Technische Universität Berlin

Fakultät IV
Dept. of Computer Engineering and Microelectronics
Remote Sensing Image Analysis Group

Master Thesis

Deep Generative Models for Nonlinear Inverse Problems and Phase Retrieval

Martin Reiche
Matriculation Number 357793

Submitted on September 19, 2019

Supervisors:

Prof. Dr. Begüm Demir
Technische Universität Berlin

Prof. Dr. Heinz-Wilhelm Hübers
Deutsches Zentrum für Luft- und Raumfahrt
Humboldt-Universität zu Berlin

Advisors:

Dr. Peter Jung
Technische Universität Berlin

Dr. Sven Augustin
Deutsches Zentrum für Luft- und Raumfahrt
Humboldt-Universität zu Berlin

Eidesstattliche Versicherung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Die selbständige und eigenständige Anfertigung versichert an Eides statt:

Berlin, September 19, 2019

Martin Reiche

Zusammenfassung

Diese Arbeit beschäftigt sich mit der Frage, wie generative neuronale Modelle zur Lösung nichtlinearer inverser Probleme eingesetzt werden können. Der Fokus liegt dabei auf dem *generalisierten Phase Retrieval-Problem*, bei dem ein Signal $\mathbf{x}$ auf Basis der Intensitäten $|\langle \mathbf{a}_i, \mathbf{x} \rangle|^2$ linearer Messungen rekonstruiert werden soll.

Die Arbeit stellt zwei neue Algorithmen vor: *Deep Regularized Gradient Descent* nutzt trainierte generative neuronale Netze als A-priori-Information über die Domäne des Signals in einem durch Total-Variation regularisierten Gradientenabstiegsverfahren. Dabei erzielt der Algorithmus bereits bei niedrigen Samplingraten eine höhere Rekonstruktionsqualität als herkömmliche Verfahren, kann jedoch bei höheren Samplingraten bei gleichzeitigem hohem Modellfehler des Generators keine *sehr hohe* Rekonstruktionsqualität erreichen. Das zweite vorgestellte Verfahren, *Deep Regularized Gradient Descent-initialized Randomized Kaczmarz*, überwindet diesen Nachteil dadurch, dass es *Deep Regularized Gradient Descent* nur für die Ermittlung eines geeigneten Startwerts verwendet und mit diesem das *Randomized Kaczmarz*-Verfahren ausführt. In einer empirischen Evaluation kann gezeigt werden, dass dieses Vorgehen sowohl höhere Rekonstruktionsqualität bei niedrigen Samplingraten als auch *sehr hohe* Rekonstruktionsqualität bei hohen Samplingraten erzielt. Das Verfahren hat zudem ein besseres Laufzeitverhalten als herkömmliche Gradienten-basierte Rekonstruktionsalgorithmen.

Des Weiteren zeigt diese Arbeit, dass *Deep Regularized Gradient Descent-initialized Randomized Kaczmarz* auch für das Problem des *Terahertz Ein-Pixel Phase Retrieval* angewendet werden kann. Dabei soll ein Signal auf der Basis seiner reellwertigen linearen Messungen bei gleichzeitigen starken Diffraktionseffekten rekonstruiert werden. Die in dieser Arbeit durchgeführte Evaluation auf simulierten Daten zeigt, dass Rekonstruktionsmethoden auf der Basis generativer neuronaler Modelle auch für praktische Probleme anwendbar sind.

Abstract

This thesis explores how deep generative models can be utilized for solving nonlinear inverse problems such as *generalized phase retrieval*, in which one wants to reconstruct a signal $\mathbf{x}$ from the squared magnitudes $|\langle \mathbf{a}_i, \mathbf{x} \rangle|^2$ of its linear measurements.

The thesis introduces two new algorithms: *Deep Regularized Gradient Descent* uses a trained deep generative model as a data prior in a total variation-regularized gradient descent scheme to reconstruct a signal in the range of that generative model. The thesis empirically shows that this method achieves higher reconstruction quality than conventional generalized phase retrieval algorithms already at very low sampling rates. It however fails to deliver *very high* reconstruction results at high sampling rates when there is significant generator model error. *Deep Regularized Gradient Descent-initialized Randomized Kaczmarz* overcomes this shortcoming by using *Deep Regularized Gradient Descent* only to calculate an initial value which is then fed into the *Randomized Kaczmarz* method to reconstruct the signal $\mathbf{x}$. The thesis empirically shows that *Deep Regularized Gradient Descent-initialized Randomized Kaczmarz* combines higher reconstruction quality at low sampling rates with *very high* reconstruction quality at high sampling rates. The algorithm also demonstrates superior runtime performance over conventional gradient-based reconstruction methods.

The thesis furthermore experimentally validates the applicability of *Deep Regularized Gradient Descent-initialized Randomized Kaczmarz* on the practical problem of *terahertz single-pixel phase retrieval*, in which a signal is to be reconstructed from its real-valued measurements under strong diffraction effects. This experimental evaluation on simulated data gives first evidence that reconstruction methods using deep generative models as data priors are suited for real-world reconstruction problems.

Contents

Notation

$\mathbf{a}$	A vector
$\mathbf{A}$	A matrix
$\mathbf{A}_{i,j}$	Element at position (i, j) of matrix $\mathbf{A}$
$\mathbf{a}^\top$, $\mathbf{A}^\top$	Transpose of a vector $\mathbf{a}$ or a matrix $\mathbf{A}$
$\mathbf{a}^\mathrm{H}$, $\mathbf{A}^\mathrm{H}$	Conjugate transpose of a vector $\mathbf{a}$ or matrix $\mathbf{A}$
$\overline{\mathbf{a}}$, $\overline{\mathbf{A}}$	The entrywise complex conjugate of $\mathbf{a}$ or $\mathbf{A}$
$\langle \mathbf{a}, \mathbf{b} \rangle$	Inner product $\mathbf{a}^\mathrm{H}\mathbf{b}$ of $\mathbf{a}$ and $\mathbf{b}$
$\mathbf{A}^\dagger$	Moore-Penrose pseudoinverse of the matrix $\mathbf{A}$
$\mathbf{I}$	The identity matrix
$\frac{\partial f}{\partial x}$	Partial derivative of f with respect to x
∇f	Gradient of f
$a \sim p$	Random variable a has distribution p
$\mathcal{N}(0, 1)$	Gaussian distribution with mean 0 and standard deviation 1
$\hat{p}(X)$	Empirical distribution $\hat{p}$ of a random variable X
$\mathcal{D}(\mathcal{A})$	Domain of operator $\mathcal{A}$
$\mathcal{R}(\mathcal{A})$	Range of operator $\mathcal{A}$
$\mathbf{a} \odot \mathbf{b}$	Point-wise multiplication operator
$\sqrt{\mathbf{a}}$	Point-wise square root operator
$\lvert\mathbf{a}\rvert$	Point-wise modulus/absolute operator
$\lVert\mathbf{a}\rVert$, $\lVert\mathbf{a}\rVert_2$	ℓ^2-norm of the vector $\mathbf{a}$
$\lVert\mathbf{A}\rVert_\mathrm{F}$	Frobenius-norm of the matrix $\mathbf{A}$
$\mathcal{F}\mathbf{x}$	Discrete Fourier transform of $\mathbf{x}$
$\mathrm{Tr}(\mathbf{X})$	Trace of matrix $\mathbf{X}$
$\mathbf{X} \succeq 0$	Matrix $\mathbf{X}$ is positive semidefinite

$f = \mathcal{O}(g)$	Function f is in the order of function g, i.e. the growth rate of f is upper-bounded by g
$\mathrm{Re}(x)$	Real part of complex variable x

Chapter 1

Introduction and Overview

Inverse problems of the form

$$\text{find} \quad \mathbf{x} \quad \text{s.t.} \quad \mathbf{y} = \mathcal{A}(\mathbf{x}) \tag{1.1}$$

(where a measurement $\mathbf{y}$ and a map $\mathcal{A}$ are given) are ubiquitous in a lot of scientific disciplines and practical applications.

In computer vision one might want to revert an operation $\mathcal{A}$ that has been applied to an image $\mathbf{x}$. In some rare cases, the operation $\mathcal{A}$ will be invertible (for example inverting the intensities in a black & white image), but most of the time it will not be (imagine trying to revert a *from color to gray scale* operation).

This example shows an important property of inverse problems: in most cases their exact solutions are either not existent or there exist infinitely many of them. In order to overcome this, one needs to rephrase those problems so that the existence and uniqueness of a solution can be guaranteed. A convenient way to do so is to restrict the signals to a certain well-defined domain. The goal is to make the resulting space of possible solutions small enough so that a unique solution exists.

Deep learning is a technique to learn approximations of functions by *training* networks of biologically inspired computational units (usually referred to as *neurons*). The most fundamental class of a neural network, the *feed-forward network*, groups neurons together at individual layers, allowing for stacked network architectures which are commonly used in the classical tasks solved by neural networks: *regression*, *classification* and *generation*. An example of a feed-forward neural network for classification is given in Figure 1.1. The architecture of the neural network (the number of layers, number of neurons per layer or number of connections between one layer and the other) defines its expressive power (Raghu et al., 2016), and different architectures are more or less useful for different kinds of problems.

We will exemplify one particular problem here, which is to learn a (finite) domain $\mathcal{X}$ of permissible signals $\mathbf{x} \in \mathcal{X} \subset X$ of a space X. We can design a neural network in such a way that it can learn to *encode* a signal $\mathbf{x} \in \mathcal{X}$ so that when it is given a corrupted signal $\tilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\epsilon} \notin \mathcal{X}$ the network is able to reconstruct the original $\mathbf{x}$. Such a neural network is called a (denoising) *autoencoder* and is schematically depicted in Figure 1.2. To push this further, neural networks can also be used to learn *infinite* domains of permissible signals and map them to a continuous distribution of a low-dimensional input vector in

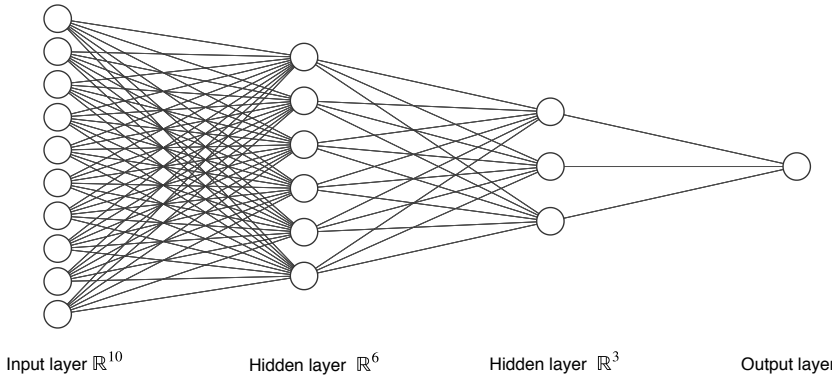


Figure 1.1: Schematic example of a fully-connected feed-forward network for *binary* classification, where the sign of the output value denotes the predicted class.

such a way that when sampling from this distribution and feeding this sample into another neural network we are able to easily *generate* permissible signals. We call such a model a *deep generative model*.

This thesis will detail how *deep generative models* can be utilized in existing algorithms for solving a very specific type of inverse problem called *generalized phase retrieval*, in which the goal is to recover a signal $\mathbf{x} \in \mathbb{R}^n$ from the squared modulus of m (complex) linear measurements, i.e., in the noiseless case:

$$y_i = |\langle \mathbf{a}_i, \mathbf{x} \rangle|^2, \ \mathbf{a}_i \in \mathbb{C}^n, \ i = 1, ..., m \tag{1.2}$$

or alternatively written in matrix notation as

$$\mathbf{y} = |\mathbf{A}\mathbf{x}|^2, \ \mathbf{A} \in \mathbb{C}^{m \times n} \tag{1.3}$$

where $\mathbf{a}_i$ are the rows of $\mathbf{A}$.

This thesis is structured as follows: the remainder of this chapter introduces *nonlinear inverse problems*, *(generalized) phase retrieval* and *deep generative models*, and examines prior art in these fields of research. Chapter 2 introduces some important concepts of numerical optimization, especially gradient-based local optimization methods, that can be used to solve nonlinear optimization problems and are of particular interest in this thesis. Chapter 3 formally introduces the basics of *machine learning* and goes into detail about *deep generative models* with a strong focus on *variational autoencoders*, their properties and training methods. Chapter 4 is about *inverse problems*, formalizes the *generalized phase retrieval problem* and introduces the most popular methods for its solution. Chapter 5 explains how some of the existing methods to solve the *generalized phase retrieval problem* can be extended to incorporate signal domain information encoded in deep generative networks. It discusses two important ways to do so, namely by adding deep generative models as signal priors in the optimization process and by using deep generative models for better initialization of other reconstruction methods. The chapter concludes with the introduction of two new algorithms for generalized phase retrieval based on the

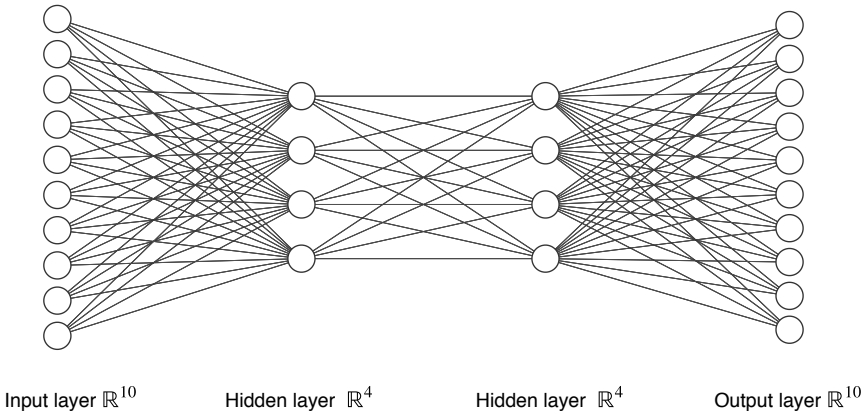


Figure 1.2: Schematic example of a fully-connected autoencoder for a signal $\mathbf{x} \in \mathbb{R}^{10}$.

aforementioned ideas. Chapter 6 provides a detailed empirical analysis of the stability of the methods introduced in Chapter 5 in a practical application for the real-world scenario of *terahertz single-pixel phase retrieval*. It therefore plays an important part in ensuring the practicality of the proposed methods in real-world scenarios. The thesis concludes with Chapter 7, which summarizes the main findings.

1.1 Nonlinear Inverse Problems in Real-World Applications

A particularly important nonlinear inverse problem is the *phase retrieval* problem, which has been extensively studied over the last decades because it arises in a lot of scientific and industrial applications, e.g. crystallography (Harrison, 1993)(Millane, 1990), astronomy (Fienup and Dainty, 1987) and (optical) imaging (Shechtman et al., 2014). It was originally formulated as the problem of reconstructing a signal from the magnitude of its Fourier transform (which arises when imaging a scene by measuring the magnitude of the scene's reflection in the far field when illuminated with a coherent electromagnetic field). Because electronic detectors cannot directly measure phase, we utilize algorithms to recover the phase based on prior knowledge of the signal domain (Shechtman et al., 2014).

