussed by (Aichler and Walch, 2015) or (Kriegsmann et al., 2015). One of the main advantages of MALDI-IMS is that it allows for high-throughput analysis of several tumor cores from different patients by arranging them in a single tissue microarray (TMA) (Casadonte et al., 2017). Thus, within a single run of the mass spectrometer, a large cohort of cancerous tissue can be analyzed in order to extract biochemical information in a spatial manner. This biochemical information may then be used for the determination of the cancer subtypes or the identification of the origin of the primary tumor in patients with metastatic disease, where 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 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). In particular, the classification of mass spectra allows for tumor (sub-)typing if suitable training data is available. Such training data can be annotated by a pathologist according to the tissue's morphological features, immunohistochemical stainings, the patient history and other available information, which may not even be available at test time.

One complication lies in technical variability of the mass spectra. These may stem from just small differences in the measurements. As each measurement involves several experimental steps (including tissue preparation), each aspect can contribute to unexpected effects within the data and add up (Cordero Hernandez et al., 2019; Buck et al., 2018). These differences can act as confounding factors during training and thus cause a

classification pipeline to classify based not only on biologically plausible features, but also on data artifacts. Any classification pipeline thus needs to exhibit robustness to these kinds of confounding factors.

This is especially relevant in many real-world scenarios, where the training data was obtained under different conditions than the data the model is applied to. In particular, this includes clinical applications for tumor typing, where the training data would typically not be created at the same time or even the same place as the data derived from the patient's tumor tissue.

Classically, when constructing a classification pipeline for IMS data, one has to decide on a combination of a multitude of methods, each of which may end up having a large impact on the result:
First, the data needs to be preprocessed. This may include different types of normalization (e.g. TIC normalization, which amounts to ℓ_1-normalization) (Deininger et al., 2011), smoothing or denoising of the spectra. Then, methods for feature extraction or feature selection need to be applied, which ensures that only the most descriptive or discriminative aspects of the data are passed on to the classifier. Often, this is accompanied by a dimensionality reduction. This is not only beneficial for numerical reasons, but may also alleviate the *curse of dimensionality*, a phenomenon that impairs the performance of classifiers on data of high dimensionality (cf. Friedman et al., 2001). Methods for feature extraction and/or selection include supervised peak picking, non-negative matrix factorization (Lee and Seung, 1999; Boskamp et al., 2017; Leuschner et al., 2018), principal component analysis or autoencoders (Thomas et al., 2016). These features are then passed to an appropriate classifier, such as linear discriminant analysis classifiers (LDA), support vector machines (SVM), neural networks (NN) or random forests (Galli et al., 2016; Boskamp et al., 2017).

Since the number of possible combinations of these steps is very large, constructing a good classification pipeline is a very challenging task. This is impeded by the fact that most methods require some form of hyperparameter tuning.
Another important aspect lies in the interpretability of the obtained classification pipeline. In order to be accepted by doctors and patients alike, a biologically plausible and interpretable model is desirable, which allows for further validation besides classification accuracies. If the mass spectra are e.g. classified with a simple linear classifier based on a subselection of peaks (e.g. based on the area under the receiver operating characteristic curve in each m/z-position, the *ROC-value*), the classifier's weights provide information which peaks the classifier takes into account the most. This can then be checked against a priori known cancer biomarkers. If on the other hand the feature extraction is performed with a kernel PCA (Schölkopf et al., 1998) and the resulting features are then classified using some classification model, a verification of the biological plausibility of this pipeline is much more difficult, since the features do not allow for easy interpretation.

In this chapter, we will thus not only focus on the classification accuracy, but also keep this aspect of 'interpretability' in mind.

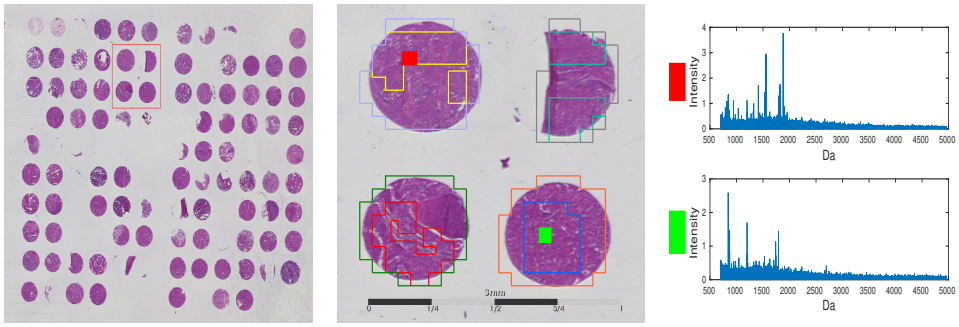


Figure 5.1: Overview on structural hierarchies of the IMS data, from TMA to tissue core to a single spectrum. Left, a stained image of a TMA is shown, which is measured in a single IMS measurement. The red box (left) marks the four tissue cores shown in the middle. These tissue cores have two annoations, 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.

5.1.1 Deep Learning Concepts for IMS Data

The vast success of deep learning in general and CNNs in particular naturally opens up the question, whether these concepts gives advantages for IMS classification as well. Deep learning has been introduced to IMS data prior to this work, but with a focus on unsupervised dimensionality reduction methods, see (Thomas et al., 2016), where autoencoders were used to reduce the dimensionality of rat brain IMS data. Moreover, (Inglese et al., 2017) introduced a neural network based dimensionality reduction to find metabolic regions within tumors. However, we focus on a fully-supervised deep learning approach which was novel for large-scale tumor classification with IMS data at the time of publication of (Behrmann et al., 2017).

5.2 Designing an Architecture for IMS

A main driving force in the design of deep CNNs for images is the need to process high-dimensional data, which is why the idea of convolutional transforms with their few parameters plays a key role. Moreover, the layered architecture is motivated by extracting features from different levels of abstraction. 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 such as the faces of humans (LeCun et al., 2015). However, the application of these concepts to spectra from IMS data poses the question of how these operations may act in this different domain. Mass spectra can also be viewed as high-dimensional data on a grid, the *m/z-bins*, where the dimensionality typically ranges from 10^5 to 10^6 for TOF data. Hence, by grouping neighboring m/z-bins together through convolutions, CNNs can offer the same working principle in this high-dimensional domain as for images. As the spectra are transformed throughout the network, the size of grouped neighboring m/z-bins grows, which is also called the *receptive field size*. By subsampling through strided convolution or max pooling, this receptive field grows roughly by the downsampling factors. For a comprehensive overview over receptive field calculations, see (Araujo et al., 2019).

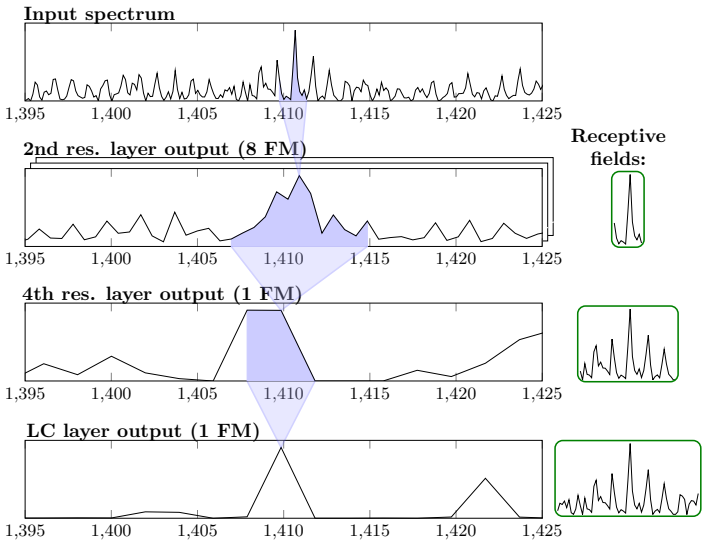


Figure 5.2: Overview over the working principle of *IsotopeNet*. The first row shows a section of a recorded mass spectrum of a squamous cell carcinoma. Several residual layers of depth 2 extract features of interest from small portions of the previous layers' outputs. Due to their consecutive (and partially 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).

While the applicability to IMS data seems reasonable, it is important to discuss the underlying assumptions. The main assumption of a convolutional transform is that neighboring m/z-bins are correlated. 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, where 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. On the highest level, tryptic-digested proteins may contribute to several measured peptides, resulting in patterns across the entire mass range. This means that mass spectra (in the application of tumor typing) have features that are inherently *local*, and others that are inherently *global*. Local features are e.g. peaks spanning several m/z-bins as well as isotopic patterns, which are comprised of several peaks. Global features on the other hand are the specific positions of these local features on the m/z-axis, which indicate whether a peak or a peptide pattern belongs to a certain biomarker.

This observation inspired an adapted architecture for mass spectrometry data, which we call *IsotopeNet*. The general idea of IsotopeNet is to use local feature extractors – convolutional layers – only until their receptive field is as large as the largest local features we expect, which are isotope patterns. These features are then related to one another by a dense softmax layer.

Table 5.1: Architecture of IsotopeNet. The residual layers' depth refers to the number of convolutional layers in the residual branch. For the locally-connected layer, the kernel size refers to the width of its weight matrix' band.

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
Dense layer (softmax)	-	-	-	1

We estimate the number of m/z-bins of large measurable isotope patterns of peptides based on the 'averagine' amino acid (Senko et al., 1995), see Appendix B.1. Note that this is dependent on the m/z-resolution. For the three datasets used in this study, this was roughly the same, which is why one fixed architecture for all datasets was used. Using the averagine estimates, we restrict the receptive fields roughly to the size of large isotope patterns, such that the local feature extraction is able to encode this feature. However, it should be noted that the receptive fields are partially overlapping, such that each variable may additionally encode parts of neighboring patterns. Untangling this information is facilitated by employing several feature maps. The desired downsampling is realized via strided convolutions. For this, the first downsampling factor (appearing as a stride of 5 in the second residual layer) is determined by the average distance between peaks. The next downsampling factor is chosen, such that our requirement for the receptive field size is met. We further utilize batch normalization (Ioffe and Szegedy, 2015) throughout the network.

While the convolutional transforms apply a uniform transformation over the feature space, a subsequent locally-connected layer is able to treat each region of the feature space differently due to its 'unshared weights'. It can be seen as a preprocessing step before the global feature extractor, the final dense layer (with softmax activation function).

The architecture of IsotopeNet is summarized in Table 5.1. Its working principle is illustrated on a spectrum of a squamous cell carcinoma in Figure 5.2.