As a practical example, let us consider the problem of reconstructing a real-world image $\mathbf{x} \in \mathbb{R}^n$ when only the magnitude of its discrete Fourier transform $|\mathcal{F}\mathbf{x}| =: \mathbf{y}$ can be measured. Obviously, the discrete Fourier transform $\mathcal{F}$ transforms the original real signal $\mathbf{x}$ into a complex signal $\mathcal{F}\mathbf{x}$, and the modulus operator $|\cdot|$ discards the phase information that is crucial in order to directly reconstruct $\mathbf{x}$ from $\mathbf{y}$. Reconstructing the missing phase information is equal to reconstructing the original signal (because we can then simply invert the problem using the inverse discrete Fourier transform $\mathcal{F}^{-1}$). Figure 1.3 visualizes the problem.

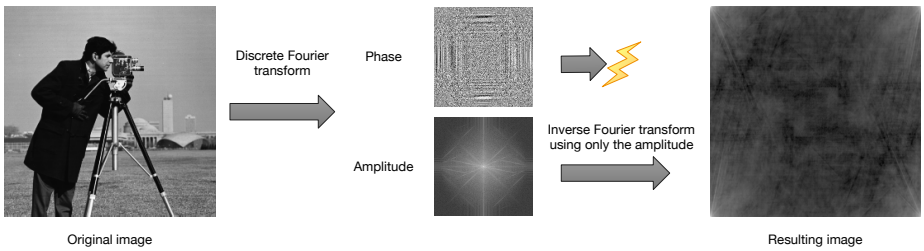


Figure 1.3: Schematic visualization of the discrete Fourier phase retrieval problem for an image $\mathbf{x} \in \mathbb{R}^n$. It is easy to see that given only $|\mathcal{F}\mathbf{x}|$, we cannot directly reconstruct $\mathbf{x}$ by applying the inverse discrete Fourier transform.

More generically, the aforementioned problem is also similar to reconstructing a signal from intensity-measurements of its diffraction pattern. Diffraction occurs when a wave hits an obstacle (whose size is in the order of the wavelength) or passes an aperture, and is an effect that drastically influences its wave pattern. The result is a complex-valued signal. However, as mentioned before, most detectors can only measure intensity, which motivates the usage of phase retrieval techniques to recover the original signal. An example of such a system is the *single-pixel imaging device* (in other literature also called *single-pixel camera*), where a target scene is illuminated with radiation that has passed a *spatial light modulator* configured with a random *on/off* pixel pattern, (where only *on*-pixels allow the radiation to pass through the modulator). The modulated radiation pattern then hits the scene and its transmission is collected through a collecting optics (e.g. a lens) at a single detector cell (hence the name *single-pixel camera*). The illumination is repeated with different *on/off* pixel patterns. The so collected information can be used to computationally recover an image of the original scene (i.e., *computational imaging*). The (noise-free) signal model for this scenario can be written as

$$y_i = \left| \sum_{j=1}^{n} \left(D_{\mathrm{S}\to\mathrm{D}} \mathrm{diag}(\mathbf{x}) D_{\mathrm{M}\to\mathrm{S}} \mathbf{a}_i \right)_j \right|^2 \tag{1.4}$$

where $\mathbf{a}_i \in \{0,1\}^n$ are real-valued vectors that represent random *on/off* pixel patterns for the spatial light modulator and $D_{\mathrm{M}\to\mathrm{S}}$ and $D_{\mathrm{S}\to\mathrm{D}}$ are complex matrices representing the diffraction effects between the spatial light modulator and the target scene $\mathbf{x}$, and between the target scene and the detector, respectively. The system is schematically described in Figure 6.2. We will go into detail about the signal model (1.4) and will evaluate some of the reconstruction algorithms presented in this thesis for the single-pixel use case (including diffraction) in Chapter 6.

Generalized Phase Retrieval

Phase retrieval can be defined with respect to arbitrary transformations, which leads to the following definition:

Definition 1.1.1. We define

$$\text{find} \quad \mathbf{x} \in \mathbb{R}^n \quad \text{s.t.} \quad \mathbf{y} = |\mathbf{A}\mathbf{x}|^2 \in \mathbb{R}^m,\ \mathbf{A} \in \mathbb{C}^{m \times n}$$

as the *(real-valued) generalized phase retrieval problem.*

In Definition 1.1.1 we refer to $\mathbf{y}$ as the *measured signal*. Additionally, we will call $\mathbf{A}$ the *sensing matrix* and its rows $\mathbf{a}_i$ the *measurement vectors*.

For the rest of this thesis, we will discuss how to solve the *generalized phase retrieval problem* for real-valued signals $\mathbf{x}$ under *random* measurement matrices $\mathbf{A} \in \mathbb{C}^{m \times n}$.

Prior Art

Alternating projections

Classical approaches to solve the *generalized phase retrieval* problem are methods based on *alternating projections*. One of the most famous algorithms that uses this approach is the *Fienup algorithm* (Fienup, 1982), which takes its idea from non-generalized phase retrieval, specifically the problem of recovering a complex or real image from magnitude Fourier measurements. The algorithm iteratively imposes real-plane and Fourier-plane constraints and is not guaranteed to recover the correct solution (Osherovich, 2012). The *Fienup* algorithm will be discussed in more detail in Section 4.2.

Semidefinite programming

Prior work also includes reconstruction algorithms based on *semidefinite relaxation*. One prominent example is *PhaseLift* (Candes et al., 2011), with the general idea being that Definition 1.1.1 can also be understood as a set of quadratic equations, which are equivalent to linear equations in higher dimensions. This yields a convex optimization problem that can be solved using semidefinite programming techniques and for which many guarantees exist (some of which will be explained later). Section 4.4.1 gives a more detailed discussion of this approach.

Sparsity-based methods

Another line of work investigates sparsity-based algorithms, which are incorporating prior knowledge of the signal into the (generalized) phase retrieval problem. The idea comes from the fact that many signals can be (approximately) represented as sparse vectors in *some* domain. For example, Figure 1.4 shows that *natural* images exhibit approximate sparsity in the wavelet domain. Therefore, the original signal $\mathbf{x}$ can be rewritten with a sparsity basis $\boldsymbol{\Psi}$ and a sparse vector $\boldsymbol{\alpha}$ as:

$$\mathbf{x} = \boldsymbol{\Psi}\boldsymbol{\alpha} \tag{1.5}$$

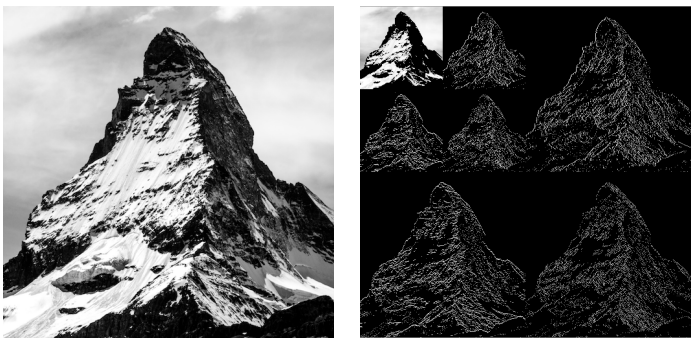


Figure 1.4: Natural image and its representation in the wavelet domain. Black pixels are coefficients close to zero, the signal in wavelet representation is therefore approximately sparse.

In this case a reconstruction algorithm can limit its search for the solution to the set of sparse vectors $\boldsymbol{\alpha}$. A popular sparsity-based algorithm is *GESPAR* (Shechtman et al., 2013), which will be addressed again in Section 4.4.2, because the idea of incorporating prior knowledge into the phase retrieval problem will be exploited heavily in the methods proposed later in this thesis.

Gradient-based methods

In 2014, Candes et al. introduced a gradient descent reconstruction scheme called *Wirtinger Flow* (Candes et al., 2015) that solves the generalized phase retrieval problem by minimizing a non-convex intensity loss function

$$f(\mathbf{x}) := \frac{1}{2m}\sum_{r=1}^{m}(y_r - |\langle \mathbf{a}_i, \mathbf{x}\rangle|^2)^2. \tag{1.6}$$

This method was extended to use *truncated generalized gradients* (Chen and Candes, 2015) to ensure that the gradients do not become too large, therefore converging faster to the optimal value.

In 2016, Wang et al. proposed *Truncated Amplitude Flow* (Wang et al., 2016), another gradient descent-based reconstruction method which minimizes the *amplitude loss*

$$h(\mathbf{x}) := \frac{1}{2m}\sum_{r=1}^{m}(\sqrt{y_r} - |\langle \mathbf{a}_i, \mathbf{x}\rangle|)^2. \tag{1.7}$$

Due to the non-convexity of the loss functions in all gradient descent reconstruction schemes, careful *initialization* is proposed for all methods, some of which will be described in more detail in Section 4.4.2.

Randomized Kaczmarz method

Another method, introduced by Tan and Vershynin in 2017, uses the observation that each real-valued measurement

$$\sqrt{y_i} = |\langle \mathbf{a}_i, \mathbf{x} \rangle| \tag{1.8}$$

in fact defines two *hyperplanes*, one corresponding to $\mathbf{x}$ and one to $-\mathbf{x}$. The authors therefore suggest to solve the generalized phase retrieval problem by iteratively choosing one of the measurements at random and projecting a running approximation $\mathbf{x}^{(k)}$ onto the closer of the two hyperplanes defined by that measurement (Tan and Vershynin, 2017). We will discuss this algorithm in more detail in Section 4.4.3.

1.2 Deep Generative Models

Machine learning has been at the core of a multitude of innovations in the last decade, and it is a powerful tool when it comes to modelling for decision making where the enumeration of all necessary rules to make the decision is impractical or impossible. Instead of writing decision rules in code, machine learning allows to learn these rules from data (more specifically in *supervised machine learning*, it learns these rules from the labeled *training set*, which contains both examples for data as well as their accompanying decisions) in such a general way that they can be used on formerly unseen data as well.

Deep learning as a specific subset of machine learning based on neural networks has been a very popular field of research in the last years. A particularly prominent example is Krizhevsky et al.'s paper *ImageNet Classification with Deep Convolutional Neural Networks* (Krizhevsky et al., 2012), in which the authors empirically showed the superior performance of *convolutional* networks over traditional machine learning methods for major image classification tasks.

Deep learning is able to model complex *nonlinear* relationships in data (such as recognizing shapes and faces in images) which traditional machine learning techniques are unable to model well. This makes them methods of choice when working with high-dimensional data (e.g. images or audio).

Depending on what a machine learning technique learns from the data, we can roughly distinguish *discriminative* and *generative* learning techniques. While *discriminative* techniques learn the *conditional probability* $p(y|x)$ of an output y (e.g. a decision) given its input x (e.g. an image), *generative* techniques aim to learn a *joint probability* $p(x, y)$ of input and output. The so learned *generative model* can then be used to generate arbitrary pairs of input and output by sampling from $p(x, y)$ (Jebara, 2004).

In this thesis we will be interested in such *generative models*, and particularly in *deep* generative models, which are neural networks trained to work as generators. An example of such a generative model is the generator part of a *variational autoencoder*, which is schematically depicted in Figure 1.5.

1.3 Solving Nonlinear Inverse Problems with Deep Learning

Due to their ability to approximate arbitrary continuous functions defined on compact subsets of $\mathbb{R}^n$ (Goodfellow et al., 2016, p.192), neural networks are of specific interest for phase retrieval problems.

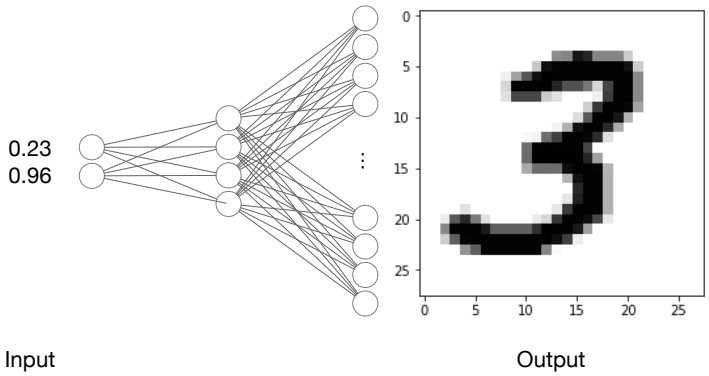


Figure 1.5: Generator part of a variational autoencoder produces images of handwritten digits out of $\mathbb{R}^2$ inputs.

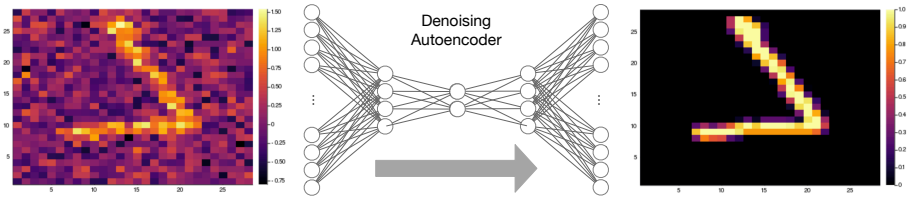


Figure 1.6: An autoencoder acting as a denoiser for the MNIST (LeCun, 1998) dataset.

For example, neural networks can be trained to *denoise* signals (with the *denoising autoencoder* (Goodfellow et al., 2016, p. 501f.) being a prominent example) while *generative neural networks* are trained on a huge amount of signals from some domain to learn the *signals' characteristics*, which in turn enables them to generate signals similar to (or, in the best case, indistinguishable from) the ones from the original signal domain. Both of these network types are of interest for the phase retrieval problem.

Not surprisingly, a lot of prior research has been conducted at the intersection of deep learning and nonlinear inverse problems, which can be roughly distinguished into two different classes: *deep regularization* and *deep generative priors*.

Deep Regularization

The idea behind *deep regularization* is that neural networks trained as *denoisers* can improve reconstruction when they are used as regularizers.

A denoiser is a function D that maps a noise-corrupted signal $\tilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\epsilon}$ to another signal $\hat{\mathbf{x}} = D(\tilde{\mathbf{x}})$ as similar to the original $\mathbf{x}$ as possible. Obviously from this definition, denoisers are dependent on the class of permissible signals. Most signal classes will be extremely large, which motivates the usage of machine learning techniques for the task of learning them. In fact, *denoising autoencoders* can learn to approximately denoise signals of such classes. An example of a denoiser based on an autoencoder is given in Figure 1.6.

prDeep (Metzler et al., 2018) is an algorithm that uses the *regularization by denoising (RED)* approach (Romano et al., 2017) to minimize the amplitude-based objective function of generalized phase retrieval by adding a generative neural network-based denoiser regularization term:

$$\min_{\mathbf{x}} \frac{1}{2}\|\sqrt{\mathbf{y}} - |\mathbf{A}\mathbf{x}|\|^2 + \lambda R(\mathbf{x}) \tag{1.9}$$

with $R(\mathbf{x}) = \mathbf{x}^\top(\mathbf{x} - D(\mathbf{x}))$ being a regularizer and $D(\mathbf{x})$ a denoiser based on a generative neural network trained on the signal domain (in the paper, they used the well-known *DnCNN* (Zhang et al., 2017) network). The regularizer acts as a penalty term for an $\mathbf{x}$ that shows a large difference between its denoised version and itself, and for correlations between $\mathbf{x}$ and $(\mathbf{x} - D(\mathbf{x}))$ (which effectively prevents the removal of structure from $\mathbf{x}$). The resulting optimization problem can be solved by proximal gradient methods, such as the *forward-backward splitting algorithm* (Parikh et al., 2014)(Goldstein et al., 2014).

Deep Generative Priors

Based on the idea that prior knowledge about the signal domain can be incorporated into the phase retrieval reconstruction process (as it was the case in the sparsity-based methods introduced in Section 1.1 and later to be detailed in Section 4.4.2), some methods have already been proposed which assume that any valid signal $\mathbf{x}^+$ is in the range of a generator function $G : \mathbb{R}^k \to \mathbb{R}^n$, $k < n$, so that $G(\mathbf{z}^+) = \mathbf{x}^+$ for some $\mathbf{z}^+$. Therefore, a reconstruction approach may effectively have the form

$$\text{find} \quad \mathbf{z} \quad \text{s.t.} \quad \mathbf{y} = |\mathbf{A}G(\mathbf{z})|^2. \tag{1.10}$$

In 2018, Shamshad and Ahmed as well as Hand et al. proposed to solve (1.10) by minimizing the amplitude loss function $\|\sqrt{\mathbf{y}} - |\mathbf{A}G(\mathbf{z})|\|^2$ with a simple gradient descent scheme in the domain $\mathcal{D}(G)$ of the generator G (Shamshad and Ahmed, 2018)(Hand et al., 2018). The *Deep Regularized Gradient Descent* method proposed in Section 5.1.1 is a regularized intensity loss-based extension of this algorithm.

In the remainder of this thesis we will not consider *deep regularization*, but will instead introduce a third class of algorithms which use the output of a deep generative prior-based reconstruction algorithm as an initializer for other reconstruction algorithms. We will call this class *deep generative initialization* and will discuss it in Chapter 5.2.

Chapter 2

Numerical Optimization

This chapter introduces the basics of numerical *unconstrained* optimization, which builds the foundation for the description of the reconstruction methods in Chapters 4 and 5.

Our general setup is that we want to find a vector $\mathbf{x}^* \in \mathbb{R}^n$ that minimizes a scalar function $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$:

$$\mathbf{x}^* := \underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}} f(\mathbf{x}). \tag{2.1}$$

If we restrict f to be continuously differentiable, then we can use *gradient descent* to find a *local* solution.

Definition 2.0.1. (Bertsekas, 2016, p.4) We define a vector $\mathbf{x}^*$ as an *unconstrained local minimum* of the function f if there exists an $\epsilon > 0$ such that

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad \forall \mathbf{x} \text{ with } \|\mathbf{x}^* - \mathbf{x}\| < \epsilon.$$

Definition 2.0.2. (Bertsekas, 2016, p.688) A set $C \subset \mathbb{R}^n$ is called *convex* if

$$\alpha\mathbf{x} + (1-\alpha)\mathbf{y} \in C \quad \forall\, \mathbf{x}, \mathbf{y} \in C,\ \forall\, \alpha \in [0,1].$$

A function $f : C \to \mathbb{R}$ defined on a convex set C is called *convex* if

$$f(\alpha\mathbf{x} + (1-\alpha)\mathbf{y}) \leq \alpha f(\mathbf{x}) + (1-\alpha) f(\mathbf{y}), \forall\, \mathbf{x}, \mathbf{y} \in C,\ \forall\, \alpha \in [0,1].$$

If we restrict f even more to be *convex*, then (2.1) has a global solution which can be found by a gradient descent algorithm, because any local minimum of f is also a global minimum.

Definition 2.0.3. (Bertsekas, 2016, p.4) Analog to Definition 2.0.1 we define a vector $\mathbf{x}^*$ as an *unconstrained global minimum* of the function f if

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad \forall \mathbf{x} \in \mathbb{R}^n.$$

However, because the phase retrieval objective function

$$f(\mathbf{x}) = \|\mathbf{y} - |\mathbf{A}\mathbf{x}|^2\| \tag{2.2}$$

is *non-convex* and *not continuously differentiable*, one cannot directly apply *gradient descent*, and even if f *would be* continuously differentiable, one cannot guarantee that the global optimum can be found in that way. We will see in the following section how gradient-based algorithms can nevertheless be used to solve (2.2).

2.1 Gradient Descent

One of the most well-known local optimization methods is *gradient descent*. The idea is relatively simple: To minimize a differentiable function $f(\mathbf{x})$, given a starting position $\mathbf{x}^{(0)}$, one descends in some direction $\mathbf{x}^{(1)} - \mathbf{x}^{(0)}$ so that $f(\mathbf{x}^{(1)}) \leq f(\mathbf{x}^{(0)})$. One proceeds like this until one can no longer find a direction in which to descend. According to Definition 2.0.1 one has found a *local minimum* of f.

Definition 2.1.1. The *gradient* of a differentiable scalar field $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ is

$$\nabla f := [\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_n}]^\top$$

i.e. the vector of all partial derivatives $\frac{\partial}{\partial x_i} f$ (Bertsekas, 2016, p. 766)(Goodfellow et al., 2016, p. 82).

The gradient is the direction of *steepest ascent* (Bertsekas, 2016, p.25).

A natural descent direction for the k-th iteration of this strategy is given by the negative gradient $-\nabla f(\mathbf{x}^{(k)})$, which is exactly the direction of steepest descent per Definition 2.1.1. Figure 2.1 shows on a contour plot how gradient descent iteratively reaches the minimum of some (convex) function f.

Mathematically, we can therefore write *gradient descent* as an iterative equation:

$$\mathbf{x}^{(k+1)} \quad := \quad \mathbf{x}^{(k)} - \alpha^{(k)} D^{(k)} \nabla f(\mathbf{x}^{(k)}), \tag{2.3}$$

where $D^{(k)}$ is a positive definite symmetric matrix (for simplicity we set $D^{(k)} = I$, which is sometimes referred to as the *method of steepest descent*) and $\alpha^{(k)}$ is the *step size* at iteration k (Bertsekas, 2016, p.25).

As it only takes into account the gradient of f, (2.3) is also commonly referred to as *first-order gradient descent*. The method terminates when the

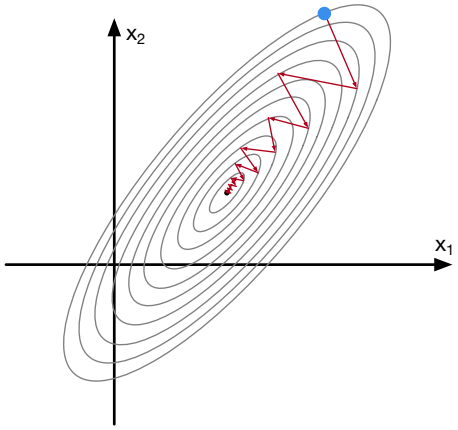


Figure 2.1: Gradient descent (starting at the solid blue dot) reaching the minimum of a convex function $f : \mathbb{R}^2 \to \mathbb{R}$ by iteratively descending in the direction of the negative gradient $-\nabla f$

gradient $\nabla f(\mathbf{x}^{(k)})$ vanishes, which means that, per Definitions 2.0.1 and 2.1.2, $\mathbf{x}^{(k)}$ is either a local minimum (in case the conditions from both definitions are fulfilled) or a saddle point (in case the condition from Definition 2.0.1 is not fulfilled). However, it can be shown that in many practical applications convergence to a saddle point is extremely unlikely and practically does not happen in higher dimensions (Lee et al., 2016), so that one can assume that (2.3) converges to a local minimum.

Definition 2.1.2. (Bertsekas, 2016, p.13) If $\mathbf{x}^*$ is an unconstrained local minimum of a function $f : \mathbb{R}^n \to \mathbb{R}$ and f is continuously differentiable in an open set S containing $\mathbf{x}^*$, then

$$\nabla f(\mathbf{x}^*) = 0.$$

Initialization

The initial value $\mathbf{x}^{(0)}$ is often chosen as a random vector. However, in the case of a non-convex function f, setting $\mathbf{x}^{(0)}$ close to the desired $\mathbf{x}^*$ (i.e., having a good initial guess) can lead to the local minimum found by gradient descent actually being the *global minimum*. This is the reason why gradient descent schemes can be used for global non-convex minimization when proper initialization values can be determined.

We will see later in this thesis that initialization plays a crucial role in the reconstruction quality of phase retrieval algorithms. Figure 2.2 shows an example of gradient descent finding the global minimum due to good initialization.

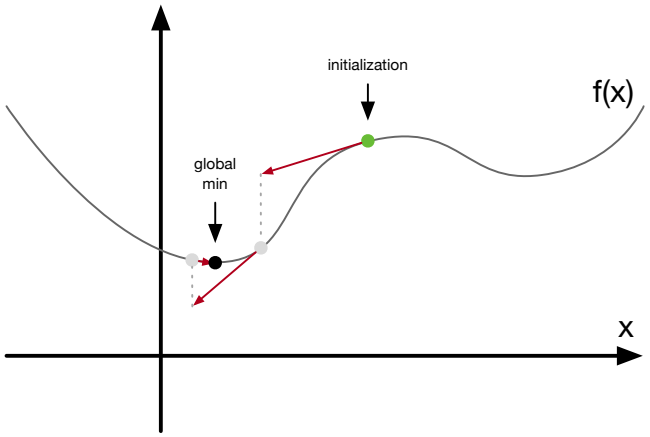


Figure 2.2: Gradient descent finding the global minimum of a nonconvex function due to good initialization.

Step size

Further importance shall be given to the *step size* $\alpha^{(k)}$ of (2.3). The most simple step size is a fixed

$$\alpha^{(k)} = \alpha, \quad k = 1, 2, ... \tag{2.4}$$

If the step size $\alpha^{(k)}$ is too large, $f(\mathbf{x}^{(k+1)}) \leq f(\mathbf{x}^{(k)})$ might no longer hold and divergence will occur; if the step size is chosen too small, convergence may be very slow. An appropriate constant step size must therefore be empirically determined (Bertsekas, 2016, p.33).

A variety of other step size rules exist, but their coverage is out of scope of this thesis.

Subgradients

In many cases our function f will *not* be continuously differentiable. If we take for example the point-wise modulus/absolute operator $f : \mathbb{R}^n \to \mathbb{R}^n : f(\mathbf{x}) = |\mathbf{x}|$, we can easily see that the gradient of f at $\mathbf{x} = \mathbf{0}$ is not defined. In order to use descent methods for this function, we can however replace the gradient with a *subgradient* of f at $\mathbf{x} = \mathbf{0}$ (Shor, 1985, p.3 ,p.22) (Boyd and Vandenberghe, 2004, p.338).

Definition 2.1.3. (Shor, 1985, p.9)(Bertsekas, 2016, p.731) For a *convex* function $f : \mathbb{R}^n \to \mathbb{R}$ and $\mathbf{x}_0 \in \mathbb{R}^n$, a vector $g_f(\mathbf{x}_0)$ is a *subgradient* of f at point $\mathbf{x}_0$ if it satisfies

$$f(\mathbf{x}) \geq f(\mathbf{x}_0) + (\mathbf{x} - \mathbf{x}_0)^\top g_f(\mathbf{x}_0)$$

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

The set of all subgradients of a function f at $\mathbf{x}_0$ is called *subdifferential* $\partial f(\mathbf{x}_0)$.

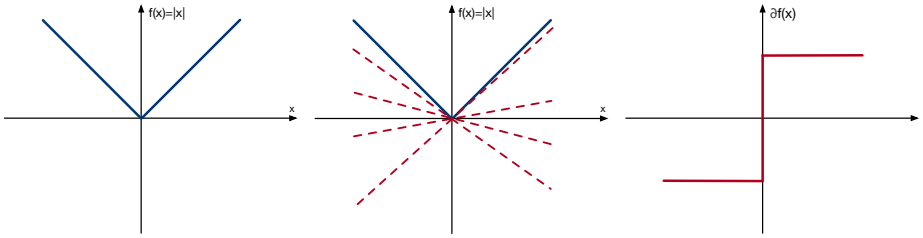


Figure 2.3: The function $f : \mathbb{R} \to \mathbb{R} : f(x) = |x|$ (blue solid line; left) with some exemplary subgradients (red dashed lines; center) as well as its subdifferential $\partial f(x)$ (red solid line; right).

This approach is also often called *subgradient method*, and it can be iteratively defined as

$$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} - \alpha^{(k)} g_f(\mathbf{x}^{(k)}), \tag{2.5}$$

where $g_f(\mathbf{x}^{(k)}) \in \partial f(\mathbf{x}^{(k)})$ is a particular subgradient (Shor, 1985, p.22) at $\mathbf{x}^{(k)}$.

To bring a visual example, Figure 2.3 shows the function $f : \mathbb{R} \to \mathbb{R} : f(x) = |x|$, some exemplary subgradients and its subdifferential.

An important observation is that subgradients are not necessarily descent directions. Therefore one commonly keeps track of the best iterate $\min_{i=1,..,k} f(\mathbf{x}^{(i)})$ (Boyd, 2014, p.4).

The subgradient method is identical to *gradient descent* for continuously differentiable functions because the subdifferential ∂f of a function f is identical to the gradient ∇f at all points $\mathbf{x}$ where $\nabla f(\mathbf{x})$ is defined.

Constant step sizes should theoretically not work well with subgradient-based descent methods because the function f might not be differentiable at its minimum and the series $\{g_f(\mathbf{x}^{(k)})\}_{k=1}^{\infty}$ might not converge to $\mathbf{0}$ (Shor, 1985, p.22). However, making use of subgradients is an intuitive way to use gradient descent for functions that are *almost everywhere* differentiable and has proven to work well in practice.

Chapter 3

Deep Learning

Deep learning is the collective term for a variety of concepts and algorithms that are based on the idea that one can approximate arbitrary continuous functions defined on compact subsets of $\mathbb{R}^n$ using a feedforward network of simple computational units with nonlinear activation functions (Goodfellow et al., 2016, p.192).[1]

Feedforward neural networks (also sometimes referred to as *multilayer perceptrons (MLPs)* or *deep feedforward networks*) are the most basic deep learning models. In the same way as with other *machine learning* methods, given a function f, we can train a feedforward network so that it learns to approximate f over any compact subset in $\mathcal{D}(f)$ (the domain of f). During training, we feed the network with input and output pairs, resulting in changes of the configuration of the network-internal parameters. We refer to this process as *learning* (Goodfellow et al., 2016, p.163).