5.3 Experiments

5.3.1 Datasets

In this study, we test the proposed methods on two IMS datasets, both comprised of several TMAs of a cohort of tissue cores. The first classification task (8 TMAs) is to distinguish two lung tumor subtypes, namely adenocarcinoma from squamous cell carcinoma (*task ADSQ*), while the second task (12 TMAs) is to discriminate lung tumors (from those 8 TMAs) and pancreas tumors (4 additional TMAs), which we will call *task LP*. The latter task of classifying tumors from different organs is a first step towards

reliably classifying tumors collected from the same organ, but having different primary sites (i.e. metastases). These datasets have been used in two prior works by (Kriegsmann et al., 2016) and (Boskamp et al., 2017), but are used in this study to verify the potential of deep learning methods for tumor classification in IMS.

Note that there are several tissue cores collected per patient (1 or 2 in lung TMAs, 3 on average in pancreas TMAs). Furthermore, in the lung dataset there are also annotated subregions called Regions-of-Interest (ROI), see Figure 5.1. These regions were marked by a pathologist as relevant subregions within the tissue core for subtyping the tumor. In order to perform classification only on those subregions, only those spectra within each ROI are used for task ADSQ, resulting in a reduced number of spectra of 4672. On the other hand, for task LP the entire tissue core was used which also include spots with non-tumor cells, resulting in a total of 27475 spectra. For evaluating the performance of our methods, we used randomized 4-fold cross-validation (CV) on TMA level.

As the performance measure for this binary classification problem, we used the *(class) balanced accuracy*

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

where TP/(TP+FN) denotes the true positive rate and TN/(TN+FP) denotes the true negative rate[1]. This measure is not biased by the proportions of class abundance in the data, unlike the accuracy (i.e. the fraction of correctly classified samples).

5.3.2 Comparison Baselines

In our experiments, we compare IsotopeNet both to a proven classical approach and a standard neural network architecture. We report the median balanced accuracy over the four cross-validation runs.

As our classical baseline method, we use supervised peak-picking method, whose selected peaks are classified with a linear discriminant analysis classifier (LDA), which is the same baseline method as in (Boskamp et al., 2017; Leuschner et al., 2018). For the peak-picking, we select the K m/z-values with the highest ROC-value on the respective training set. This is equivalent to computing the Mann-Whitney-Wilcoxon statistic for each m/z-value separately. For K, we chose values between $K = 5$ and $K = 100$. The combined method is referred to as *ROC/LDA* in the following

In order to test IsotopeNet against more standard approaches from image processing, we test it against a deep residual networkork (He et al., 2016), the standard design paradigm for image processing. Their residual connections are an important ingredient to making networks increasingly deep, which is regarded as a key factor for training highly accurate image classification models. Due to this depth, the deeper convolutional layers compute features whose receptive fields span the whole input image (Araujo et al. (cf. 2019) for a list of receptive fields of common architectures). The number of feature maps is chosen to be comparatively large in order to encourage the extraction of diverse features. The final, chosen architecture for the residual network is given in Table B.1 in the Appendix.

[1]For the binary classification problems considered, one can define either one of the two classes to be 'positive' respectively 'negative' and get the same results. For multi-class problems, one should instead consider the average sensitivity per class, which reduces to the class-balanced accuracy for the binary classification task.

Table 5.2: Comparison table of 4-fold cross-validation on both ADSQ and LP. 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 obtain by taking the majority of the predicted label.

Method	Task ADSQ		Task LP	
	Bal. Accur. (Spot)	Bal. Accur. (Core)	Bal. Accur. (Spot)	Bal. Accur. (Core)
ROC/LDA	0.758	0.788	0.794	0.876
	0.787	0.860	0.840	0.918
Residual Network	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

The concept of IsotopeNet is in some sense diametrically opposed to this: Only few residual layers (with convolutions) are used in order to extract local features. These are then classified using a locally-connected layer, which – in contrast to convolutional layers – can treat the individual regions of their input features differently. A subsequent dense layer then allows for the classification according to the features' location within the mass spectrum.

Prior to feature extraction, all spectra were normalized by the total ion count (TIC) (Deininger et al., 2011). Figure 5.3 (left) reports the results on both tasks ADSQ and LP.. For this baseline method 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 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 5.3 (left) further shows how the residual network compares to the domain adapted architecture *IsotopeNet*. Due to the stochasticity of the training process through stochastic gradient descent, random initialization and regularization by dropout, 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 estimation of the average performance. Furthermore, the interquartile range is stated below the median to estimate the variance induced by the above-mentioned stochasticity. Overall, the domain adapted architecture *IsotopeNet* performs better than both *ResidualNet* and *ROC/LDA*, for example with a spot level balanced accuracy of 84.5% for task ADSQ. The reported tests were conducted on a GeForce GTX TITAN X GPU (Nvidia) and computations were compiled to CUDA via the Python framework `Theano 0.8` (Al-Rfou et al., 2016).

Whereas the previous discussion considered the variance of several runs over the entire dataset, Figure 5.3 (right) visualizes the variance over the cross-validation folds on spot level. For this box plot, the balanced accuracy of the four identical runs was computed 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. Furthermore, the variance is lower but still rather large and

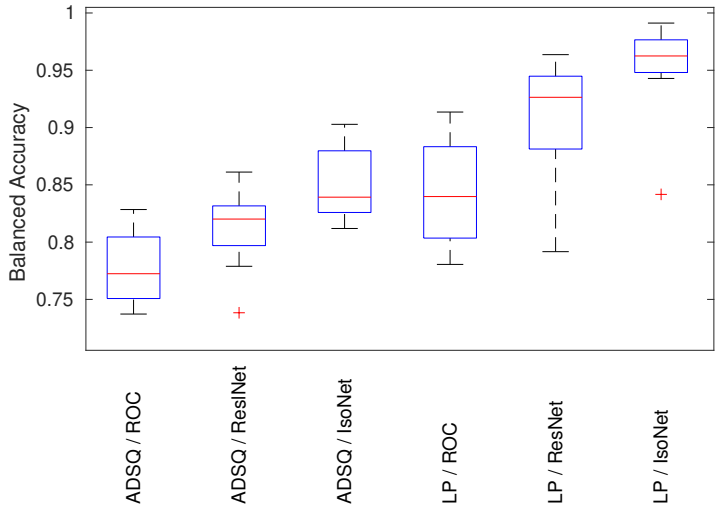


Figure 5.3: Boxplot of the balanced accuracies from each method over the four cross-validation folds, reported on spot level for tasks ADSQ and LP. Neural networks outperform the linear baseline for both datasets. Furthermore, the domain-adapted IsotopeNet architecture reacher higher balanced accuracies than the more standard approach of the residual network.

Table 5.3: Comparison table of both architectures showing the number of trainable parameter and the runtime per epoch.

Method	Runtime (ADSQ) per epoch	Runtime (LP) per epoch	Number of Parameters
ResidualNet	44.55 s	109.44 s	2,132,130
IsotopeNet	14.16 s	35.34 s	13,935

outliers (red +) occur for both methods. Hence, the impact of the choice of the splitting between training and test may have an influence, which is why a conclusion based on small performance differences may be too early. Moreover, Figure 5.3 shows that task LP seems to be easier for all methods. This is expected, as the task to differentiate primary tumor is most likely easier and more spectra were available which is especially crucial for deep learning.
Additionally to the test set performance discussed previously, the training set performance can be used to judge overfitting of methods. For example, *IsotopeNet* consistently reached a training balanced accuracy of about 95% on task ADSQ, whereas *ResidualNet* had a balanced accuracy of more than 98%. This effect may be explained by the number of parameters shown in Table 5.3, as *ResidualNet* is by far the larger model. Furthermore, the runtime per epoch is reported in this table, which further underlines the effectiveness of *IsotopeNet*.

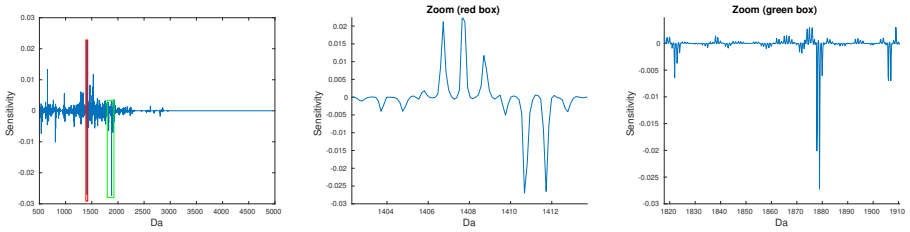


Figure 5.4: Left: Sensivitiy of *IsotopeNet* on task ADSQ, where the sensitivity was computed for the predicted probability of class AD. The red and green boxes mark the zoomed in region of the figures shown on the right. Middle: Zoom in with high peaks at 1406.6 Da and 1410.7 Da. Right: Zoom in with high peaks at 1821.8 Da, 1877.8 Da and 1905.9 Da.

5.3.3 Interpretation via Sensitivity Analysis

A competitive performance is only the first step towards the acceptance of an automated model for tumor typing. Interpretation from a biological point of view is crucial to uncover the strengths and weaknesses of a model. The common approach is to look for discriminative m/z-values of the feature extraction process, which was done by examining the ROC values of the m/z-values over the training set. However, finding the most significant m/z-values for deep learning models is more involved.

In order to perform a similar analysis for neural networks, we study their input-output-sensitivity in order to find out, which aspects of the input spectra contribute most to the classification. As with image data, this can be done by calculating the gradient of an output neuron with respect to the neural network's input. This can be expressed via $\nabla_x(f_\Theta(x)_j)$, the gradient of the class j, the *saliency map*. Compared to image classification models, where saliency maps are most often considered, the classification of biochemical mass spectra however has adverse properties, which result in unwanted behavior of visualizations with the pure gradient.
The partial derivatives $\frac{\partial(f_\Theta(x)_j)}{\partial x_i}$ (which comprise the entries of the vector $\nabla_x(f_\Theta(x)_j)$) can be interpreted as the change in $f_\Theta(x)_j$ for every unit of change in x_i (under linearization). The various considered ions however can vastly differ in abundance, meaning that a 'unit-strength' variation may be relatively large for some m/z-bins and comparatively small for others. To be precise, for a peak that usually exhibits low variation, the numerical value of the partial derivative will thus typically overestimate the peak's contribution to the classification. In the case of a high-variation peak on the other hand, the contribution hence tends to be underestimated.
In this study, we thus multiply each $\frac{\partial(f_\Theta(x)_j)}{\partial x_i}$ by a factor, which represents the typical variation in x_i. For this, we simply use the the standard deviation of x_i, denoted σ_i, as a crude measure for its typical variation. For $\sigma := (\sigma_1, \ldots, \sigma_n)^T$, we call

$$\text{sens}_j(x) := \sigma \odot \nabla_x(f_\Theta(x)_j) \tag{5.1}$$

the *sensitivity* of f_Θ with respect to class j.