This ability to learn a function makes machine learning and especially feedforward neural networks suitable for a multitude of tasks from *regression* (predicting the state of a system in a given configuration) to *classification* (mapping an input to a certain class or deciding if a certain decision should be taken) or *generation* (generating new data based on the internal characteristics of the data that the network has been trained on), where manually writing down rules would be inappropriate. These tasks can also be combined or extended to perform higher-level tasks, such as *machine translation*, *anomaly detection*, *missing value imputation* or *density estimation* (Goodfellow et al., 2016, p.98ff).

This chapter will give an overview of machine learning in general before defining feedforward neural networks and the special variety of *variational autoencoders* in Sections 3.2 and 3.6.

[1] A stricter version of this is the *universal approximation theorem*, which states that "a feedforward network with a linear output layer and at least one hidden layer with any *squashing* activation function [...] can approximate any Borel measurable function from one finite-dimensional space to another with any desired nonzero amount of error, provided that the network is given enough hidden units." (Goodfellow et al., 2016, p.192). However, this does not automatically mean that one will also be able to train the neural network to *learn* any such function (Goodfellow et al., 2016, p.193).

3.1 Machine Learning Essentials

We can divide most machine learning techniques into *supervised* and *unsupervised* techniques. While *supervised machine learning techniques* learn a relationship between the features of data points $\mathbf{X}$ and accompanying labels $\mathbf{y}$, which allows them to predict labels for unseen data points, *unsupervised machine learning algorithms* are trained only on the data points (without labels) with the aim to learn useful properties from them (Goodfellow et al., 2016, p.95, p.102). Which properties are *useful* though is highly dependent on the *task* at hand: while for an anomaly detection algorithm the distance of a data point to the rest of the data points or any other measure of similarity might be highly useful, another task might demand the machine to learn the general shape or structure of an object with the goal to learn how to come up with new object shapes by itself.

Consider the example of trying to learn to predict the function values of an arbitrary continuous scalar function $f : \mathbb{R}^n \to \mathbb{R}$, $f(\mathbf{x}) := y$ from k pairs of inputs and *noisy* outputs $(\mathbf{x}_k, y_k + \epsilon_k)$ where ϵ_k can for simplicity assumed to be zero-mean Gaussian noise $\epsilon_k \sim \mathcal{N}(0, \sigma)$. We call the individual entries of $\mathcal{D}(f)$ features and the ys *targets*. This is a classic *regression analysis* setup. Based on the learning task and the properties of f there are a variety of methods to address this problem.

Let us assume that our task is to find the best possible linear approximation of the data in a *least squares* sense and furthermore that we want to enforce the resulting regression to depend on as few of the original features as possible (which might be important to ensure good predictive power). This task can be written as an optimization problem as follows

$$\boldsymbol{\beta}^* := \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \frac{1}{n} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1 \tag{3.1}$$

where $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_k]^\top$ is the data (or *covariate*) matrix, $\mathbf{y}$ is the vector of targets (or *outcomes*) and $\boldsymbol{\beta} \in \mathbb{R}^n$ is the vector of feature weights of the linear regression model. This problem is often called *LASSO* (*least absolute selection and shrinkage operator*) (Hastie et al., 2009) and is a commonly used technique in machine learning and many other disciplines. Its objective function is non-linear, not everywhere differentiable (due to the presence of the *regularization term* $\|\beta\|_1$) and has no closed-form solution (Hastie et al., 2009, p.68).

Despite not having a closed form solution, (3.1) is convex (because both the least-squares operator and the ℓ^1-norm are convex) and can therefore be solved using the *subgradient method* introduced in Section 2.1. The resulting $\boldsymbol{\beta}^*$ is the vector with the weights for the individual features, so we can use it to compute a prediction $\hat{y} = \boldsymbol{\beta}^{*\top}\mathbf{x}$.

Empirical risk minimization

As we have seen, machine learning can be understood in terms of an optimization problem. Let us consider again a function $f : X \to Y$ (and call this function *hypothesis*) which we want to learn from data X and labels Y. We will also assume that there exists a joint probability distribution $p(X, Y)$ and that our dataset consists of k samples drawn *i.i.d.* from this distribution $p(X, Y)$.

We want to fit a linear function f to our available data pairs ($\mathbf{x}_i \in X, y_i \in Y$) in a *least squares* sense which, if written down for each data point, gives us a quadratic *loss function*

$$L(\hat{y}, y) = (\hat{y} - y)^2 = (f(\mathbf{x}) - y)^2. \tag{3.2}$$

A loss function is a non-negative scalar function that maps two labels to a value that indicates their difference, which in our case is the squared error $(\hat{y} - y)^2$. We call

$$R(f) = \mathbb{E}_{(\mathbf{x},y)\sim p(X,Y)} L(f(\mathbf{x}), y) \tag{3.3}$$

the *risk* associated with our hypothesis f.

The goal of machine learning is to find the best hypothesis f^* such that the *risk* is minimized, i.e.

$$f^* := \underset{f}{\operatorname{argmin}}\, R(f) \tag{3.4}$$

which, however, due to the joint probability $p(X, Y)$ being unknown can only be approximated using the *empirical* distribution $\hat{p}(X, Y)$, i.e. the set $(\mathbf{x}_i, y_i)$, $i = 1, ..., n$ of known data pairs. Therefore, we call

$$R_{\text{emp}}(f) = \mathbb{E}_{(\mathbf{x},y)\sim \hat{p}(X,Y)} L(f(\mathbf{x}), y) = \frac{1}{n} \sum_{i=1}^{n} L(f(\mathbf{x}_i), y_i) \tag{3.5}$$

the *empirical risk* and can thus rephrase the goal of a machine learning algorithm to solve the *empirical risk minimization*

$$\hat{f} := \underset{f}{\operatorname{argmin}}\, R_{\text{emp}}(f). \tag{3.6}$$

instead (Goodfellow et al., 2016, p.268f)(Vapnik, 1992).

3.2 Feedforward Neural Networks

Despite the vast applicability of traditional machine learning methods, there are some real-world challenges for which most of them have been shown not to work well. If one thinks of the central challenges in artificial intelligence, such as object or speech recognition, we have seen a specific class of machine learning methods show superior performance compared to traditional techniques. *Deep neural networks* have broken barriers of performance on many computer vision and machine understanding tasks, notably the already mentioned *ImageNet* on image classification (Krizhevsky et al., 2012) or the partly neural network-based *AlphaGo* program beating one of the world's best Go players in 2016 (Silver et al., 2017).

This section will deal with only one variety of deep neural networks, which are also the most basic ones: *fully-connected feedforward neural networks*. These networks consist of layers of many individual computational units called *neurons*, which are connected in a feedforward manner from the input layer towards the output layer, while every neuron in one layer is connected to every neuron in the next layer. The input layer has a number of neurons that is equal to the number of features of the data and the output layer is commonly modeled to have as many neurons as the label has dimensions. This can be a single neuron

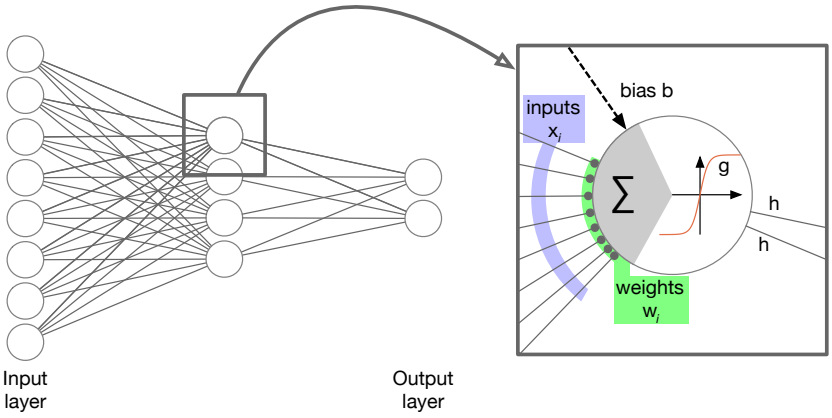


Figure 3.1: Exemplary architecture of a simple feedforward neural network and close-up of a single neuron. The neuron computes its output h by calculating the sum of the inputs x_i weighted by w_i (plus an additional bias b) and applying a nonlinear activation function g afterwards.

for a simple one-dimensional regression problem or k neurons for a classification problem with k classes, where the output of every neuron in the output layer can be understood as an indicator (e.g. probability) for class membership. Figure 3.1 shows an exemplary architecture of a simple feedforward neural network.

Every single neuron is defined by its inputs $\mathbf{x}$ (plus a separate *bias* b), input weights $\mathbf{w}$, a *nonlinear activation function* $g : \mathbb{R} \to \mathbb{R}$ and its output h (Duda et al., 2001, p.285). The neuron's output is given by

$$h = g(\mathbf{w}^\top \mathbf{x} + b). \tag{3.7}$$

The activation function g is usually chosen to be the same for all neurons of a single layer, so we can write the vectorized version of (3.7) as

$$\mathbf{h} = g(\mathbf{W}^\top \mathbf{x} + \mathbf{b}) \tag{3.8}$$

with $\mathbf{W}$ being a matrix with rows $\mathbf{w}_i$ as the input weights, $\mathbf{b}$ being the vector of all biases of the layer and g being applied point-wise (Goodfellow et al., 2016, p.187, p.191).

Popular activation functions in neural networks are the *sigmoid*, tanh or the *rectified linear unit*. The $\text{sigmoid}(x) : \mathbb{R} \to [0, 1]$ function (Goodfellow et al., 2016, p.65) is defined as

$$\text{sigmoid}(x) := \frac{1}{1 + \exp(-x)} \tag{3.9}$$

and is an activation function that is particularly often used in the output layer to scale the output to the range $[0, 1]$.

The rectified linear unit ($ReLU$) (Goodfellow et al., 2016, p.187)

$$\text{relu}(x) := \max\{0, x\} \tag{3.10}$$

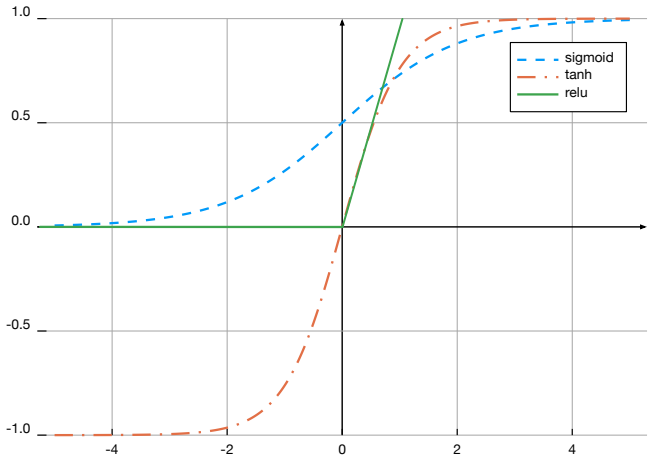


Figure 3.2: Plots of the *sigmoid*, tanh and the *rectified linear unit* activation functions commonly used in neural networks.

is another activation function which is especially fast to derive (although not directly differentiable in 0) and therefore despite its similarity very often used in deep neural networks. A very popular extension of the $ReLU$ is the *leaky* $ReLU$, which is defined[2] as

$$\text{leakyrelu}(x) := \max\{0.01x, x\} \tag{3.11}$$

Figure 3.2 plots the sigmoid, tanh and the relu activation functions.

All l layers in the network together form the neural network with input $\mathbf{x}$ and output $\hat{\mathbf{y}}$, which allows us to concisely write down the full network in vectorized form as

$$\hat{\mathbf{y}} := g^{(l)}(\mathbf{W}^{(l)\top} \cdots g^{(3)}(\mathbf{W}^{(3)\top} g^{(2)}(\mathbf{W}^{(2)\top}\mathbf{x} + \mathbf{b}^{(2)}) + \mathbf{b}^{(3)}) \cdots + \mathbf{b}^{(l)}). \tag{3.12}$$

Note that the input layer (technically layer 1 here) does not appear in the equation as it is usually only mentioned for a better understanding but not explicitly modeled.

Training a neural network

To fit an untrained neural network, we feed training data (pairs $(\mathbf{x}, y)$ of features $\mathbf{x}$ and targets y, which we will for simplicity assume to be scalar, although y can be a vector if the output layer of the network contains multiple neurons) to the input layer, pass it through the network and take the output $\hat{y}$ at the output layer to compare it to the target value y. A non-negative function of the difference between y and $\hat{y}$ is called *loss* and is a scalar function $L(\hat{y}, y)$ as explained in Section 3.1. Our aim is to minimize this loss, which means to match the network outputs $\hat{y}$ with the desired outputs y as closely as possible (Duda et al., 2001, p.294f).

[2] `https://github.com/FluxML/NNlib.jl/blob/52b9c39e0be6423afa1867838bf68472a87692dd/src/activation.jl`

We can define the *total loss* J with respect to the model parameters $\boldsymbol{\theta}$ (including weights $\mathbf{W}$ and model biases $\mathbf{b}$) as a function of the loss L with a potentially added regularization term $\Omega(\boldsymbol{\theta})$:

$$J(\mathbf{x}, y; \boldsymbol{\theta}) := L(f(\mathbf{x}; \boldsymbol{\theta}), y) + \Omega(\boldsymbol{\theta}). \tag{3.13}$$

Using this total loss J, we can update our network parameters $\boldsymbol{\theta}$ for every training pair $(\mathbf{x}, y)$ by performing a gradient descent step using the gradients with respect to biases and weights calculated using the well-known *backpropagation* algorithm described in Algorithm 1.

Algorithm 1: Backpropagation algorithm for training neural networks (cf. (Goodfellow et al., 2016, p.206))

Data: input $\mathbf{x}$
target y
the model's weight matrices $\mathbf{W}^{(k)}$ for all layers $k = 1, ..., l$
the activations at all layers $k = 1, ..., l$: $\mathbf{a}^{(k)} := \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}$
$\mathbf{h}^{(k)} := g^{(k)}(\mathbf{a}^{(k)})$ for all layers k

Result: gradients $\nabla_{\mathbf{b}^{(k)}} J$ and $\nabla_{\mathbf{W}^{(k)}} J$ for all layers k to be used in a gradient descent update step

After the forward pass (3.12) through the network has been computed, compute the gradient on the output layer: $\boldsymbol{\gamma} \leftarrow \nabla_{\hat{y}} J := \nabla_{\hat{y}} L(\hat{y}, y)$

for $k = l, l-1, ..., 1$ **do**

Propagate output layer's gradients into the nonlinear activations:
$\boldsymbol{\gamma} \leftarrow \nabla_{\mathbf{a}^{(k)}} J := \boldsymbol{\gamma} \odot \frac{\partial}{\partial \mathbf{a}^{(k)}} g^{(k)}(\mathbf{a}^{(k)})$

Now compute gradients on weights $\mathbf{W}^{(k)}$ and biases $\mathbf{b}^{(k)}$:
$\nabla_{\mathbf{b}^{(k)}} J := \boldsymbol{\gamma} + \lambda \nabla_{\mathbf{b}^{(k)}} \Omega(\boldsymbol{\theta})$
$\nabla_{\mathbf{W}^{(k)}} J := \boldsymbol{\gamma}\mathbf{h}^{(k-1)\top} + \lambda \nabla_{\mathbf{W}^{(k)}} \Omega(\boldsymbol{\theta})$

Now propagate gradients to the next-lower layer:
$\boldsymbol{\gamma} \leftarrow \nabla_{\mathbf{h}^{(k-1)}} J := \mathbf{W}^{(k)\top}\boldsymbol{\gamma}$

end

It is important to notice that while the method described in Algorithm 1 is technically correct (and most commonly known as *stochastic gradient descent*, because it performs a gradient descent step based on one (randomly chosen) $(\mathbf{x}, y)$ at a time), most practical applications usually make use of *minibatches*, where the gradient is calculated not with respect to only one $(\mathbf{x}, y)$ pair, but to a (usually small) number of them.