The goal of this section is to analyze the proposed *IsotopeNet* for both tasks. For this, we select the best network out of four consecutive runs with the same setting. Then, the model from the best cross-validation-fold is taken into consideration. After choosing the model, the class under examination is chosen (AD for task ADSQ, Lung for task

LP). Finally, the sensitivity $\text{sens}_j(x^{(i)})$ from equation (5.1), where j denotes the chosen class, is computed for all spectra in the training data ($i = 1, ..., N$) and the mean over the samples i is taken.
In Figure 5.4 (left), the sensitivity for task ADSQ from the best performing *IsotopeNet* is shown. This sensitivity has the same dimension as a raw spectrum, which makes interpretation in the input domain feasible. In contrast to spectra, negative values occur in the sensitivity as well. The sign of a value indicates the slope direction, which means that positive values indicate a positive slope in the direction of higher probabilities for AD. On the other hand, negative values indicate a negative slope for AD, which in turn means that an increase of intensities with negative slope will result in higher probabilities for the other class, SQ. Hence, both the sign and the height of each peak in the sensitivity map in Figure 5.4 are important.

Most striking in this sensitivity is the concentration of sensitive peaks in the interval between 1000 and 2000 Da, which is to be expected as most peptides are measured in this range. Furthermore, the red and green boxes mark the zoomed in region shown in Figure 5.4 (middle) and (right), respectively. As the zoom of the red box shows, high peaks are found at 1406.6 Da and 1410.7 Da, together with a small isotope pattern. The positive peak 1406.6 Da acts as a marker for AD, while 1410.7 Da has a negative value and thus marks SQ. Most importantly, both peaks have been identified in (Kriegsmann et al., 2016) as a peptide of cytokeratin-7 (CK7, 1406.6 Da) and cytokeratin-5 (CK5, 1410.7 Da). Additionally, the zoom of the green box on the right shows a pattern at 1821.8 Da, a more expressed pattern at 1877.8 Da and a less expressed pattern at 1905.9 Da. Again, these were identified in (Kriegsmann et al., 2016) as peptides of cytokeratin-15 (CK15, 1821.8 Da and at 1877.8 Da) and heat shock protein beta-1 (HSP27, 1905.9 Da). Out of these four markers, CK5 and CK7 are already well-known IHC markers, whereas CK15 and HSP27 are two new potential markers (Kriegsmann et al., 2016). Hence, through analyzing the input-output relationship of the deep CNN through our sensitivity analysis, it stands to reason that the model for task ADSQ bases its classification on actual biomarkers.

However, the sensitivity of the best model for task LP in Figure 5.5 appears different at first glance. Compared to the sensitivity for ADSQ, the mass range below 1000 Da and high mass range over 2000 Da exhibits a higher sensitivity. We again observe zoomed-in regions of the average sensitivity in Figure 5.5. The figure in the middle shows peaks at 836.5 Da, 852.4 Da and 868.5 Da, which were previously observed as discriminative m/z-vales in (Boskamp et al., 2017). However, the local structure does not exhibit the isotope patterns that were visible for task ADSQ. Moreover, the zoom at the m/z-range from 2100 Da to 2900 Da (Figure 5.5 (right)) shows high oscillations almost for the entire interval. Hence, both zooms uncover a substantially different behavior compared to the sensitivities of ADSQ, which might indicate flaws in the model. An explanation for these effects may be artifacts induced by the measurement. These may be caused by the fact that the different classes were each collected in different measurement cohorts. These possible measurement artifacts should thus persist between the training and test data and make the classification task unrealistically easy. To draw a comparison to image data, this would be as if one class in an image classification task exhibited an unintentional red tint, which would make the classification trivially easy *without* ever learning the true data characteristics.

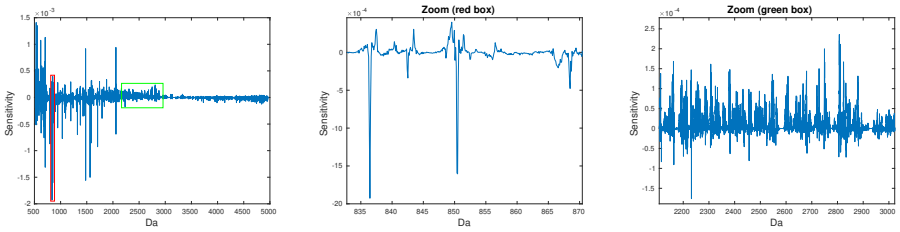


Figure 5.5: Left: Sensivitiy of *IsotopeNet* on task LP, where the sensitivity was computed for the predicted probability of class 'lung'. The red and green boxes mark the zooming region of the figures shown on the right. Middle: Zoom in with high peaks at 836.5 Da, 852.4 Da and 868.5 Da. Right: Zoom in showing high oscillations in the range of 2100 Da to 2900.

All in all, the sensitivity analysis has proven to be a useful tool for judging trained classification models for tumor (sub-)typing beyond (balanced) accuracies. In order to find out whether

5.4 Deep Relevance Regularization

The sensitivity analysis uncovered that on task LP, the classification was apparently performed not entirely based on biologically relevant peaks, but based on confounding factors. However, since these confounding factors correlated with the class labels, these confounding factors may have actually *helped* with the classification, as evidenced by the near-perfect test performance. This correlation is due to the fact that the samples from both classes were each collected in different measurement cohorts. This is however a very unrealistic setting for clinical applications, where the tumor type is a priori unknown. We thus test the viability of neural networks in a more realistic setting, which mirrors clinical applications.

5.4.1 Multi-Laboratory Study

Tissue samples from ovarian and breast tumors were gathered and assembled to 4 TMAs. Two of these TMAs consisted solely of breast tissue, while the other two consisted solely of ovarian tissue. The same TMAs were repeatedly measured in two different laboratories, once in a lab in Bremen (HB) and once in a lab in Trier (TR). The exact data gathering procedure is describe in much more detail in Appendix B.2.2.

We employ an *inter-lab cross-validation procedure*, where we divide the patients randomly into 5 roughly equally-sized groups. For every cross-validation configuration, we then train on 4 folds of one lab and afterwards classify the *other lab's* spectra of the patients *not used for training*. In order to simulate realistic, real-world clinical applications, we will never report test scores on the same patients or the same lab as the data the model was trained on. This procedure is performed for both labs. The cross-validation procedure is visualized in Figure 5.7(a). This 5-fold CV procedure for 2 labs results in a total of 10 neural networks trained for the unregularized baseline model. As the predicted labels form disjoint sets, we can calculate the balanced accuracy for their disjoint union. This procedure much more closely resembles the clinical use case, in which the data is typically not recorded under the exact same conditions as the training data.

Since our target is a high class-balanced accuracy, one should take the class proportions into account in the training procedure, such that members of rare classes are assigned more weight than members of a frequent class. For a spectrum-label pair (x, y), we weigh its loss $\ell(f_\Theta(x), y)$ in the ERM-term inversely proportionally to the relative abundance of class y. We normalize this by the number of classes, such that the original ERM is recovered for balanced classes. This resulting class-weighted ERM-term is thus written as

$$\frac{1}{N}\sum_{i=1}^{N} w_y \cdot \ell(f_\Theta(x^{(i)}), y^{(i)}),$$

with $w_y = \frac{1}{C} \cdot \frac{N}{N_y}$, where N_y denotes the number of samples of class y in the respective training set. This is a common approach, for example utilized as the standard option for class weighting in `scikit-learn`, a popular machine learning library in Python (Pedregosa et al., 2011). The calculations were performed in our MATLAB-library `MSClassifyLib`, which serves as an API to `Tensorflow 1.12`, using an NVIDIA GeForce GTX 1080Ti GPU.

The above-described procedure resulted in a (spot-wise) balanced accuracy of just 37.3%. When aggregating over the patients (i.e. when assigning a single label to each patient according to a majority vote over the their spots), the balanced accuracy drops even further to 34.9%. As these results are worse than guessing on average, they suggest that the neural networks learn confounding factors instead of biologically relevant properties.

We further tried out weight decay as a regularization strategy with different parameter values, which was not successful either. This underlines that this is not a problem of overfitting (induced by using too little training data), but a true distribution shift between the training and test data.

5.4.2 Interpretation of Classification Results via Layerwise Relevance Propagation

Similarly to the sensitivity analysis performed in section 5.3.3, we check whether our assumption that the misclassification is caused by spurious correlations is true.

While the sensitivity did provide additional information about our models beyond the classification score, it has some aspects which could be improved upon. A core concept of the sensitivity analysis is to attempt to overcome the problem of very uneven variations between the different peaks by scaling via the standard deviation over each m/z-position. This scaling is *uniform* over all samples. As such, it cannot adapt to each sample. Furthermore, taking the gradient of the output probabilities has the disadvantage that output probabilities close to 0 or 1 result in sensitivities close to 0, an effect that is more closely investigated later this section. When taking averages over a set of sensitivites, this hence reduces the impact of well-classified samples.

For assessing the relevant portions of a neural network's input, a multitude of *attribution methods* for neural networks have been developed in the literature, which allow for a *post hoc* interpretation by providing some estimate of feature importance for the classification. The resulting *relevance map* $r_\Theta(x, y) \in \mathbb{R}^n$ should exhibit large positive values at the

parts of the input x that act as strong evidence for class y, whereas large negative values should signify strong counterevidence against class y. Values close to 0 should play little to no role in the classification.

Attribution methods roughly fall into one of two categories: perturbation-based attribution methods and backpropagation-based attribution methods. Perturbation-based methods such as Occlusion (Zeiler and Fergus, 2014) and Prediction Difference Analysis (Zintgraf et al., 2017) observe the influence on the classification score after perturbing different parts of the input. Backpropagation-based methods typically calculate their relevance attribution via modifications of gradient backpropagation in the neural network. Among these are the *saliency maps* (Simonyan et al., 2013), which were the focus of chapter 4 and the basis for the sensitivity analysis. Other backpropagation-based attribution methods are the misnomered *deconvolution* (Zeiler and Fergus, 2014), guided backpropagation (Springenberg et al., 2014), layer-wise relevance propagation (Bach et al., 2015) and its generalization deep Taylor decomposition (Montavon et al., 2017a,b), DeepLIFT (Shrikumar et al., 2016), forward-backward interpretability (Balu et al., 2017), VisualBackProp (Bojarski et al., 2016), Excitation Backprop (Zhang et al., 2016), GradCAM (Selvaraju et al., 2016) and PatternNet/PatternAttribution (Kindermans et al., 2017). There are also 'meta' attribution methods, which calculate averages or line integrals of the above-mentioned attribution methods over regions of the input space, such as SmoothGrad (Smilkov et al., 2017) (which were examined more closely in Section 4.5), and integrated gradients (Sundararajan et al., 2017). For image data, these are known to result in better-looking relevance maps, but require many evaluations.

Here, we choose a different approach that is theoretically well-founded and which ends up being conceptionally very similar, the above-mentioned *layerwise relevance propagation* (LRP). While LRP has a few varieties, we choose a variant (*z-rule*) that allows for easy implementation in order to encourage adoption by researchers and practitioners. Propagating from the value of an output neuron back to the input, LRP uses a conservation law in order to assign a neuron's relevance according to its additive contribution to its output. As (Ancona et al., 2017; Kindermans et al., 2016; Shrikumar et al., 2016) show, for neural networks with exlusively ReLU nonlinearities, the LRP of the output logit $\Psi_\Theta(x)_y$ reduces to the very simple form

$$r_\Theta(x, y) = x \odot \nabla_x(\Psi_\Theta(x)_y) \tag{5.2}$$

(where $\odot$ denotes the entry-wise multiplication), which is the case e.g. for IsotopeNet and many other neural network architectures. This is in fact very similar to the sensitivity analysis formula (5.1). The multiplication of the gradient with the standard deviations is here replaced by a multiplication with the input. This in particular means that the scaling is now adaptive to the input instead of uniform.

From a application perspective, this multiplication by the input should also fulfill our original requirement of scaling the partial derivatives to reflect the variation per m/z-value: It is reasonable to assume that more abundant peaks exhibit a larger (absolute, not relative) variation than less abundant peaks. When assuming that the variation is proportional to the abundance x_i, this leads to a formula that is proportional to the LRP-term (5.2).

We now apply LRP to the model trained on the multi-lab dataset to find out, whether the very low balanced accuracy can indeed be attributed to measurement differences

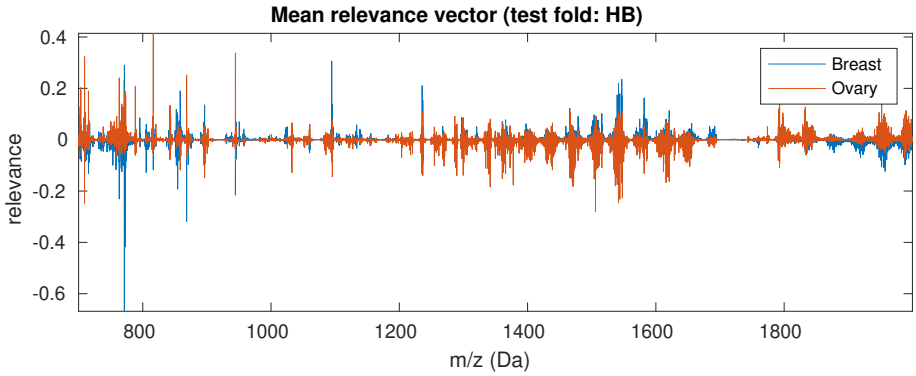


Figure 5.6: Mean relevance maps per class of one of the five test folds of the HB lab. Large intervals of seemingly relevant peaks instead of isolated peaks indicate that confounding factors (e.g. baselines) are being learned.

between the two laboratories. In Figure 5.6, the average relevance map with respect to the respective correct class of one of the test sets (on HB) is visualized. While a few singular peaks appear, the neural network mostly seems to regard larger intervals as relevant. This points towards confounding factors (e.g. differing baselines) being learned instead of actual biomarkers.
We emphasize that we decided to visualize the relevances with respect to the logit of the *correct class*. While the logit of the other class may be larger (resulting in a misclassification), both relevance maps may be inspected independently.

5.4.3 Designing a Regularization Term

For IMS data used in tumor classification tasks, it can often be assumed that only a fraction of each input spectrum is actually relevant in the classification. For example for task ADSQ, 4 potentional biomarkers were identified. Due to artifacts induced during the tissue preparation or the spectral measurement (such as delocalization or ion suppression effects (Cole, 2011)), some region of the mass spectra may, however, correlate with a certain class, despite not actually being a biomarker. This may e.g. occur, if the test data was recorded with a different machine or by a different operator than training data, which is a scenario simulated by this multi-laboratory study.

Ideally, we would like to restrict our model to only take into account the most *relevant* m/z-values, while ignoring the above mentioned data artifacts. One such restricted model can be regarded as simpler and may thus be expected to be more robust to overfitting.

In other words, for the classifier to only take into account the most important parts of x, the vector $r_\Theta(x, y)$ should generally be *sparse*, i.e. have most entries close to zero. To this end, we modify our training objective to be

$$\min_\Theta \frac{1}{N} \sum_{i=1}^{N} \left[\ell(f_\Theta(x^{(i)}), y^{(i)}) + p(\lambda, r_\Theta(x^{(i)}, y^{(i)})) \right], \tag{5.3}$$

where the *regularization term* $p(\lambda, r_\Theta(x, y))$ assumes low values for sparse relevance maps and high values for non-sparse (*dense*) relevance maps. Here, λ controls the regulariza-

tion strength. The formulation in (5.3) ensures that the resulting classifier f_Θ strikes a balance between accuracy and relevance sparsity. We call this proposed approach *Deep Relevance Regularization* (DRR). For sufficiently overparameterized model families (like most NNs), DRR can be regarded as a model selection mechanism, in which case one might hope to find a model which does not even reduce the accuracy at all compared to the unregularized network. This approach is conceptionally similar to (Ross et al., 2017) and (Rieger et al., 2019). Generally, a sparser relevance map can be expected to be easier to interpret, because fewer m/z-values would need to be tested for their biological relevance.

A natural measure for sparsity is the '0-norm'[2] $\|r_\Theta(x,y)\|_0$, which counts the non-zero entries of $r_\Theta(x,y)$. The 0-norm is unsuited for derivative-based optimization purposes, however, because it has derivative 0 almost everywhere and exhibits jump discontinuities. Furthermore, it may be too restrictive for our purposes, since a very small entry of $r_\Theta(x,y)$ is given the same weight as a very large entry. We therefore use the 1-norm as a proxy measure for sparsity. The sparsity-promoting property of the 1-norm is well known in areas such as inverse problems (Jin and Maass, 2012), compressed sensing (Donoho et al., 2006) and machine learning. One example is the LASSO for linear regression (Tibshirani, 1996), where the loss function

$$\frac{1}{N}\sum_{i=1}^{N} \|Wx^{(i)} - y^{(i)}\|_2^2 + \lambda\|W\|_1$$

is minimized[3] with respect to W and data $\mathcal{T} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$, which leads to sparse W. However, a well-known phenomenon that arises in LASSO models is that out of a group of highly correlated patterns, only one (or few) are selected by W. This can be explained by the model gaining additional sparsity through dropping this supposedly 'redundant' information. If this is not desired, the effect can be mitigated by adding an additional regularization term $\mu\|W\|_F^2$ to (5.3), where $\mu \geq 0$. The resulting model is then called the *elastic net* (Zou and Hastie, 2005). In trypsin-digested IMS data, highly correlated patterns are found on several levels: Each peak is several m/z-bins wide, isotopic patterns consist of several neighboring peaks and peptide patterns contain correlated patterns of different peptides. If one wishes to include this information for DRR, a similar regularization term should be employed. This also offers advantages for interpretability, since this information is retained. For these reasons, in the following we choose

$$p(\lambda, r_\Theta(x,y)) = \lambda_1\|r_\Theta(x,y)\|_1 + \lambda_2\|r_\Theta(x,y)\|_2^2 \tag{5.4}$$

to be the DRR term in the objective (5.3) for $\lambda_1, \lambda_2 \geq 0$.

Whether the 'surviving' m/z-values are actually biologically relevant or themselves confounding factors, is to be determined empirically. We however expect that e.g. differing baselines between the classes should result in large intervals of the spectrum being deemed 'relevant' for the classification, in contrast to isolated peaks, which can be expected to be the result of specific measured ions.

[2]The 0-norm is not actually a norm, because it lacks the property of absolute homogeneity.
[3]Here, $\|\cdot\|_1$ denotes the entrywise 1-norm, not the operator 1-norm.

Logit or Softmax Scores

It is generally advisable to use logit scores instead of softmax scores for the calculation of the LRP-term (5.2). This is because for softmax scores, the gradients (and thus the relevance maps with respect to these softmax scores) tend to 0 as class prediction probabilities tend to either 0 or 1. This is especially striking in the 2-class case. Let $(q_1, q_2)^T := f_\Theta(x)$ and $(z_1, z_2)^T := \Psi_\Theta(x)$, then

$$\begin{aligned}\nabla_x q_1 &= q_1(1-q_1)\left(\nabla_x z_1 - \nabla_x z_2\right)\\ \nabla_x p_2 &= \nabla_x(1-q_1) = -\nabla_x q_1\end{aligned}$$

meaning that for small or large probabilities, the gradients would tend to 0.

When using this as the relevance function for DRR, this has the adverse effect that the samples with probabilities close to 50% are penalized more strongly than those that the NN is 'surer' about. However, for the already well-classified samples *in the training set*, one can 'afford' a more restrictive model (i.e. with a higher imposed sparsity on the relevance), whereas the badly-classified examples require more flexibility.
Note that this logic is reversed for *test data*: For badly-classified examples in the test set, a more restrictive model (i.e. higher λ_1, λ_2) can be expected to yield a better classification for spectra that are difficult to classify. What are the 'right' values of λ_1 and λ_2 can however not be assessed purely by training on the training set, because performance on the training set tends to be highest for the most unrestricted models. One should therefore select these values based on a (cross- or hold-out) validation procedure.

DRR and Weight Regularization

One might also consider applying a sparsity penalty to the weights instead of the relevance map. For multilayer perceptrons (MLPs, i.e. NNs that consist only of dense layers), this is a sensible approach, as apprioriately-set zeroes in the first layer will eliminate the influence of certain entries of x. In most common situations, as in the case of e.g. IsotopeNet, this is however not adequate:

- For high-dimensional data (like IMS data), the number of weights of deep NNs may increase drastically. Apart from memory restrictions, this also increases the likelihood of overfitting. Furthermore, these do not make use of localized information, for which convolutional layers are more suited, as explained in (Behrmann et al., 2018).
- In convolutional layers, parameter sparsity implies neither sparse gradients nor activations: For example, a very large convolutional kernel with zeroes everywhere except for the entry 1 in the middle, while being highly sparse, realizes exactly the identity function[4].
- Residual connections (He et al., 2016) allow for the training of very deep NNs and are ubiquitous in classification architectures (such as IsotopeNet). Here, penalizing the norm of weights lead to a mapping that is close to the identity function.

For the last two points, however, certain activation functions like ReLU may affect the sparsity as well. Another big advantage of DRR over weight sparsity is that of higher flexibility: For MLPs, a zero in the first hidden layer's weight matrix excludes this m/z-position for every examined mass spectrum. DRR on the other hand allows the

[4]For appropriately zero-padded convolutions.

model to assign a low relevance to a certain position for some mass spectrum, whereas the model may assign a high relevance to it for a different spectrum. In other words, the neural network may individually 'decide' whether a certain peak is relevant for the classification of a mass spectrum.