3.3 Algorithmic Differentiation

In order to execute Algorithm 1, we need to be able to solve for the gradients of J programmatically in a timely and exact manner. To do this, we have a number of options: we could *(1) manually differentiate* all necessary functions and implement them in code, we could *(2) numerically differentiate* all functions, we could *(3) symbolically differentiate* the complete algorithm (Goodfellow et al., 2016, p.206) or we could *(4) change our code on-the-fly* so that variables are

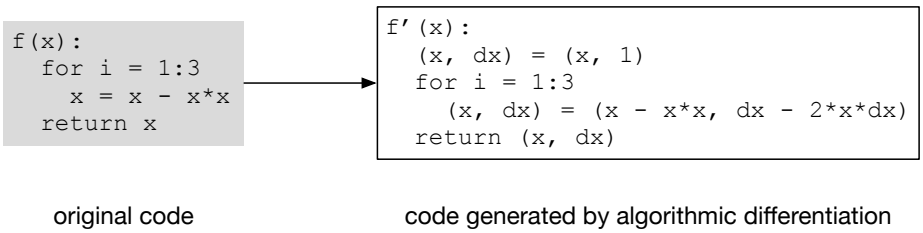


Figure 3.3: Pseudocode of a function and its derivative with respect to x created using automatic differentiation. Every occurrence of the variable `x` is replaced by a tuple consisting of a variable for `x` and one for its derivative `dx`, and every (basic) operation affecting `x` is replaced by a tuple consisting of the operation itself and its derivative. Adapted from (Baydin et al., 2015).

saving additional information for their derivatives and all operators know their derivatives which they propagate per the chain rule of differential calculus (Baydin et al., 2015).

Manual differentiation *(1)* can get out of hand and is prone to error for very large functions. Numerical differentiation *(2)*, while easy to implement, can be very inexact due to rounding and truncation errors (Jerrell, 1997). Symbolic differentiation *(3)* requires the functions to be defined as a closed-form mathematical expression, therefore making it unattractive given the potentially complex control flow of a program (Baydin et al., 2015).

Method *(4)* is also commonly called *automatic* or *algorithmic differentiation* and is at the core of many deep learning libraries (such as Python's *PyTorch*[3] or Julia's *Flux*[4]). It usually involves having explicit symbolic derivatives of all the programming language's basic operators (e.g. $+$, $*$, $/$) and standard functions (e.g. $\exp(x)$, $\text{abs}(x)$, $\max(x, y)$) and creating specialized code for the program to get derivatives or gradients for arbitrary inputs. Algorithmic differentiation does not suffer from truncation errors (Griewank and Walther, 2008, p.2) and is a low-overhead way to differentiate through arbitrarily complex computer programs, which includes, but is obviously not limited to, deep neural networks. In fact, automatic differentiation is used in many other cases outside of deep learning, and we will use it to compute gradients for the deep generative model-supported algorithms in Chapter 5.

Figure 3.3 shows in pseudocode how the derivative of a simple function is created using automatic differentiation.

As seen in the the list of standard functions in the last paragraph, some elemental functions or constructs of programming languages are inherently non-differentiable (such as $\text{abs}(x)$, $\max(x, y)$ or *if-else-statements*). If we can, however, ensure that a function $f : \mathcal{D} \to \mathbb{R}^n$ is *piecewise differentiable* on an open domain $\mathcal{D}$, which means that there exists a *selection* of functions $f^{(k)} : \mathcal{D}^{(k)} \to \mathbb{R}^n$ that are continuously differentiable on their open domains $\mathcal{D}^{(k)}$, then we can differentiate them at any point in $\bigcup_{k=1,\dots,l} \mathcal{D}^{(k)}$. As a result, we can (often) get meaningful replacements for gradients at points of non-differentiability (Griewank and Walther, 2008, p.335, p.342).

[3] `https://pytorch.org/`
[4] `https://fluxml.ai/`

As an example, the absolute operator $\text{abs}(x) : \mathbb{R} \to \mathbb{R}$ is piecewise differentiable, and we can write it down as

$$\text{abs}(x) = |x| = \begin{cases} x & =: \text{abs}^{(1)}(x) : \mathbb{R} \to \mathbb{R}, \quad \text{if } x > 0 \\ 0 & =: \text{abs}^{(2)}(x) : \mathbb{R} \to \{0\}, \quad \text{if } x = 0 \\ -x & =: \text{abs}^{(3)}(x) : \mathbb{R} \to \mathbb{R}, \quad \text{if } x < 0. \end{cases} \tag{3.14}$$

While we cannot directly derive $\text{abs}(x)$ at $x = 0$, we can instead derive $\text{abs}^{(2)}(x)$ at $x = 0$ because it is continuously differentiable.

As the topic of algorithmic differentiation is much more complex than introduced here and out of scope of this thesis, the interested reader may refer to Griewank and Walther for a deeper discussion on the topic (Griewank and Walther, 2008).

3.4 Convolutional Neural Networks

As this thesis deals with the reconstruction of images, we want to briefly introduce a certain kind of neural network that is specifically designed for grid-like data, such as images (2D) or time series (1D) data.

We call a neural network *convolutional* if it uses a convolution operator instead of the general matrix multiplication in at least one of its layers. Moreover, convolutional neural networks typically make use of *sparse interactions*, which results in *not* every neuron of layer l being connected to every neuron of the subsequent layer $l + 1$ anymore (in comparison to the fully-connected layers in regular networks). This is achieved by using convolution kernels which are smaller than the size of the image/grid at layer l (Goodfellow et al., 2016, p.321, p.325). Convolutional networks are trained in the same way as regular neural networks, but (instead of weights) the convolution kernels are learned.

Definition 3.4.1. (Goodfellow et al., 2016, p. 332) The convolution of a 2-dimensional image $\mathbf{M}$ with a 2-dimensional kernel $\mathbf{K}$ is defined as

$$(\mathbf{M} * \mathbf{K})_{i,j} := \sum_m \sum_n \mathbf{M}_{m,n} \mathbf{K}_{i-m,j-n}.$$

We can describe a *convolutional layer* as a sequence of two operations:

1. A *convolution operation* where the input data (i.e. the output from the previous layer) is convolved with a kernel $\mathbf{K}$.
2. An (optional) *detector operation*, in which a nonlinear activation function is applied point-wise to the output of the convolution stage.

Another important concept besides convolution is *pooling*, which is essentially a *summary* operator (for example the max function, then also commonly referred to as *maxpooling*) applied with a certain neighborhood size (just like the *regular* convolution). Pooling allows to learn e.g. invariances to spatial translations (by applying spatial pooling operations at one or multiple stages of

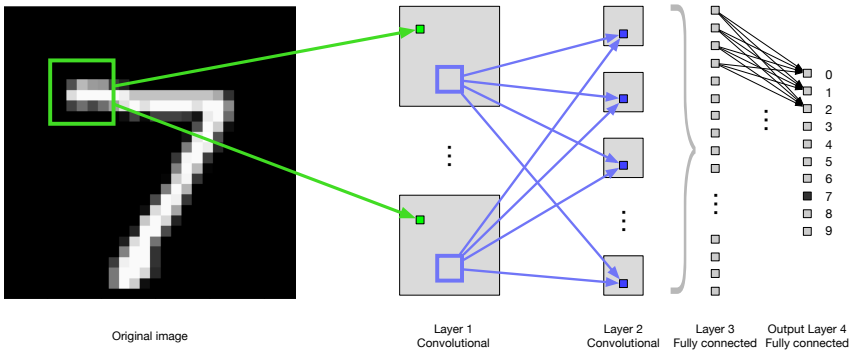


Figure 3.4: Schematic view of a convolutional neural network for the classification of handwritten digits of the MNIST dataset. At the first layer, the image is convolved with different masks, resulting in multiple new intermediate images, which are again convolved at the second layer with further masks. The resulting intermediate images are vectorized so that they can be consumed by the fully-connected third layer. The output layer consists of ten neurons, which serve as indicators for class membership. In this example, the network is assigning a high value to the output-layer neuron 7, indicating that the image has been classified as a 7. Figure based on (O'Neill, 2006).

the network). We call a layer that performs a pooling operation a *pooling layer* (Goodfellow et al., 2016, p.330).

Both pooling and convolutional layers can be parameterized by a *stride* width, which defines the number of pixels to skip in each direction between each convolution operation. This effectively downsamples the image.

Common convolutional neural networks make use of many convolutional and/or pooling layers and also combine them, depending on the task, with regular fully-connected layers. Figure 3.4 schematically shows a convolutional neural network used to classify handwritten digits.

The topic of convolutional neural networks is again much broader and only the basics needed for the understanding of the models used in the next chapters have been introduced here, so we refer to Goodfellow et al. (Goodfellow et al., 2016) for a deeper discussion on the topic.

3.5 Generative Machine Learning

A useful way to distinguish machine learning models is on the basis of what kind of probability distributions they learn from the data. In the example in Section 3.1 we have seen that LASSO learns to predict an outcome $y \in Y$ based on an input $\mathbf{x} \in X$. LASSO does not learn any information about the probability distribution $p(X, Y)$ that underlies the data and the labels, i.e., it is agnostic to the way how the data has been generated.

There are, however, other machine learning methods that are able to learn the joint probability distribution $p(X, Y)$ of data and labels. Those machine learning models are typically called *generative models*, and knowledge about

$p(X, Y)$ allows us to solve for and therefore also sample from $p(X)$. Sampling from $p(X)$ is equivalent to generating new (unseen!) data points from the source distribution of the data X. This property makes generative models particularly interesting for certain machine learning tasks (such as *density estimation*) and will serve us very well in the following sections.

One example for a generative machine learning technique is the *variational autoencoder*, which will be explained in the next section.

3.6 Variational Autoencoders

Before we come to the introduction of the *variational autoencoder*, we need to take a look at a certain type of networks, the *differentiable generator networks*. These are differentiable functions $g(\mathbf{z}; \boldsymbol{\theta})$ (where $\boldsymbol{\theta}$ represents the model parameters) which transform samples of a variable $\mathbf{z}$ to samples or to distributions of $\mathbf{x}$ parameterized by $\boldsymbol{\theta}$. The variables $\mathbf{z}$ are often drawn from a probability distribution (for example the normal distribution $\mathcal{N}(0, 1)$) themselves (Goodfellow et al., 2016, p.684f).

Let our goal be to generate samples from a very complicated distribution, which we assume to be the distribution $p(X)$ underlying our training data X. We will assume that based on our limited amount of samples, we cannot directly specify $p(X)$, so we use a feedforward network to learn an approximation $p_{\text{model}}(X)$ of this probability distribution $p(X)$.

A *variational autoencoder (VAE)* is a neural network that uses a differentiable generator network $g : Z \to X$ (called the *generator* or *decoder*) in combination with an inference network $e : X \to Z$ (called the *encoder*) to learn the probability distribution $p_{\text{model}}(X)$. These two networks are linked together in the following way: a training data point $\mathbf{x}$ gets fed into the encoder network which outputs a sample of the latent (*encoded*) representation $\mathbf{z} = e(\mathbf{x}) \sim q(\mathbf{z}|\mathbf{x})$. This $\mathbf{z}$ is fed into the generator which is able to translate it to an approximation $\hat{\mathbf{x}} = g(\mathbf{z}) \sim p_{\text{model}}(\mathbf{x}|\mathbf{z})$ of the original $\mathbf{x}$ (Goodfellow et al., 2016, p.687f).

A core idea of variational autoencoders is to *train the encoder together with the generator*: we train the encoder so that *(1)* $q(\mathbf{z}|\mathbf{x})$ follows a predefined probability distribution $p_{\text{model}}(\mathbf{z})$ (for example $\mathcal{N}(0, 1)$) and that *(2)* $\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})} p_{\text{model}}(\mathbf{x}|\mathbf{z})$ is maximized.

Definition 3.6.1. (Duda et al., 2001, p. 632) The *Kullback-Leibler divergence* is a measure of difference between two probability distributions $p(X)$ and $q(X)$ over the same random variable X and is defined as

$$D_{\text{KL}}(p\|q) := \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)}.$$

We can achieve *(1)* by minimizing the *Kullback-Leibler divergence* between $q(\mathbf{z}|\mathbf{x})$ and the predefined probability distribution $p_{\text{model}}(\mathbf{z})$. To achieve *(2)*, instead of maximizing $\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})} p_{\text{model}}(\mathbf{x}|\mathbf{z})$ directly, we will maximize $\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})} \log p_{\text{model}}(\mathbf{x}|\mathbf{z})$. Overall, we can rewrite the training objective as a loss function

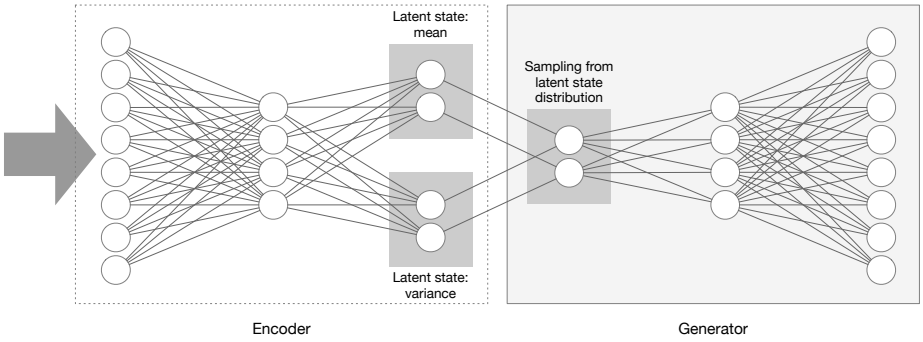


Figure 3.5: Schematic architecture of a variational autoencoder. The *encoder* part produces a latent representation which follows the two-dimensional Normal distribution (modeled explicitly by two neurons for the means of the latent representation and two neurons for the variances of the latent representation). The *decoder* part samples from the latent state distribution and generates the output by propagating the values through its network.