5.5 Experiments with Deep Relevance Regularization

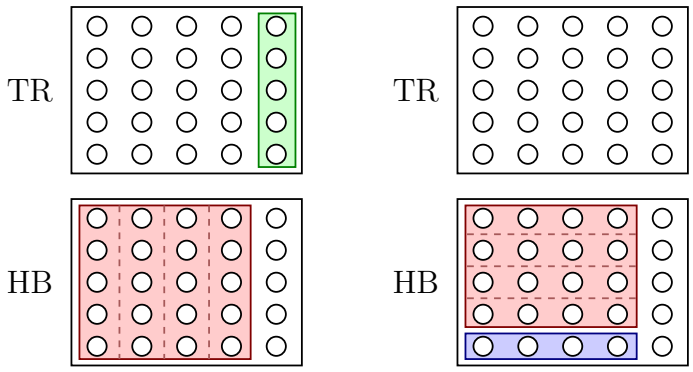
In the following, we aim to find out whether a deep relevance regularized network is able to overcome the low accuracies that were observed in the case of the unregularized IsotopeNet (Section 5.4.1).

The regularization strength for DRR is chosen based on a cross-validation procedure within the same lab. For a fair comparison with other established methods, we further train a ROC/LDA model, where the number K of peaks is determined according to the same hyperparameter search strategy as in the case of DRR.

5.5.1 Application of DRR

We now repeat the experiment using DRR, with otherwise equal settings. Our goal is to find out whether a neural network with DRR trained on data from one lab performs well on another lab's data. For this reason, a fair hyperparameter selection strategy should be employed, which *only* uses information from the training lab. We therefore employ a *nested inter-lab cross-validation strategy*, which is visualized in Figure 5.7. The outer CV corresponds to the same CV as for the unregularized network. Every training set of the outer CV is again randomly divided into 5 folds (based on patients) for the inner CV. There, different values for the regularization strength are tested on the respective remaining fold, this time *on the same laboratory* as the training data. This is done to check, whether the DRR model is able to perform well even when only taking into account the training data.

Due to the relatively high computational cost of training many neural networks, we chose the same value for λ_1 and λ_2 in (5.4) for faster training, which we will call λ. This results in a linear growth in the number of tried-out parameter values instead of a quadratic growth, which saves a considerable amount of computation time. In prior *intra-lab* experiments, this was found as a useful heuristic for TIC-normalized data. The optimal value for λ was searched for on the logarithmic grid $G = (10^{-5}, 10^{-4.5}, \ldots, 10^{-2})$, consisting of 7 values in total. In the prior experiments, this was deemed a sensible range, at the ends of which the performance started to drop again. After the inner CV is finished, we do not choose the value for λ with the respective highest validation score, but the next highest value in the grid G (if there is one). We chose this approach, because of the induced overoptimism from validating on the same laboratory as the training set. In other words: For the test sets on the outer CV (which stem from a different lab than the training data), we expect to require a more robust model (i.e. a higher value of λ) than for the validation sets. This approach mirrors strategies like the *one-standard error rule* (cf. Hastie et al., 2015), which is another popular heuristic for choosing more robust regularization parameters. For this cohort, a total of $2 \cdot 5 \cdot 5 \cdot 7 + 2 \cdot 5 = 360$ neural networks were thus trained, taking roughly 25 GPU-days. While this is a long time, the training of a single model just takes between 1.5 and 2 hours. By using hyperparameter choice heuristics such as the one proposed in Figure 5.8, the computation time can be



(a) Outer CV (b) Inner CV

Figure 5.7: Visualization of the inter-lab nested cross-validation procedure. For the outer CV, the models are trained on one lab (shown in red), but tested on the data from the remaining patients recorded in the other lab (shown in green). In order to choose a good hyperparamter for the DRR, an inner CV is employed, where the model is tested on the same lab (shown in blue). This is done for both labs.

greatly reduced.

The balanced accuracy for this cohort was 77.4%, or 80.6% aggregated on patient-level. This stark increase in accuracy suggests that instead of confounding factors, relevant information is being learned, which persists across measurements. We therefore inspect the mean relevance maps (per class) of the same fold as in the case of the unregularized network and visualize them in Figure 5.9. As desired, these are much sparser than those of the unregularized neural network (Figure 5.6).

5.5.2 Comparison to Linear Model

We employ the same inter-lab cross-validation procedure for hyperparameter tuning to a ROC/LDA comparison model. Like in the case of DRR, the number K of picked peaks is chosen based on the performance in the inner CV. Due to the relatively low computation time, we chose a grid of 16 numbers for K between 5 and 200. Analogously to the DRR-experiment, we chose the next-lowest value on this grid (if possible) compared to the value that gave the best performance on the (intra-lab) validation set.

The above setup leads to a spot-wise balanced accuracy of 75.5% (78.8% aggregated over patients). Only taking into account a small subset of m/z-values apparently tends to filter out some of the confounding factors, mirroring the DRR-network approach. Still, the performance of the neural network with DRR is superior to this linear classification model.

5.5.3 Inspection of Relevance Maps

In the following, we will take a closer look at the relevance maps induced by both deep relevance regularized networks and unregularized neural networks.

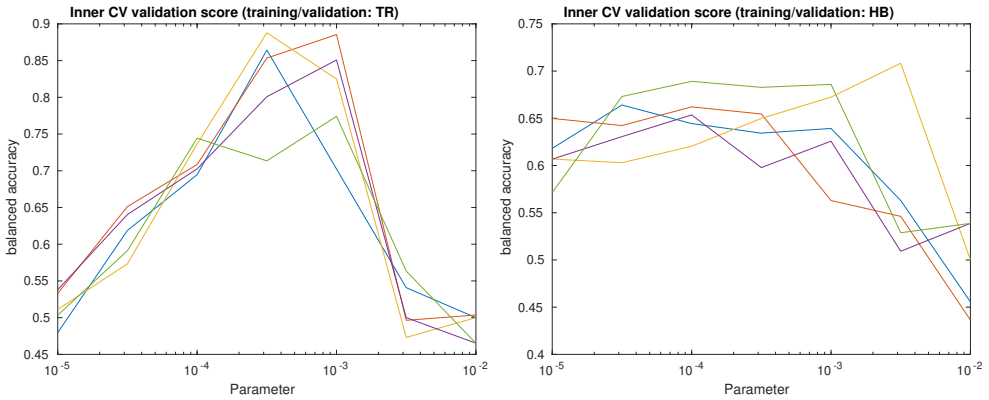


Figure 5.8: Parameter choice for different validation folds of the inner CV. While on HB, the performance is a little more consistent than on TR, the drop in balanced accuracy towards the upper and lower ends indicate a mostly suitable choice of parameter range. In practice, one could try to determine this range and choose the mean on the logarithmic scale as a simple heuristic.

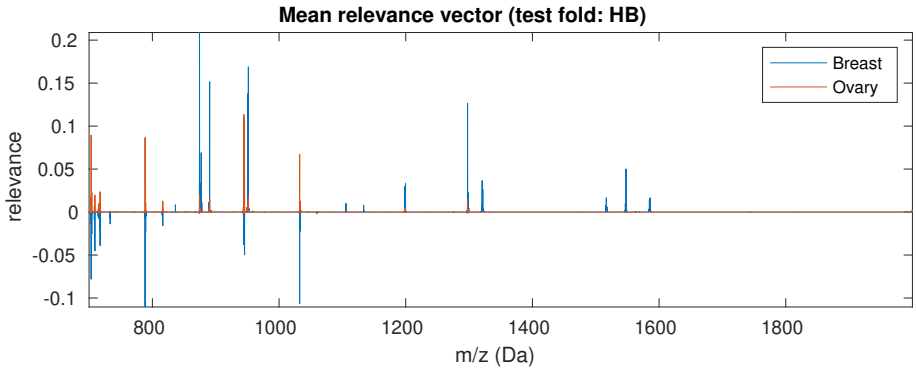


Figure 5.9: The network trained with deep relevance regularization produces a much sparser mean relevance map on the same test set as the unregularized network in Figure 5.6. This may mean that actual biomarkers are being learned instead of confounding factors. Interestingly, the trained neural network only seems to collect 'pro-ovary' evidence for the class 'ovary', whereas it collects both evidence and counter-evidence for class 'breast'.

Table 5.4: Balanced accuracies (spot-wise and patient-wise) for the unregularized NN, the deep relevance regularized NN and the LDA with ROC feature selection.

	NN	DRR-NN	ROC/LDA
balanced accuracy (spot level)	37.3%	77.4%	75.5%
balanced accuracy (patient level)	34.9%	80.6%	78.8%

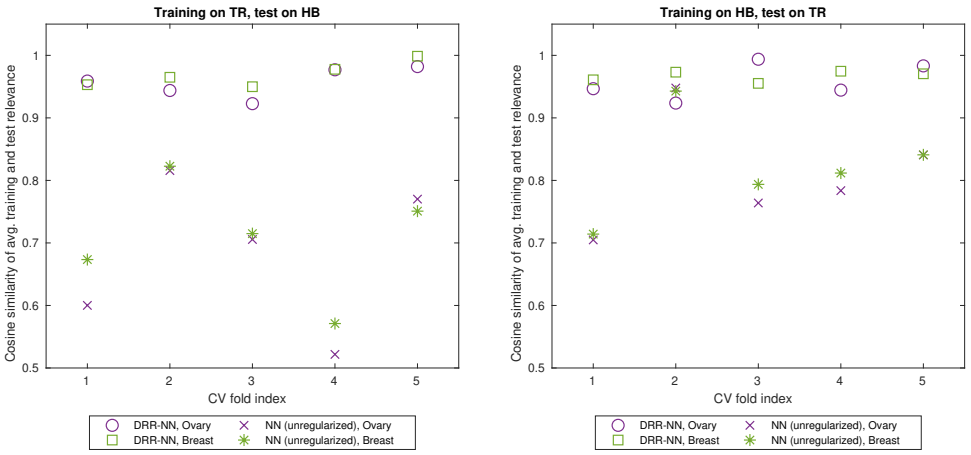


Figure 5.10: These plots show the cosine similarities between the mean relevance maps of the training sets (from one lab) and the respective test set (from the *other* lab), using the correct labels. Relevance maps of DRR-regularized NNs are considerably more consistent between training and test sets from different labs.

In Figure 5.10, we visualize the cosine similarity

$$\cos(u, v) = \frac{\langle u, v \rangle}{\|u\|_2 \|v\|_2}$$

between the average relevance map for one lab's training folds and the respective test fold of the other lab. The DRR-NNs' relevance maps exhibit a much higher *inter-lab* cosine similarity between training and test sets than their unregularized counterparts. When using an unregularized neural network, it seemingly tends to focus on different portions of the input for unseen data from a different lab. Assuming that relevance maps indeed highlight biologically relevant features, this explains the much better performance when applying DRR.

While we previously examined only average relevance maps, we now look at how consistently these m/z-values are taken into account by the network when using DRR. As exemplified for a high-relevance peak from Figure 5.9, in Figure 5.11 we observe that the relevances of the chosen peak are quite consistent throughout each class throughout the test set and that they are indeed highly indicative of the ground truth label.

5.6 Conclusion & Outlook

In this chapter, we identified the potential as well as failures of neural networks for classifying IMS data and were able to overcome these by employing a novel regularization strategy.

In the first study, a neural network architecture named *IsotopeNet* was introduced, which is adapted to the special characteristics of mass spectrometry data for tumor (sub-)typing. On two such datasets, this architecture was able to outperform both a classical peak picking approach as well as a more standard neural network architecture. A sensitivity

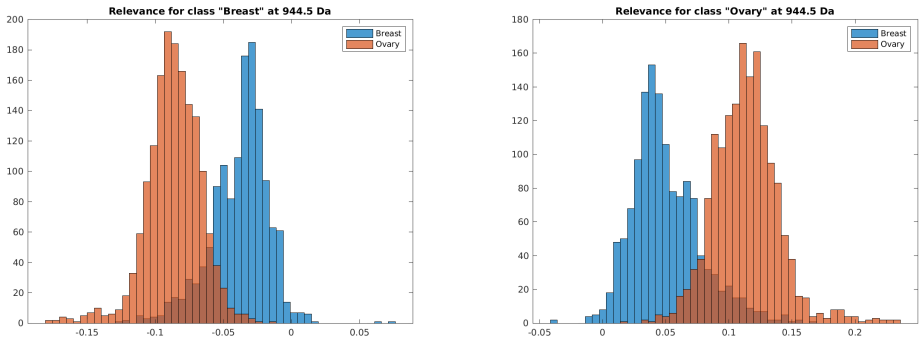


Figure 5.11: Histograms of relevances on the same test fold as in Figure 5.9 exemplified for a fixed m/z-value, where the two colors indicate the ground truth. There is a strong correlation between the ground truth class label and the relevance for each class.

analysis was able to confirm the plausibility of the model for one of the two datasets and uncovered possible data artifacts in the other dataset.

This aspect was further examined in the second study. We did so by employing layerwise relevance propagation, a method for assessing which parts of a mass spectrum contribute most to the classification decision, similar to the previous sensitivity analysis. Based on this assessment, we proposed a regularization strategy called *deep relevance regularization*, which enforces the classifier to base its decision on few m/z-values. In contrast to classical feature selection approaches, this happens individually for each sample. While the vanilla IsotopeNet without deep relevance regularization yields a very bad class-balanced accuracy of 37.3% on a challenging multi-centric tumor typing task, the thus-regularized version increases this score to 77.4%, beating a linear comparison method. Both the regularized neural network and the linear method were tuned using an extensive hyperparameter search with a nested inter-lab cross-validation method. The obtained relevance maps of the regularized networks were indeed much sparser than those of their unregularized counterparts, explaining the vastly improved performance.

Still, this work leaves much room for further considerations and possible extensions. Layerwise relevance propagation is just one of many possible attribution methods, as outlined in section 5.4.2. While layerwise relevance propagation is both theoretically well-founded and easy to implement for many models, there may be methods that are more suited to the task of IMS data classification.

Moreover, many different modifications to the exact form of the penalty are possible: Apart from simply using different norms than proposed here, one may penalize e.g. differences between class relevances in order to always take all classes into account.
The presented framework of deep relevance regularization also makes the incorporation of prior knowledge about measurable biomarkers possible: If one expects certain m/z-values (e.g. of specific peptides) to be of importance for the classification task, these m/z-values can simply be excluded (or assigned a lower penalty) than the remaining spectrum. This may be realized using a vector w of weights (e.g. consisting of zeros and ones), resulting in a modified penalty term $p(\lambda, w \odot r_\Theta(x, y))$.
Apart from classification, other tasks such as representation learning can be considered.

The layerwise relevance propagation of an autoencoder can be equipped with a penalty term, similarly to contractive autoencoders (Rifai et al., 2011). By weighting a-priori known m/z-values as proposed above, one would steer the resulting presentation towards a desired task – e.g. for biologically sensible dimensionality reduction before clustering.

Chapter 6

Conclusion

This thesis examined several topics loosely connected by the topic of double backpropagation.

While double backpropagation has been described for the first time almost 30 years ago at the time of writing, the recent years have seen a large increase in the number of articles dealing with this technique. In this work, a first extensive description of the involved phenomena was provided in Chapter 3. This was done in a general, coordinate-free Hilbert space framework. For this, a theory of adjoint operators of bilinear, continuous operators in Hilbert spaces was developed, which makes very short and elegant descriptions of the double backpropagation procedure possible. While it is trivially easy to compute these using automatic differentiation, this does not lend itself to gaining new insights about how to improve upon naïve double backpropagation. By closely examining the formulas, we were able to derive an algorithm that is applicable to practical models, which reduces the number of computations by roughly a third.
At first glance, it may be surprising that double backpropagation with ReLU activation functions actually works, since their gradients' jump discontinuities should make the optimization very difficult. Here, we show that batch optimization can alleviate these concerns, since this induces a 'pseudo-smoothing' effect on the loss landscape. Going forward, an extension of the theory for more general settings than Hilbert spaces was proposed. This might for instance be helpful for analyzing unrolled iterative schemes, e.g. on Banach spaces.

In Chapter 4, a so far unexplained phenomenon could be explained for the first time: Neural networks that have a high degree of robustness against adversarial perturbations seem to exhibit more structured or 'interpretable' saliency maps. These seemingly unrelated properties could be explained by looking at the alignment (roughly, the angle) between the model's input image and the output logit's gradient with respect to the input, the saliency map. In the linear, binary toy case, an increase of the distance to the nearest decision boundary will strictly lead to a lower angle between the image and the saliency map. For more general models such as neural networks, we were able to derive an estimation of how strongly these terms diverge. For this, a homogeneous decomposition theorem was employed, which could relate the *linearized robustness* to the alignment term. Empirically, this connection could be confirmed for models trained on ImageNet as well as MNIST, all of which were robustified using double backpropagation.

The used explanation was then furthermore applied to explaining the high degree of structure in the case of SmoothGrad saliency maps. For future work, we proposed to test the hypothesis, whether an increase in alignment will also lead to an increase in robustness, for which we propose a regularization term.

Chapter 5 dealt with the (sub-)typing of tumors based on mass spectra recorded from resected tumor tissue. For this, an architecture was developed, which is specifically adapted to the structure of mass spectra for tumor typing. By assuming that there exist *local* and *global* features, the architecture called *IsotopeNet* was designed to keep this in mind. The idea was to only increase the receptive field size induced by using convolutions and downsampling up to the size of the largest local patterns, i.e. isotope patterns. This model was able to improve upon the accuracy of both a linear baseline model as well as a more standard approach inspired by image classification on two tumor typing datasets. On a different, multi-laboratory study however, the performance suddenly dropped below guessing-level. This could be explained by employing the technique of *layerwise relevance propagation*, an attribution method which was able to uncover biologically unplausible regions. We hence deduced that the model apparently learns to distinguish the two types of tumor not based on real biomarkers, but rather based on measurement artifacts induced by the multi-laboratory setting. We next employed a sparsifying penalty term to the layerwise relevance propagation in the hopes of forcing the model to concentrate on few, biologically plausible peaks. This again amounted to using double backpropagation for the training of the thus-penalized loss. The penalty term increased the performance of the classifier greatly, resulting in a model which was able to beat the linear baseline again.

To conclude, double backpropagation is a very versatile technique which may appear for a diverse set of tasks. The author hopes that the presented theoretical examination can be a guideline to researchers wishing to understand the phenomena involved in double backpropagation.

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

Michaela Aichler and Axel Walch. MALDI imaging mass spectrometry: current frontiers and perspectives in pathology research and practice. *Laboratory investigation*, 95(4): 422, 2015.

Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 2016.

Theodore Alexandrov. Maldi imaging mass spectrometry: statistical data analysis and current computational challenges. *BMC bioinformatics*, 13(S16):S11, 2012.

Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. Towards better understanding of gradient-based attribution methods for deep neural networks. *arXiv preprint arXiv:1711.06104*, 2017.

Cem Anil, James Lucas, and Roger Grosse. Sorting out lipschitz function approximation. *arXiv preprint arXiv:1811.05381*, 2018.

Andre Araujo, Wade Norris, and Jack Sim. Computing receptive fields of convolutional neural networks. *Distill*, 2019. doi: 10.23915/distill.00021. https://distill.pub/2019/computing-receptive-fields.

Richard Arens. The adjoint of a bilinear operation. *Proceedings of the American Mathematical Society*, 2(6):839–848, 1951.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223, 2017.

Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

Aditya Balu, Thanh V Nguyen, Apurva Kokate, Chinmay Hegde, and Soumik Sarkar. A forward-backward approach for visualizing information flow in deep networks. *arXiv preprint arXiv:1711.06221*, 2017.

Viorel Barbu and Teodor Precupanu. *Convexity and optimization in Banach spaces*. Springer Science & Business Media, 2012.

Jens Behrmann, Christian Etmann, Tobias Boskamp, Rita Casadonte, Jörg Kriegsmann, and Peter Maaβ. Deep learning for tumor classification in imaging mass spectrometry. *Bioinformatics*, 34(7):1215–1223, 2017.

Jens Behrmann, Christian Etmann, Tobias Boskamp, Rita Casadonte, Jörg Kriegsmann, and Peter Maass. Deep learning for tumor classification in imaging mass spectrometry. *Bioinformatics*, 34(7):1215–1223, 04 2018. ISSN 1367-4803. doi: 10.1093/bioinformatics/btx724.

Jens Behrmann, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, and Joern-Henrik Jacobsen. Invertible residual networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 573–582, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

Mariusz Bojarski, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Larry Jackel, Urs Muller, and Karol Zieba. Visualbackprop: efficient visualization of cnns. *arXiv preprint arXiv:1611.05418*, 2016.

Jérôme Bolte and Edouard Pauwels. Conservative set valued fields, automatic differentiation, stochastic gradient method and deep learning. *arXiv preprint arXiv:1909.10300*, 2019.

Thomas Bonesky, Kamil S Kazimierski, Peter Maass, Frank Schöpfer, and Thomas Schuster. Minimization of tikhonov functionals in banach spaces. In *Abstract and Applied Analysis*, volume 2008. Hindawi, 2008.

Tobias Boskamp, Delf Lachmund, Janina Oetjen, Yovany Cordero Hernandez, Dennis Trede, Peter Maass, Rita Casadonte, Jörg Kriegsmann, Arne Warth, Hendrik Dienemann, et al. 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.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

Kristian Bredies and Dirk Lorenz. *Mathematical Image Processing*. Springer International Publishing, 2018. doi: 10.1007/978-3-030-01458-2.

Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1872–1886, 2013.

Achim Buck, Bram Heijs, Birte Beine, Jan Schepers, Alberto Cassese, Ron MA Heeren, Liam A McDonnell, Corinna Henkel, Axel Walch, and Benjamin Balluff. Round

robin study of formalin-fixed paraffin-embedded tissues in mass spectrometry imaging. *Analytical and bioanalytical chemistry*, 410(23):5969–5980, 2018.

Richard M Caprioli, Terry B Farmer, and Jocelyn Gile. Molecular imaging of biological samples: localization of peptides and proteins using MALDI-tof ms. *Analytical chemistry*, 69(23):4751–4760, 1997.

Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 39–57, 2017.

R Casadonte, R Longuespee, J Kriegsmann, and M Kriegsmann. Maldi ims and cancer tissue microarrays. In *Advances in cancer research*, volume 134, pages 173–200. Elsevier, 2017.

Rita Casadonte, Mark Kriegsmann, Friederike Zweynert, Katrin Friedrich, Gustavo Bretton, Mike Otto, Sören-Oliver Deininger, Rainer Paape, Eckhard Belau, Detlev Suckau, et al. Imaging mass spectrometry to discriminate breast from pancreatic cancer metastasis in formalin-fixed paraffin-embedded tissues. *Proteomics*, 14(7-8): 956–964, 2014.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1310–1320, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

Richard B Cole. *Electrospray and MALDI mass spectrometry: fundamentals, instrumentation, practicalities, and biological applications*. John Wiley & Sons, 2011.

Yovany Cordero Hernandez, Tobias Boskamp, Rita Casadonte, Lena Hauberg-Lotte, Janina Oetjen, Delf Lachmund, Annette Peter, Dennis Trede, Katharina Kriegsmann, Mark Kriegsmann, et al. Targeted feature extraction in MALDI mass spectrometry imaging to discriminate proteomic profiles of breast and ovarian cancer. *PROTEOMICS–Clinical Applications*, 13(1):1700168, 2019.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20 (3):273–297, 1995.

Wojciech M Czarnecki, Simon Osindero, Max Jaderberg, Grzegorz Swirszcz, and Razvan Pascanu. Sobolev training for neural networks. In *Advances in Neural Information Processing Systems*, pages 4278–4287, 2017.

Sören-Oliver Deininger, Dale S Cornett, Rainer Paape, Michael Becker, Charles Pineau, Sandra Rauser, Axel Walch, and Eryk Wolski. Normalization in MALDI-tof imaging datasets of proteins: practical considerations. *Analytical and bioanalytical chemistry*, 401(1):167–181, 2011.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

Sören Dittmer, Tobias Kluth, Peter Maass, and Daniel Otero Baguer. Regularization by architecture: A deep prior approach for inverse problems. *Journal of Mathematical Imaging and Vision*, Oct 2019. ISSN 1573-7683. doi: 10.1007/s10851-019-00923-x.

David L Donoho et al. Compressed sensing. *IEEE Transactions on information theory*, 52(4):1289–1306, 2006.

Harris Drucker and Yann Le Cun. Improving generalization performance using double backpropagation. *IEEE Transactions on Neural Networks*, 3(6):991–997, 1992.

Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. Incorporating second-order functional knowledge for better option pricing. In *Advances in neural information processing systems*, pages 472–478, 2001.

Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

Gamaleldin Fathy Elsayed, Dilip Krishnan, Hossein Mobahi, Kevin Regan, and Samy Bengio. Large margin deep networks for classification. 2018.

Christian Etmann. Classification methods for mass spectrometry data with application to tumor typing. Master's thesis, University of Bremen, 2016.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

Manuel Galli, Italo Zoppis, Andrew Smith, Fulvio Magni, and Giancarlo Mauri. Machine learning approaches in MALDI-msi: clinical applications. *Expert review of proteomics*, 13(7):685–696, 2016.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv 1412.6572*, 2014.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Will Grathwohl, Ricky TQ Chen, Jesse Betterncourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in neural information processing systems*, pages 5767–5777, 2017.

Trevor Hastie, Robert Tibshirani, and Martin Wainwright. *Statistical learning with sparsity: the lasso and generalizations*. Chapman and Hall/CRC, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Matthias Hein and Maksym Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. In *Advances in Neural Information Processing Systems*, pages 2266–2276, 2017.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Paolo Inglese, James S McKenzie, Anna Mroz, James Kinross, Kirill Veselkov, Elaine Holmes, Zoltan Takats, Jeremy K Nicholson, and Robert C Glen. Deep learning and 3d-desi imaging reveal the hidden metabolic heterogeneity of cancer. *Chemical science*, 8(5):3500–3511, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Daniel Jakubovitz and Raja Giryes. Improving DNN robustness to adversarial attacks using jacobian regularization. In *The European Conference on Computer Vision (ECCV)*, September 2018.

Bangti Jin and Peter Maass. Sparsity regularization for parameter identification problems. *Inverse Problems*, 28(12):123001, 2012.

Simran Kaur, Jeremy Cohen, and Zachary C Lipton. Are perceptually-aligned gradients a general property of robust classifiers? *arXiv preprint arXiv:1910.08640*, 2019.

Beomsu Kim, Junghoon Seo, and Taegyun Jeon. Bridging adversarial robustness and gradient interpretability. *CoRR*, abs/1903.11626, 2019.

Pieter-Jan Kindermans, Kristof Schütt, Klaus-Robert Müller, and Sven Dähne. Investigating the influence of noise and distractors on the interpretation of neural networks. *arXiv preprint arXiv:1611.07270*, 2016.

Pieter-Jan Kindermans, Kristof T Schütt, Maximilian Alber, Klaus-Robert Müller, and Sven Dähne. Patternnet and PatternLRP – improving the interpretability of neural networks. *arXiv preprint arXiv:1705.05598*, 2017.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10215–10224, 2018.

Jörg Kriegsmann, Mark Kriegsmann, and Rita Casadonte. MALDI tof imaging mass spectrometry in clinical pathology: a valuable tool for cancer diagnostics. *International journal of oncology*, 46(3):893–906, 2015.

Mark Kriegsmann, Rita Casadonte, Jörg Kriegsmann, Hendrik Dienemann, Peter Schirmacher, Jan Hendrik Kobarg, Kristina Schwamborn, Albrecht Stenzinger, Arne Warth, and Wilko Weichert. Reliable entity subtyping in non-small cell lung cancer by matrix-assisted laser desorption/ionization imaging mass spectrometry on formalin-fixed paraffin-embedded tissue specimens. *Molecular & Cellular Proteomics*, 15(10): 3081–3089, 2016.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

Serge Lang. *Real and Functional Analysis*. Springer Science & Business Media, Berlin Heidelberg, 3rd edition, 1993.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436, 2015.

Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788, 1999.

Johannes Leuschner, Maximilian Schmidt, Pascal Fernsel, Delf Lachmund, Tobias Boskamp, and Peter Maass. Supervised non-negative matrix factorization methods for MALDI imaging applications. *Bioinformatics*, 35(11):1940–1947, 2018.

Alexander Levine, Sahil Singla, and Soheil Feizi. Certifiably robust interpretation in deep learning. *arXiv preprint arXiv:1905.12105*, 2019.

Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. *Master's Thesis (in Finnish), Univ. Helsinki*, pages 6–7, 1970.

Sebastian Lunz, Ozan Öktem, and Carola-Bibiane Schönlieb. Adversarial regularizers in inverse problems. In *Advances in Neural Information Processing Systems*, pages 8507–8516, 2018.

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML Workshop on Deep Learning for Audio, Speech, and Language Processing*, 2013.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. 2018.

Sebastian Mika, Gunnar Ratsch, Jason Weston, Bernhard Scholkopf, and Klaus-Robert Mullers. Fisher discriminant analysis with kernels. In *Neural networks for signal processing IX: Proceedings of the 1999 IEEE signal processing society workshop (cat. no. 98th8468)*, pages 41–48. Ieee, 1999.

Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, 65:211–222, 2017a.

Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 2017b.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582, 2016.

Boris S Mordukhovich. *Variational analysis and generalized differentiation I: Basic theory*, volume 330. Springer Science & Business Media, 2006.

Mervin E Muller. A note on a method for generating points uniformly on n-dimensional spheres. *Communications of the ACM*, 2(4):19–20, 1959.

Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

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

Adam Noack, Isaac Ahern, Dejing Dou, and Boyang Li. Does interpretability of neural networks imply adversarial robustness? *arXiv preprint arXiv:1912.03430*, 2019.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl Dickstein. On the expressive power of deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2847–2854. JMLR. org, 2017.

Jonas Rauber, Wieland Brendel, and Matthias Bethge. Foolbox v0. 8.0: A python toolbox to benchmark the robustness of machine learning models. *arXiv preprint arXiv:1707.04131*, 2017.

Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

Laura Rieger, Chandan Singh, W James Murdoch, and Bin Yu. Interpretations are useful: penalizing explanations to align neural networks with prior knowledge. *arXiv preprint arXiv:1909.13584*, 2019.

Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 833–840. Omnipress, 2011.

Andrew Slavin Ross, Michael C. Hughes, and Finale Doshi-Velez. Right for the right reasons: Training differentiable models by constraining their explanations. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 2662–2670, 2017. doi: 10.24963/ijcai.2017/371.

Kevin Roth, Aurelien Lucchi, Sebastian Nowozin, and Thomas Hofmann. Stabilizing training of generative adversarial networks through regularization. In *Advances in neural information processing systems*, pages 2018–2028, 2017.

W. Rudin. *Functional Analysis*. International series in pure and applied mathematics. McGraw-Hill, 2nd edition, 1991. ISBN 9780070542365.

Eric Schechter. *Handbook of Analysis and its Foundations*. Academic Press, 1996.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

Frank Schöpfer, Alfred Karl Louis, and Thomas Schuster. Nonlinear iterative methods for linear ill-posed problems in banach spaces. *Inverse problems*, 22(1):311, 2006.

Lukas Schott, Jonas Rauber, Wieland Brendel, and Matthias Bethge. Robust perception through analysis by synthesis. *CoRR*, abs/1805.09190, 2018.

Ramprasaath R Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-CAM: Why did you say that? visual explanations from deep networks via gradient-based localization. *arXiv preprint arXiv:1610.02391*, 2016.

Michael W Senko, Steven C Beu, and Fred W McLaffertycor. 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.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

Avanti Shrikumar, Peyton Greenside, Anna Shcherbina, and Anshul Kundaje. Not just a black box: Learning important features through propagating activation differences. *arXiv preprint arXiv:1605.01713*, 2016.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362 (6419):1140–1144, December 2018. doi: 10.1126/science.aar6404.

Barry Simon. *A comprehensive course in analysis*. American Mathematical Society Providence, Rhode Island, 2015.