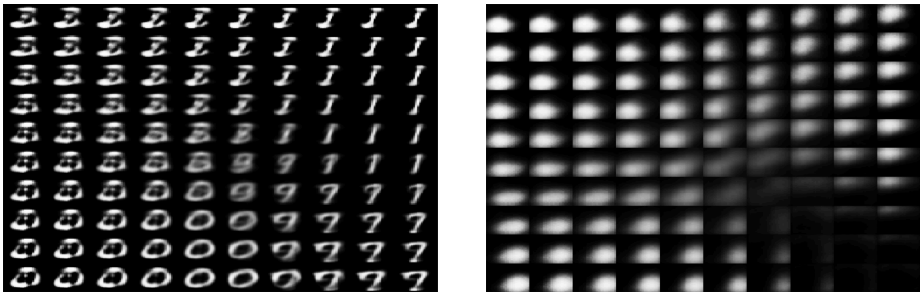


Figure 3.6: Exemplary output of the generator parts of two variational autoencoders trained on the MNIST and Shepp-Logan dataset when sweeping through parts of the two-dimensional latent distribution.

for a *maximization* problem as follows

$$L(q) = \text{ELBO}(q) = \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})} \log p_{\text{model}}(\mathbf{x}|\mathbf{z}) - D_{\text{KL}}\left(q(\mathbf{z}|\mathbf{x}) \| p_{\text{model}}(\mathbf{z})\right) \quad (3.15)$$

which is in other literature also known as the *evidence lower bound (ELBO)* (Goodfellow et al., 2016, p.687).

Learning now means maximizing L by adapting the parameters $\boldsymbol{\theta}$ of the encoder and generator networks, which we can do using backpropagation by sampling from the training data.

After the training, we can use the generator part of the variational autoencoder to produce new samples from the distribution $p_{\text{model}}(\mathbf{x}|\mathbf{z})$ after randomly sampling $\mathbf{z} \sim p_{\text{model}}(\mathbf{z})$ (in practice, $p_{\text{model}}(\mathbf{z})$ will often be Gaussian), by feeding $\mathbf{z}$ into the generator g.

Figure 3.5 shows an exemplary architecture of a variational autoencoder and highlights its most important parts. Additionally, Figure 3.6 shows the output

of the generator part of a variational autoencoder for inputs $\mathbf{z}$ sweeping through parts of a two-dimensional distribution $p_{\text{model}}(\mathbf{z})$.

Variational autoencoders can also be combined with other techniques such as convolutional neural networks. The fully-connected feedforward networks for generator and encoder networks are then replaced with convolutional neural networks.

Chapter 4

Inverse Problems

We will now take a look at a generalization of the initial problem statement from Chapter 1. Given some map $\mathcal{A} : \mathcal{D}(\mathcal{A}) \subset X \to Y$ (where X and Y are finite-dimensional normed spaces) and a *measurement* $y \in Y$, we are interested to recover the original signal $x \in X$:

$$\text{find} \quad x \quad \text{s.t.} \quad ||y - \mathcal{A}(x)||_Y \leq \delta \tag{4.1}$$

where $\delta \geq 0$ is a small constant to account for measurement errors.

We will call $\mathcal{A}$ from now on the *forward operator*. This is a very generic definition of an *inverse problem* as we do not impose any constraints on the map $\mathcal{A}$ (Mueller and Siltanen, 2012, p.140).

We call an inverse problem of the form (4.1) *well-posed* if all of the following *Hadamard conditions* are fulfilled (Mueller and Siltanen, 2012, p.139):

- There is a unique solution for every $y \in Y$;
- The solution is stable under perturbations of y, i.e. the operator $\mathcal{A}^{-1}$ is defined on all Y and is continuous.

For many interesting problems the Hadamard conditions will however *not* be fulfilled (such a problem is also called *ill-posed*), which means that we will not be able to find a *backward operator* $\mathcal{A}^{-1}$ which is applicable to all $y \in Y$. Under certain assumptions on the forward operator $\mathcal{A}$ and if we restrict our x to belong to a *permissible signal domain* $\mathcal{X} \subset X$, we might however have a chance to recover x from y.

In the next section, we will investigate how to solve (4.1) for certain *linear* forward operators $\mathcal{A}$, while Sections 4.2 and 4.3 will explore the problem for a certain class of *nonlinear* forward operators, which will result in the definition of the *generalized phase retrieval* problem. This problem is a fundamental one that occurs in many disciplines, and we will focus on how to solve it for the rest of this thesis.

4.1 Linear Inverse Problems

Before we start looking at *nonlinear* inverse problems, we first want to look at a technique for solving *linear* inverse problems of the form

$$\text{find} \quad \mathbf{x} \quad \text{s.t.} \quad \mathbf{y} = \mathbf{A}\mathbf{x}, \quad \mathbf{A} \in \mathbb{C}^{m \times n}, \, \mathbf{x} \in \mathbb{C}^n, \, \mathbf{y} \in \mathbb{C}^m. \tag{4.2}$$

In the case that $\mathbf{A}$ is invertible, all Hadamard conditions are fulfilled, and the solution to (4.2) is directly provided by

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{y}. \tag{4.3}$$

Definition 4.1.1. (Goodfellow et al., 2016, p. 45f) The Moore-Penrose pseudoinverse $\mathbf{A}^\dagger$ of a matrix $\mathbf{A}$ is defined as

$$\mathbf{A}^\dagger = \lim_{\alpha \to 0} (\mathbf{A}^\top \mathbf{A} + \alpha \mathbf{I})^{-1} \mathbf{A}^\top.$$

However, in most practical use cases, $\mathbf{A}$ will not be quadratic, and therefore will not be invertible. A natural approach, then, would be the usage of the *Moore-Penrose pseudoinverse* $\mathbf{A}^\dagger$ (cf. Definition 4.1.1), which yields the solution to the following least squares problem:

$$\mathbf{A}^\dagger \mathbf{y} := \operatorname*{argmin}_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{y}\|^2. \tag{4.4}$$

For some applications, this can deliver an appropriate approximation of the true value $\mathbf{x}$ (this is the simplest case of a *linear regression*). However, in many cases, the least squares solution will not be appropriate.

Besides solving (4.4) to obtain an approximation of $\mathbf{x}$, there exist a variety of other methods. The *LASSO* method has already been introduced in Section 3.1 and obtains an estimate of $\mathbf{x}$ by solving

$$\operatorname*{argmin}_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{y}\|^2 + \lambda \|\mathbf{x}\|_1, \tag{4.5}$$

which is basically selecting an $\mathbf{x}$ that is close to the least squares solution but also to some extent (characterized by the factor λ) minimizes the *regularization term*, which is the ℓ^1-norm of $\mathbf{x}$. The collective minimization of the *data fidelity* term $\|\mathbf{A}\mathbf{x} - \mathbf{y}\|^2$ and the regularization term $\|\lambda \mathbf{x}\|_1$ in (4.5) yields results which are often very desirable, because they promote the sparsity of $\mathbf{x}$. We will now explore a field of research on linear inverse problems called *compressed sensing* that is particularly interested in finding such sparse solutions.

4.1.1 Compressed Sensing

Compressed sensing looks at (4.2) from a slightly different angle. The relatively new field of research (which has been introduced by the seminal works of Donoho (Donoho, 2006) and Candes and Tao (Candes and Tao, 2006)) is interested in the reconstruction of signals $\mathbf{x}$ from *measurements* $\mathbf{y} = \mathbf{A}\mathbf{x}$ under the assumption that $\mathbf{x}$ is *k-sparse*, i.e. that $\mathbf{x}$ has only a maximum of k non-zero entries, or that $\mathbf{x}$ is at least *compressible*, i.e. that only very few of its entries are large and the others are very small in comparison.

Many natural signals exhibit this property if they are represented in another basis. The *wavelet basis* for example allows for a compressible representation of natural images, which was already shown in Figure 1.4. We can therefore also (approximately) express our true image $\mathbf{x}$ as a multiplication of a sparsity basis

and a sparse vector ($\mathbf{x} = \mathbf{\Psi}\boldsymbol{\alpha}$). This observation is the key insight that allows us to compress signals (e.g. JPEG for images) and recover them without major information loss.

Let us imagine that we want to take a photograph of a scene. A modern digital camera takes an image by saving the intensities of every pixel of its 2D light sensor array (i.e. we directly sample $\mathbf{x} = \mathbf{I}\mathbf{x}$). Afterwards, it compresses this data (to reduce the file size) and saves it to its data storage. One might ask the question: If we know that our signal (in this case, an image) is actually sparse (or compressible) in a known basis, then do we really need to sample all pixels individually? Can we not find a different way to sample so that we capture only the absolute minimum of information that is needed to recreate our true image?

Translated into our theoretical framework, the resulting question is therefore: Knowing that $\mathbf{x}$ is k-sparse, can we design a *measurement matrix* $\mathbf{A}$ in such a way that we can recover the true $\mathbf{x}$ from an *underdetermined system* $\mathbf{y} = \mathbf{A}\mathbf{x}$?

Research shows that explicitly designing such matrices optimally in terms of sampling rate is extremely hard, but that matrices based on some probability distributions are suitable for that purpose, for example Gaussian matrices ($\mathbf{A}_{i,j} \sim \mathcal{N}(0,1)$ or $\mathbf{A}_{i,j} \sim \mathcal{N}(0,1/2) + \mathrm{i}\mathcal{N}(0,1/2)$) (Baraniuk et al., 2008).[1]

This means that using a Gaussian measurement matrix $\mathbf{A}$, with overwhelming probability we are able to recover a sparse $\mathbf{x}$ given $\mathbf{y}$ from $\mathbf{y} = \mathbf{A}\mathbf{x}$.

Recovering $\mathbf{x}$ from $\mathbf{y} = \mathbf{A}\mathbf{x}$ is now no different than solving any other inverse problem, and one approach (of many!) is to apply the aforementioned LASSO, which, in the compressed sensing community, is sometimes also called *basis pursuit* (Chen et al., 1998).

4.2 Nonlinear Inverse Problems

We can as well aim to solve inverse problems in which the forward operator $\mathcal{A}$ is *nonlinear* in a least-squares sense by solving the minimization problem

$$\min_{\mathbf{x}} \|\mathbf{y} - \mathcal{A}(\mathbf{x})\|_2^2. \tag{4.6}$$

Assuming only differentiability of $\mathcal{A}$ and without knowledge of an inverse operator $\mathcal{A}^{-1}$, a generic approach to solve this problem is the usage of gradient descent methods (Mueller and Siltanen, 2012, ch.11). As in the linear case, we could add a regularization term (such as the ℓ^1-regularizer $\|\mathbf{x}\|_1$ or a regularizer based on discrete (anisotropic) total variation $\|\mathbf{x}\|_{\mathrm{TV}}$ (Condat, 2017) (see Definition 4.2.1)) to the objective and solve for x iteratively (Engl and Kügler, 2005). Since $\mathcal{A}$ is most probably not convex, those methods will get stuck in local minima though, so proper initialization is needed.

Definition 4.2.1. Condat (Condat, 2017) defines

$$||\mathbf{X}||_{\mathrm{TV}} := \sum_{i=1}^{n_1}\sum_{j=1}^{n_2} |\mathbf{X}_{i+1,j} - \mathbf{X}_{i,j}| + |\mathbf{X}_{i,j+1} - \mathbf{X}_{i,j}|$$

[1] For a deeper discussion on the properties that $\mathbf{A}$ needs to fulfill, see the works by Candes and Tao (Candes and Tao, 2005) and (Eldar and Kutyniok, 2012).

as the *discrete anisotropic total variation (norm)* of an image $\mathbf{X} \in \mathbb{R}^{n_1 \times n_2}$ (Chambolle et al., 2009).

Based on this, we say that

$$||\mathbf{x}||_{\text{TV}} := ||\mathbf{X}||_{\text{TV}}$$

is the total variation (norm) of the signal $\mathbf{x} \in \mathbb{R}^{n_1 \cdot n_2}$ if the vector $\mathbf{x}$ implicitly defines a matrix $\mathbf{X}$ (e.g. when representing a 2D image as a vector instead of as a matrix).

Note also that as in the linear case, for such an ill-posed problem without any further assumptions on $\mathcal{A}$, our chances might be low that we are actually able to recover the original signal $\mathbf{x}$ from the measurement $\mathbf{y}$: imagine the most extreme case of $\mathcal{A}(\mathbf{x}) \equiv \mathbf{0}$, where every $\mathbf{x}$ becomes a valid global minimum of (4.6). In that case, we obviously cannot recover any $\mathbf{x}$ as all information about the signal is lost in $\mathcal{A}$.

We will therefore only look at nonlinear inverse problems where $\mathcal{A}$ does *not discard too much information* about the signal. One prominent such nonlinear inverse problem (where reconstruction is often possible) is the *Fourier phase retrieval problem*. It is defined as:

$$\text{find} \quad \mathbf{x} \quad \text{s.t.} \quad \mathbf{y} = |\mathcal{F}\mathbf{x}| \tag{4.7}$$

where $\mathcal{F}\mathbf{x}$ denotes the Fourier transform of the signal $\mathbf{x}$.

We know that the Fourier transform has a well-defined inverse, the *inverse Fourier transform* $\mathcal{F}^{-1}$, but we have seen earlier in Figure 1.3 that due to the absolute operator $|\cdot|$ in (4.7) we cannot simply apply $\mathcal{F}^{-1}$ to $\mathbf{y}$ to recover our signal $\mathbf{x}$.

Fienup algorithm for Fourier phase retrieval

There are, however, other methods available to address the Fourier phase retrieval problem. A prominent one is the *Fienup algorithm*, which solves problem (4.7) by iteratively applying real-space and Fourier constraints. The Fourier constraint is a constraint on the magnitude of the Fourier image, while for the real-space constraint non-negativity is used. Multiple versions of the Fienup algorithm are suggested in the original paper (Fienup, 1982); in Algorithm 2 we describe the most commonly used *hybrid input-output* method (Osherovich, 2012).

Figure 4.1 shows the schematics of Fienup's hybrid input-output algorithm for Fourier phase retrieval. However, there is no proof that the algorithm converges to a global optimum (Shechtman et al., 2014) (Osherovich, 2012).

There exist a variety of other methods to solve the Fourier phase retrieval problem, which are also applicable to a more generalized version of the problem, which will be introduced in the next section.

Algorithm 2: Fienup Hybrid Input-Output algorithm for Fourier phase retrieval (cf. (Shechtman et al., 2014))

Data: Estimate $\mathbf{x}^{(0)}$ of original signal $\mathbf{x}$ (e.g. real-space magnitude, if available, or measured Fourier magnitude with additional random phase)
Fourier magnitude measurement $|\mathcal{F}\mathbf{x}| \in \mathbb{R}^l$
Error threshold ϵ

Result: $\mathbf{x}^{(k+1)}$ that fulfills both real-space magnitude constraints $|\mathbf{x}^{(k+1)}| \approx |\mathbf{x}|$ and Fourier magnitude constraints $|\mathcal{F}\mathbf{x}^{(k+1)}| \approx |\mathcal{F}\mathbf{x}|$

repeat

1. Fourier transform: $\mathbf{z}^{(k)} \leftarrow \mathcal{F}\mathbf{x}^{(k)}$
2. Impose Fourier magnitude constraint: $\mathbf{z}'^{(k)} \leftarrow |\mathcal{F}\mathbf{x}| \frac{\mathbf{z}^{(k)}}{|\mathbf{z}^{(k)}|}$
3. Inverse Fourier transform to obtain a real-space image estimate: $\mathbf{x}'^{(k+1)} \leftarrow \mathcal{F}^{-1}\mathbf{z}'^{(k)}$
4. Apply correction to the real-space image estimate:
$$\mathbf{x}_i^{(k+1)} \leftarrow \begin{cases} \mathbf{x}_i'^{(k+1)}, & i \notin \gamma \\ \mathbf{x}_i^{(k)} - \beta \mathbf{x}_i'^{(k+1)}, & i \in \gamma \end{cases}$$
where β is a small parameter and γ the set of indices for which $\mathbf{x}'^{(k+1)}$ violates the real-space assumptions (e.g. non-negativity, ...)

until $\sum_i \left| |\mathcal{F}\mathbf{x}^{(k+1)}|_i - |\mathcal{F}\mathbf{x}|_i \right|^2 \leq \epsilon$;

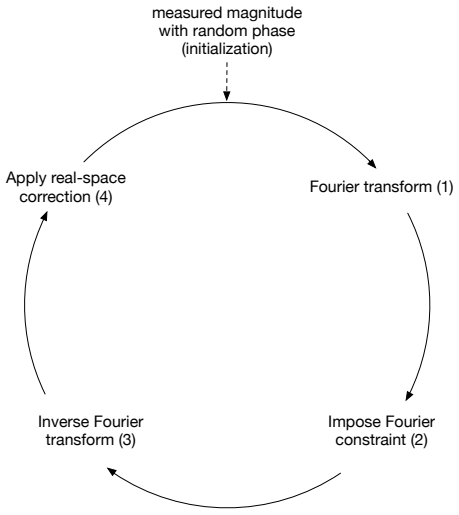


Figure 4.1: Schema of Fienup hybrid input-output algorithm. Figure based on (Shechtman et al., 2014).

4.3 Phase Retrieval from General Measurements

Taking a look at Definition 1.1.1 from Chapter 1, one can formulate the (noise-free) *generalized phase retrieval problem* in a least squares sense:

$$\min_{\mathbf{x}} \sum_{k=1}^{m} (y_k - |\langle \mathbf{a}_k, \mathbf{x} \rangle|^2)^2 \quad \equiv \quad \min_{\mathbf{x}} \| \mathbf{y} - |\mathbf{A}\mathbf{x}|^2 \|^2 \tag{4.8}$$

where the $y_k \in \mathbb{R}$ are the measurements, the vectors $\mathbf{a}_k \in \mathbb{C}^n$ are called the measurement vectors and $\mathbf{x} \in \mathbb{R}^n$ (or $\in \mathbb{C}^n$) (Shechtman et al., 2014).

For the rest of this thesis, we will assume $\mathbf{x}$ to be real-valued, and will refer to Problem 4.8 as *real-valued* generalized phase retrieval.

Stated differently, generalized phase retrieval is the problem of *recovering a signal from only the squared absolute values of its linear measurements*. These measurements are also often called *intensity measurements* (Bandeira et al., 2014).

This sparks a number of questions, namely for which measurement vectors $\mathbf{a}_k$ reconstruction is possible, how many measurements y_k are needed for successful recovery and what kind of signals can be recovered.

If we assume that our data $\mathbf{y}$ consists of quadratic absolute measurements of an unknown input $\mathbf{x} \in \mathbb{R}^n$, then Eldar and Mendelson have shown that $\mathcal{O}(n)$ measurement vectors drawn from a *Gaussian distribution* are sufficient to ensure with overwhelming probability that a unique solution can be found (for the noise-free, *real-valued* case). For that reason one typically assumes the measurement vectors to be Gaussian, i.e. $\mathbf{a}_i \sim \mathcal{N}(0, \mathbf{I})$ or $\mathbf{a}_i \sim \mathcal{N}(0, \mathbf{I}/2) + \mathrm{i}\mathcal{N}(0, \mathbf{I}/2)$ random measurements (Eldar and Mendelson, 2014).

A solution is obviously unique only up to its sign, i.e. given $|\mathbf{A}\mathbf{x}|^2$, we will not be able to distinguish $\mathbf{x}$ and $-\mathbf{x}$ (Eldar and Mendelson, 2014).

Definition 4.3.1. (Pisier, 2016, eq.1.4) A complex-valued random variable X is called $\mathbb{C}$-subgaussian if there is a constant $s \geq 0$ such that for any $x \in \mathbb{C}$

$$\mathbb{E} \exp \mathrm{Re}(xX) \leq \exp \frac{s^2 |x|^2}{2}.$$

Other *subgaussian* distributions are also sometimes used as a random model for the measurement vectors, such as the *Rademacher distribution* $\mathbf{a}_i \sim \{-1, 1\}^n$ (Eaton, 1970) or the (symmetric) Bernoulli distribution $\mathbf{a}_i \sim \{0, 1\}^n$, for which other uniqueness results and other constraints on the domain of admissible signals $\mathbf{x}$ hold (Krahmer and Liu, 2018)(Krahmer and Stöger, 2019). A deeper discussion on suitable distributions for phase retrieval measurement vectors is a field of active research and beyond the scope of this thesis.

4.4 Numerical Reconstruction Methods for Generalized Phase Retrieval

This section will give an overview of some important reconstruction algorithms for the generalized phase retrieval problem. The first class of algorithms con-

sists of methods that make use of semidefinite programming, the second are the already mentioned gradient-type methods. The third class is geometrically-motivated. While semidefinite programming methods will not be used later in this thesis, they are included in this section because global convergence guarantees exist for them, and to be comprehensive in the documentation of the most important modern approaches to phase retrieval.

4.4.1 Semidefinite Relaxation Methods

This class of methods is based on the observation that $y_k = |\langle \mathbf{a}_k, \mathbf{x} \rangle|^2$ actually describes a set of quadratic equations. Let $\mathbf{X} = \mathbf{x}\mathbf{x}^{\mathrm{H}}$ and $\mathbf{A}_k = \mathbf{a}_k \mathbf{a}_k^{\mathrm{H}}$, then

$$y_k = |\langle \mathbf{a}_k, \mathbf{x} \rangle|^2 = \mathbf{x}^{\mathrm{H}} \mathbf{a}_k \mathbf{a}_k^{\mathrm{H}} \mathbf{x} = \mathbf{x}^{\mathrm{H}} \mathbf{A}_k \mathbf{x} = \mathrm{Tr}(\mathbf{A}_k \mathbf{X}). \tag{4.9}$$

The constraint $\mathbf{X} = \mathbf{x}\mathbf{x}^{\mathrm{H}}$ (which is also commonly referred to as *lifting*) can be translated to $\mathbf{X}$ having rank 1 and being positive semidefinite. Our phase retrieval problem can therefore be stated as a semidefinite program, (Shechtman et al., 2014):

$$\begin{aligned} \text{find} \quad & \mathbf{X} \\ \text{s.t.} \quad & y_k = \mathrm{Tr}(\mathbf{A}_k \mathbf{X}) \quad k = 1, ..., m \\ & \mathbf{X} \succeq 0 \\ & \mathrm{rank}(\mathbf{X}) = 1 \end{aligned} \tag{4.10}$$

which is equivalent to the rank minimization problem

$$\begin{aligned} \text{minimize} \quad & \mathrm{rank}(\mathbf{X}) \\ \text{s.t.} \quad & y_k = \mathrm{Tr}(\mathbf{A}_k \mathbf{X}) \quad k = 1, ..., m \\ & \mathbf{X} \succeq 0. \end{aligned} \tag{4.11}$$

Since solving problem (4.11) is known to be NP-hard, we relax the $\mathrm{rank}(\mathbf{X})$ minimization objective with the usage of the convex trace $\mathrm{Tr}(\mathbf{X})$, yielding the semidefinite program

$$\begin{aligned} \text{minimize} \quad & \mathrm{Tr}(\mathbf{X}) \\ \text{s.t.} \quad & y_k = \mathrm{Tr}(\mathbf{A}_k \mathbf{X}) \quad k = 1, ..., m \\ & \mathbf{X} \succeq 0, \end{aligned} \tag{4.12}$$

which can be solved by most convex semidefinite optimization tools (e.g. *MOSEK*[2] or *CVX*[3]). This method is also known as *PhaseLift*. It is able to recover the original vector $\mathbf{x}$ with high probability from $\mathcal{O}(n \log n)$ random Gaussian measurements. Unfortunately, problem (4.12) is higher-dimensional in comparison to the original problem and computationally extremely demanding and is therefore more of theoretical than of practical use for very large signal vectors. (Candes et al., 2011)

4.4.2 Gradient-type Methods

Gradient-based methods that aim at solving the phase retrieval problem with local minimization methods and good initialization have seen significant attraction in the last years. We will introduce some of them here, as they will serve as baseline methods for the generative model-supported methods that will be explored in Chapter 5.

[2] `https://www.mosek.com/`
[3] `http://cvxr.com/cvx/`

Wirtinger Flow

In 2015, Candes et al. proposed the *Wirtinger Flow* algorithm, which is essentially a gradient descent scheme applicable for both *real* and *complex* signals $\mathbf{x}$. In the complex case it makes use of *Wirtinger calculus* to minimize the following nonconvex objective

$$f(\mathbf{x}) := \frac{1}{2m} \sum_{r=1}^{m} L(y_r, |\mathbf{a}_r^{\mathrm{H}} \mathbf{x}|^2), \tag{4.13}$$

where $L(\hat{y}, y)$ is a loss function, typically $L(\hat{y}, y) = (\hat{y} - y)^2$. The algorithm iterates until convergence (or until a maximum amount of iterations is reached) with the following update rule:

$$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} - \frac{\mu^{(k+1)}}{\|\mathbf{x}^{(0)}\|^2} \nabla f(\mathbf{x}^{(k)}) \tag{4.14}$$

where

$$\mu^{(k)} := \min\{1 - \exp(\frac{-k}{k_0}), \mu^{(\max)}\} \tag{4.15}$$

with $k_0 \approx 330$ and $\mu^{(\max)} \approx 0.4$ experimentally found by the original authors (Candes et al., 2015).

For $\mathbf{x} \in \mathbb{R}^n$, the gradient $\nabla f(\mathbf{x})$ is easily defined, but for $\mathbf{x} \in \mathbb{C}^n$, $f : \mathbb{C}^n \to \mathbb{R}$ is not complex-differentiable. The solution to this problem is the usage of *Wirtinger derivatives* (cf. (Wirtinger, 1927) or (Bouboulis, 2010)) for the complex case (Candes et al., 2015). This thesis will only explore the generalized phase retrieval problem for *real-valued* $\mathbf{x}$, and will therefore not go into any more detail about the complex-valued case here.

Being a *local* optimization method, Wirtinger Flow has shown best empirical results when its initialization $\mathbf{x}^{(0)}$ is chosen close to the optimal value. The authors therefore suggest a *spectral method* for this purpose, which sets $\mathbf{x}^{(0)}$ as the leading eigenvector of the positive semidefinite Hermitian matrix $\sum_r y_r \mathbf{a}_r \mathbf{a}_r^{\mathrm{H}}$ normalized to a scaling factor dependent on the size and the intensity sums of the entries of $\mathbf{a}_r$ (Candes et al., 2015). The method is described in pseudocode as Algorithm 3.

Algorithm 3: Basic Wirtinger Flow algorithm (cf. (Candes et al., 2015))

Data: measurements $y_r \in \mathbb{R}$, $r = 1, ..., m$
measurement vectors $\mathbf{a}_r \in \mathbb{R}^n$, $r = 1, ..., m$
number of iterations $k_{\max}$
Result: reconstruction $\mathbf{x}^{(k_{\max})}$
$\lambda^2 := n \frac{\sum_r y_r}{\sum_r \|\mathbf{a}_r\|^2}$
Set $\mathbf{x}^{(0)}$ as the leading eigenvector of $\mathbf{Y} = \frac{1}{m} \sum_{r=1}^{m} y_r \mathbf{a}_r \mathbf{a}_r^{\mathrm{H}}$ normalized to $\|\mathbf{x}^{(0)}\| = \lambda$.
for $k = 0$ **to** $k_{\max} - 1$ **do**
  $\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} - \frac{\mu^{(k+1)}}{\|\mathbf{x}^{(0)}\|^2} \nabla f(\mathbf{x}^{(k)})$
end

Wirtinger Flow with spectral initialization has been shown to work for both real and complex Gaussian measurements $\mathbf{a}_r$ (with convergence at a linear rate

if the number of measurements m are in the order of $n(\log n)^2)$ as well as for measurements sampled i.i.d. from a random variable d for which

$$\mathbb{E}d = 0, \quad \mathbb{E}d^2 = 0 \quad \text{and} \quad \mathbb{E}|d|^2 = 2(\mathbb{E}|d|^2)^2 \tag{4.16}$$

holds (Candes et al., 2015).

Also, Chen et al. were able to show in 2019 that for real or complex Gaussian measurements, a gradient descent scheme converges to the true solution for $m \gtrsim n \operatorname{poly} \log m$ even when randomly initialized (Chen et al., 2019).

Truncated Wirtinger Flow

An extension of the Wirtinger Flow algorithm has been proposed in 2015 by Chen and Candes (Chen and Candes, 2015), in which both the initialization and the iterative gradient update steps are regularized by restricting them to a subset of indices that varies with each iteration. The authors also proposed to fix the step size for all iterations and to exchange the loss function L of the original Wirtinger Flow algorithm with the Poisson log-likelihood function

$$L(\mathbf{x}, y_i) := y_i \log(|\mathbf{a}_i^{\mathrm{H}}\mathbf{x}|^2) - |\mathbf{a}_i^{\mathrm{H}}\mathbf{x}|^2 \tag{4.17}$$

which results (for the real case) in the fast to calculate gradient

$$\nabla L(\mathbf{x}, y_i) = 2\left(\frac{y_i - |\mathbf{a}_i^{\top}\mathbf{x}|^2}{\mathbf{a}_i^{\top}\mathbf{x}}\mathbf{a}_i\right) \tag{4.18}$$

and which assumes a Poisson data model, i.e.

$$y_i \sim \operatorname{Poisson}(|\langle \mathbf{a}_i, \mathbf{x}\rangle|^2). \tag{4.19}$$

The *Truncated Wirtinger Flow* algorithm then breaks down into two steps:

- As initialization, we compute an estimate $\mathbf{x}^{(0)}$ using a spectral method as in the Wirtinger Flow method, but applying it only to a subset $\mathcal{T}^{(0)}$ of our measurements y_i which are not *too large*. The idea comes from the observation that the initialization method of Wirtinger Flow, which uses the first eigenvector of $\sum_i y_i \mathbf{a}_i \mathbf{a}_i^{\mathrm{H}}$, is subject to issues arising from very large y_i dominating the whole sum and therefore essentially not providing a good initial estimate.
- We iterate using the gradient of the Poisson log-likelihood loss function but also restrict it to only use the gradient components that are again not *too large* or *too small* (because in the case of the gradient of this specific loss function they can be seen as outlier components). Figure 4.2 shows the locus over all unit vectors of $\mathbf{a}_i$ of the negative gradient of the Poisson log-likelihood function for some exemplary running estimate $\mathbf{z}$ and true value $\mathbf{x}$ and shows that good descent directions can be found by restricting the length of the gradient components to not be too large or too small.