Carl-Johann Simon-Gabriel, Yann Ollivier, Bernhard Schölkopf, Léon Bottou, and David Lopez-Paz. Adversarial vulnerability of neural networks increases with input dimension. *arXiv preprint arXiv:1802.01421*, 2018.

Carl-Johann Simon-Gabriel, Yann Ollivier, Leon Bottou, Bernhard Schölkopf, and David Lopez-Paz. First-order adversarial vulnerability of neural networks and input dimension. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5809–5817, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: Removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017.

Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472. IEEE, 2017.

Jure Sokolić, Raja Giryes, Guillermo Sapiro, and Miguel RD Rodrigues. Robust large margin deep neural networks. *IEEE Transactions on Signal Processing*, 65(16):4265–4280, 2017.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

J.T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. In *ICLR (workshop track)*, 2015.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Markus Stoeckli, Pierre Chaurand, Dennis E Hallahan, and Richard M Caprioli. Imaging mass spectrometry: a new technology for the analysis of protein expression in mammalian tissues. *Nature medicine*, 7(4):493, 2001.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. *arXiv preprint arXiv:1703.01365*, 2017.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. 2014.

Spencer A Thomas, Alan M Race, Rory T Steven, Ian S Gilmore, and Josephine Bunch. Dimensionality reduction of mass spectrometry imaging data using autoencoders. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7. IEEE, 2016.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. In *International Conference on Learning Representations*, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Thomas Wiatowski and Helmut Bölcskei. A mathematical theory of deep convolutional neural networks for feature extraction. *IEEE Transactions on Information Theory*, 64 (3):1845–1866, 2017.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

Renato Zenobi and Richard Knochenmuss. Ion formation in MALDI mass spectrometry. *Mass spectrometry reviews*, 17(5):337–366, 1998.

Jianming Zhang, Zhe Lin, Jonathan Brandt, Xiaohui Shen, and Stan Sclaroff. Top-down neural attention by excitation backprop. In *European Conference on Computer Vision*, pages 543–559. Springer, 2016.

Luisa M Zintgraf, Taco S Cohen, Tameem Adel, and Max Welling. Visualizing deep neural network decisions: Prediction difference analysis. *arXiv preprint arXiv:1702.04595*, 2017.

Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 67(2):301–320, 2005.

Appendix A

Generalizations of Adjoint Operators

As mentioned in Section 3.2.1, Arens (1951) presents a similar construction to the *adjoints* for bilinear, continuous operators introduced in chapter 3, which we will call the *Arens-adjoint* in the following.

Let $K : \mathcal{P} \times \mathcal{X} \to \mathcal{Y}$ be bilinear and continuous, where $\mathcal{P}$, $\mathcal{X}$ and $\mathcal{Y}$ are normed spaces. Then Arens (1951) defines the adjoint of K to be the function

$$K' : \mathcal{Y}' \times \mathcal{P} \to \mathcal{X}'$$

defined by $K'(\gamma, \theta)(x) := \gamma(K(\theta, x))$, which can be shown to be bilinear and continuous. Here, $\mathcal{Y}'$ and $\mathcal{X}'$ denote the respective dual spaces of $\mathcal{Y}$ and $\mathcal{X}$. The Arens-adjoint of K' is then a function

$$K'' : \mathcal{X}'' \times \mathcal{Y}' \to \mathcal{P}',$$

defined in the same way as K'.
Hilbert spaces are isomorphic to their dual space (due to the Riesz representation theorem), and consequently to their bidual space. If $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{P}$ are Hilbert spaces, then K' can hence be identified with K^T and K'' can be identified with $K^\square$ (which follows from the uniqueness of K^T and $K^\square$ up to the order of arguments).

Appendix B

Additional Information for MALDI Experiments

B.1 Receptive Field Size Choice for IsotopeNet

As mentioned in Section 5.2, the receptive field size of *IsotopeNet* is determined by the size of large isotopic patterns of peptides within the mass spectra. According to our specifications, the receptive fields of the local feature extractors should cover even the largest observable isotope patterns, since these are considered to be the largest *local* structures within the data. Hence, the size of isotope patterns in terms of m/z-bins is to be determined. It has to be kept in mind that the m/z-resolution decreases over the m/z-axis.

Senko et al. (1995) proposed a simple model for isotope patterns of peptides: An average amino acid called *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 isotopic rates of carbon. We set a threshold $t_{iso} > 0$ to cut off very unlikely, insignificant isotopes and represent each peak with a Gaussian filter. Here, we made the calculations based on task ADSQ (see Section 5.3.1), but point out that the other two considered datasets have comparable m/z resolutions.

Finally, this model allows a computation of the number of m/z-bins for the estimated isotope patterns, as shown in Figure B.1 (in blue). The receptive field size of *IsotopeNet* (43 m/z-bins) is shown as the red line in Figure B.1. Hence, the used receptive field is slightly larger than the estimated isotope patterns, which fulfills our original requirement of IsotopeNet.

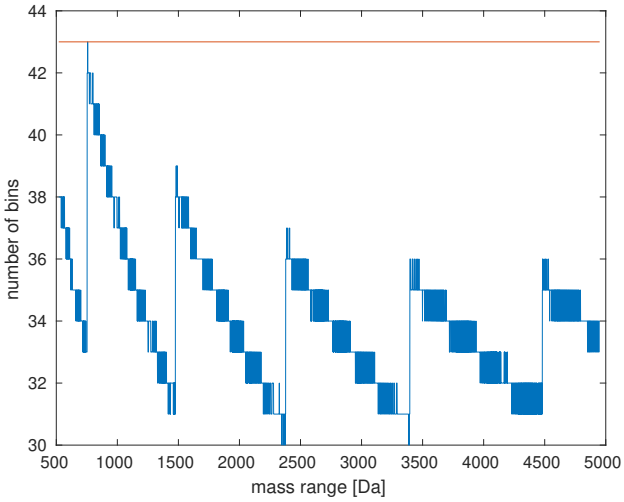


Figure B.1: Number of bins per isotope pattern over the course of the m/z-axis, shown in blue. Since the m/z-resolution decreases, the number of bins decreases as well. The observable jumps are due to threshold t_{iso}, which cuts of very unlikely, insignificant isotopes. The receptive field size of IsotopeNet is depicted in red, such that IsotopeNet should be able to encode all isotopic patterns of interest.

B.2 MALDI Dataset Acquisition

B.2.1 ADSQ/LP

Sample acquisition, preparation, MALDI-IMS measurement and data preprocessing are described in more detail in Boskamp et al., 2017.

Tissue Preparation

FFPE samples were provided by the tissue bank of the National Center for Tumor Diseases (NCT, Heidelberg, Germany). Tumor status and typing of all cores were confirmed by standard histopathological examination of hematoxylin and eosin (HE) stained slides and additional immunohistochemical stains. Cylindrical tissue cores of all tissue samples were assembled to 12 TMA blocks, such as the one in Figure 5.1 (left). Tissue sample preparation for MALDI-IMS measurement was performed according to a previously published protocol (Zenobi and Knochenmuss, 1998), including tryptic digestion of proteins to peptides.

IMS Measurement

After the application of a MALDI matrix solution onto digested sections, MALDI-IMS data was acquired using a MALDI-time-of-flight (TOF) instrument (Autoflex speed, Bruker Daltonik) in positive ion reflector mode. Spectra were measured in the mass range of 500-5000 m/z at 150 μm spacing between spot centers using 1600 laser shots per position.

Preprocessing

After measurement, the raw MALDI-IMS data were combined into a single dataset using the SCiLS Lab software (version 2016a, SCiLS, Bremen, Germany), followed by baseline correction using the default settings. Next, the data was imported into MATLAB R2016b (Mathworks, Natick, MA, USA) for further analysis using our MATLAB library `MSClassifyLib`.

B.2.2 Ovary/Breast Dataset

Tissue Preparation

Here, we give a brief description of the used dataset, which is a subset of the data used in (Cordero Hernandez et al., 2019). Readers interested in the minutiae of the data acquisition protocols are referred to the original publication.
FFPE tissue samples from breast carcinoma ($N = 99$ patients, all human epidermal growth factor (Her2) positive) and ovarian carcinoma ($N = 84$ patients, various kinds) were kindly provided by the University Hospital Heidelberg in accordance with the regulations of the local ethics committee. The cancer biopsies were assembled to four tissue microarray (TMA) blocks as cylindrical tissue cores with 1 mm diameter. Two of these TMAs consisted solely of breast tissue, while the other two consisted solely of ovarian tissue[1]. Slices of 5 μm thickness were washed in an ethanol series and antigen retrieval was performed in a Tris buffer. Tryptic digestion was performed on-tissue. CHCA matrix was applied with an ImagePrep device (Bruker Daltonik, Bremen, Germany).

IMS Measurement

The same TMAs were measured in two different laboratories, one in Bremen (HB) and one in Trier (TR). In both cases, the measurements were performed with an autoflex speed MALDI mass spectrometer (Bruker) in positive-ion mode. Mass spectra were collected with 100 μm spacing between spot centers. An external calibration was performed using Peptide Calibration Standard II (Bruker). Mass resolving power was approx. R = 11 000. The mass accuracy was visually estimated to be approx. 50 – 100 ppm.

Afterwards, the matrix was removed with 100% methanol, which was followed by an H&E staining. A pathologist annotated regions of high tumor concentration, which were transferred to the recorded MALDI measurement on the level of single IMS spots and henceforth used as labels for the data analysis. Note that this was done separately for each section measured in the two laboratories, such that their annotated regions differ. After this procedure, the HB measurement consisted of 5230 spectra of breast tumors and 6777 spectra of ovarian tumors. For TR, 3479 points were assigned the label 'breast' and 4621 were assigned the label 'ovary'.

Preprocessing

A baseline correction of the MALDI IMS data was performed using SCiLS Lab (version 2017a, SCiLS, Bremen, Germany) with default settings. Next, the data was imported into MATLAB (version 2018a, MathWorks, Natick, MA, USA), and for this study reduced to the m/z-range 800 Da – 2000 Da. The spectra were subsequently normalized by total ion count.

[1]Note that Cordero Hernandez et al. (2019) used 5 TMA blocks. We discarded one TMA of ovarian tissue samples in order to take the stark class imbalance out of the equation.

B.3 MALDI Residual Comparison Architecture

Table B.1: Architecture of the Residual Network. The 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	2	5	3	128
Residual layer	2	5	1	128
Residual layer	2	5	3	128
Residual layer	2	5	1	128
Residual layer	2	5	3	128
Residual layer	2	5	1	128
Residual layer	2	5	3	128
Residual layer	2	5	1	128
Residual layer	2	5	3	128
Residual layer	2	5	1	128
Residual layer	2	5	3	128
Residual layer	2	5	3	256
GlobalPool layer	-	-	-	1
Fully connected (softmax)	-	-	-	1