The full Truncated Wirtinger Flow algorithm is given in pseudocode in Listing 4.

The authors of the original paper showed that under Gaussian measurements, Truncated Wirtinger Flow achieves exact recovery from $\mathcal{O}(n)$ measurements and furthermore suggest that the method is applicable to other subgaussian measurement vectors as well, although truncation thresholds need to be tweaked (Chen and Candes, 2015).

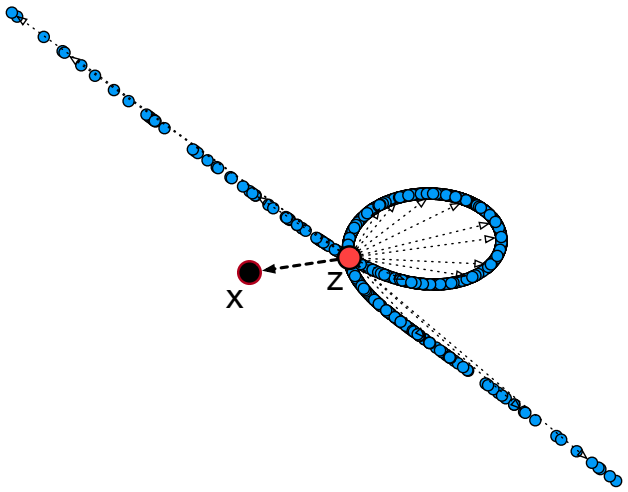


Figure 4.2: Locus over unit vectors of $\mathbf{a}_i$ of the negative gradient of the Poisson log-likelihood function $\frac{|\mathbf{a}_i^{\mathrm{H}}\mathbf{z}|^2 - |\mathbf{a}_i^{\mathrm{H}}\mathbf{x}|^2}{\mathbf{a}_i^{\mathrm{H}}\mathbf{z}}\mathbf{a}_i$ for running estimate $\mathbf{z}$ and true value $\mathbf{x}$. Based on a figure by (Chen and Candes, 2015).

Algorithm 4: Truncated Wirtinger Flow algorithm (cf. (Chen and Candes, 2015))

Data: measurements $y_i \in \mathbb{R}$, $i = 1, ..., m$
measurement vectors $\mathbf{a}_i \in \mathbb{R}^n$, $i = 1, ..., m$
number of iterations $k_{\max}$
parameters a_z^{lb} (default 0.3), a_z^{ub} (default 5), a_h (default 5), a_y (default 3), μ_k (default 0.2)

Result: reconstruction $\mathbf{x}^{(k_{\max})}$

$\lambda_0 := \sqrt{\frac{1}{m}\sum_{i=1}^m y_i}$

Let $\tilde{\mathbf{x}}$ be the leading eigenvector of $\mathbf{Y} := \frac{1}{m}\sum_{i=1}^m y_i \mathbf{a}_i \mathbf{a}_i^{\mathrm{H}} \mathbf{1}_{\{|y_i| \le a_y^2 \lambda_0^2\}}$

$\mathbf{x}^{(0)} := \sqrt{\frac{mn}{\sum_{i=1}^m \|\mathbf{a}_i\|^2}}\lambda_0 \tilde{\mathbf{x}}$

for $k = 0$ **to** $k_{\max} - 1$ **do**

$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + \frac{2\mu_k}{m}\sum_{i=1}^m \frac{y_i - |\mathbf{a}_i^{\mathrm{H}}\mathbf{x}^{(k)}|^2}{\mathbf{x}^{(k)\mathrm{H}}\mathbf{a}_i}\mathbf{a}_i \mathbf{1}_{\mathcal{E}_1^i \cap \mathcal{E}_2^i}$

where $\mathcal{E}_1^i := \left\{ a_z^{\mathrm{lb}} \le \frac{\sqrt{n}}{\|\mathbf{a}_i\|}\frac{|\mathbf{a}_i^{\mathrm{H}}\mathbf{x}^{(k)}|}{\|\mathbf{x}^{(k)}\|} \le a_z^{\mathrm{ub}} \right\}$,

$\mathcal{E}_2^i := \left\{ |y_i - |\mathbf{a}_i^{\mathrm{H}}\mathbf{x}^{(k)}|^2| \le a_h K_k \frac{\sqrt{n}}{\|\mathbf{a}_i\|}\frac{|\mathbf{a}_i^{\mathrm{H}}\mathbf{x}^{(k)}|}{\|\mathbf{x}^{(k)}\|} \right\}$

and $K_k := \frac{1}{m}\sum_{l=1}^m |y_l - |\mathbf{a}_l^{\mathrm{H}}\mathbf{x}^{(k)}|^2|$

end

Sparsity-based Methods

Another important subclass of gradient-based methods are sparsity-based methods, which assume that the signal to recover is k-sparse, i.e. it has a maximum of k nonzero entries. *GESPAR (Greedy Sparse Phase Retrieval)* (Shechtman et al., 2013) is such an iterative algorithm that solves problem (4.6) under the prior knowledge that the sought-after signal $\mathbf{x}$ is sparse in some *known* representation. As already stated in the introduction, this means that $\mathbf{x}$ can be expressed in terms of a sparsity basis $\boldsymbol{\Psi}$ and a sparse vector $\boldsymbol{\alpha}$ as

$$\mathbf{x} = \boldsymbol{\Psi}\boldsymbol{\alpha}. \tag{4.20}$$

The details of the GESPAR algorithm are out of scope of this thesis, but on a high-level, the algorithm works like this: starting from an initial random support set for $\mathbf{x}$, GESPAR executes a gradient descent (damped Gauss-Newton) method to obtain an estimate of $\mathbf{x}$ based on which it is updating its support set. This continues until convergence (Shechtman et al., 2013)(Shechtman et al., 2014).

With knowledge of the sparsity factor k and the sparsity basis, GESPAR has shown to provide fast and accurate results for a variety of phase retrieval applications.

The important bottom line of sparsity-based methods is that we can improve solutions to the (generalized) phase retrieval problem by introducing prior knowledge about the domain of permissible signals. In a lot of practical cases, signals will not be arbitrary but will be part of a family of permissible signals. We can use this information in our reconstruction processes.

4.4.3 Randomized Kaczmarz Method

The last method that will be explored in this chapter is based on the *Randomized Kaczmarz* method for solving linear equations (Strohmer and Vershynin, 2007). In the setting of the original paper, the system

$$\mathbf{y} = \mathbf{A}\mathbf{x} \tag{4.21}$$

is understood as defining a set of hyperplanes $y_i = \langle \mathbf{a}_i, \mathbf{x} \rangle$. The idea of the algorithm is now to iterate by projecting the running estimate of $\mathbf{x}$ onto the hyperplane of a randomly chosen equation. Strohmer and Vershynin proved that this method converges linearly to the true solution for a large class of matrices $\mathbf{A}$ (Strohmer and Vershynin, 2007).

The method can also be applied to the generalized phase retrieval problem with magnitude measurements (Wei, 2015):

$$\sqrt{y_i} = |\langle \mathbf{a}_i, \mathbf{x} \rangle|, \quad \mathbf{x} \in \mathbb{R}^n,\ \mathbf{a}_i \in \mathbb{C}^n \tag{4.22}$$

and is based on the observation that each equation (4.22) defines *two* hyperplanes, one corresponding to $\mathbf{x}$ and one to $-\mathbf{x}$ (Tan and Vershynin, 2017). The method now projects the running estimate of $\mathbf{x}$ onto the closer of the two hyperplanes of a randomly chosen equation.

Based on an initialization value $\mathbf{x}^{(0)}$, which can be found e.g. using a spectral initialization method described in the last sections, we can now iteratively define

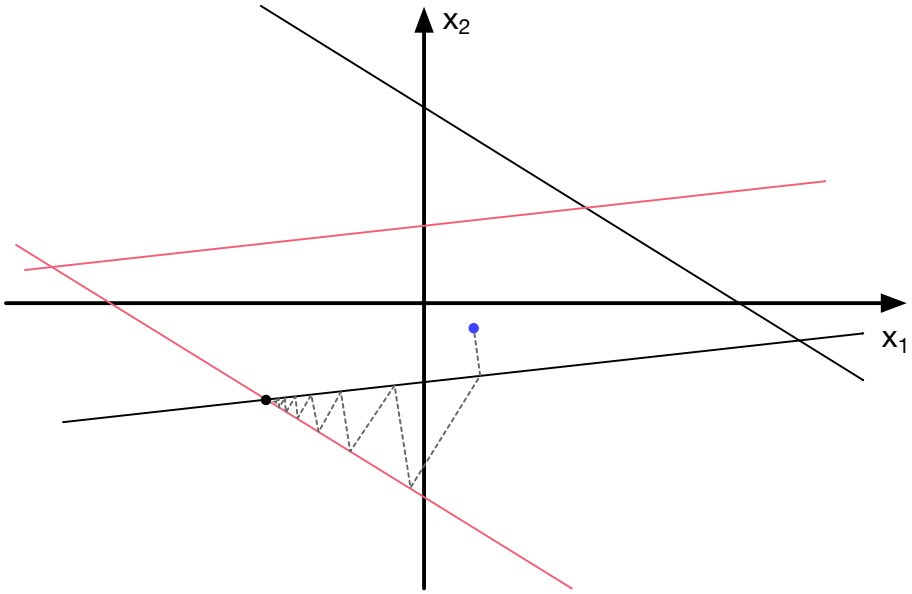


Figure 4.3: Intuition for the *Randomized Kaczmarz* method for a system of two equations $3x_1 + 4x_2 = \pm 12$ and $x_1 - 3x_2 = \pm 6$. The blue dot is the initial estimate, and the dashed line indicates the update steps towards a true solution (black dot).

the randomized Kaczmarz update rule:

$$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + \left(\frac{\text{sign}(\langle \mathbf{a}_{r(k+1)}, \mathbf{x}^{(k)} \rangle) \sqrt{y_{r(k+1)}} - \langle \mathbf{a}_{r(k+1)}, \mathbf{x}^{(k)} \rangle}{\|\mathbf{a}_{r(k+1)}\|_2^2} \right) \mathbf{a}_{r(k+1)} \tag{4.23}$$

where $r(k)$ describes the index of the randomly (with a probability proportional to $\|\mathbf{a}_{r(k+1)}\|_2^2$) chosen equation at iteration k.

Despite its simplicity, Wei was able to show that the *Randomized Kaczmarz* method achieves good performance for the generalized phase retrieval problem based on a variety of experiments using real and complex Gaussian random measurement vectors (Wei, 2015).

Figure 4.3 shows the intuition behind the algorithm for a very simple 2-dimensional real-valued phase retrieval case.

Chapter 5

Solving Nonlinear Inverse Problems with Deep Generative Models

We have seen in the last chapter that prior information about permissible signals $\mathbf{x}$ can drastically improve the reconstruction process of gradient-based methods, and it is this observation that we want to use to motivate the usage of *generative models* (see Section 3.5) as prior information for generalized phase retrieval algorithms.

Generative models *based on deep (feedforward) neural networks* are particularly interesting as they are able to learn to generate samples even from very complicated signal distributions (such as e.g. natural images (Gulrajani et al., 2016) or faces (Karras et al., 2018)).

There are two major approaches that we want to distinguish in this chapter:

- Using deep generative models as *data priors*. Here, the inverse problem is again stated as an optimization problem, but this time, the desired signal $\mathbf{x} \in \mathbb{R}^n$ is replaced by the output of a generator network $G : \mathbb{R}^k \to \mathbb{R}^n$ trained on the signal domain $\mathcal{X}$ that $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^n$ belongs to. The resulting problem is

$$\min_{\mathbf{z}\in\mathbb{R}^k} \sum_{i=1}^{m} L(y_i, |\langle \mathbf{a}_i, G(\mathbf{z})\rangle|^2), \tag{5.1}$$

  which for a quadratic loss results in

$$\min_{\mathbf{z}\in\mathbb{R}^k} \|\mathbf{y} - |\mathbf{A}G(\mathbf{z})|^2\|_2^2. \tag{5.2}$$

- Using deep generative-model supported reconstruction methods as *initializers* for *traditional* reconstruction methods. In this approach, we take advantage of the good (but not perfect) results of the methods that will be introduced in Section 5.1 and use them as initial values in traditional generalized phase retrieval methods (which we explored in Section 4.4).

5.1 Deep Generative Models as Priors

Using deep generative models as priors follows the idea that if we have a differentiable generator network G that is able to approximately generate the domain $\mathcal{X} \subset \mathbb{R}^n$ of all permissible signals $\mathbf{x}$, then we can replace each occurrence of $\mathbf{x}$ in the optimization objective with $G(\mathbf{z})$.

We therefore get the following non-convex optimization problem in empirical risk formulation:

$$\mathbf{z}^* := \underset{\mathbf{z} \in \mathcal{D}(G)}{\operatorname{argmin}} \sum_{i=1}^{m} L(y_i, |\langle \mathbf{a}_i, G(\mathbf{z}) \rangle|^2), \tag{5.3}$$

which we solve using gradient descent methods. Instead of reconstructing our signal $\mathbf{x}^*$ directly, we solve problem (5.3) to obtain $\mathbf{z}^*$ and then plug this into our generator G to yield the reconstruction $G(\mathbf{z}^*) = \mathbf{x}^*$.

Remark. Hand et al. were able to prove in 2018 that for differentiable generator networks consisting of layers with ReLU activation functions, the objective function in (5.3) does not (with overwhelming probability) have spurious local minima away from neighborhoods of the true solution (or negative multiples thereof) under a squared loss and Gaussian $\mathbf{a}_i$ (Hand et al., 2018).

An analogous proof has been provided by Hand and Voroninski in 2017 for the *linear* inverse problem formulation

$$\min_{\mathbf{z} \in \mathcal{D}(G)} \sum_{i=1}^{m} (y_i - \langle \mathbf{a}_i, G(\mathbf{z}) \rangle)^2 \tag{5.4}$$

(Hand and Voroninski, 2017), which is in line with the results of Bora et al. (Bora et al., 2017), who proved that the linear problem (5.4) can be solved with high probability for d-layer generators $G : \mathbb{R}^k \to \mathbb{R}^n$ of variational autoencoders or generative adversarial networks with *arbitrary* activation functions using $\mathcal{O}(kd \log n)$ measurements. Furthermore, in 2019, Asim et al. were experimentally validating an approach to solve the regularized linear problem

$$\min_{\mathbf{z} \in \mathcal{D}(G)} \sum_{i=1}^{m} (y_i - \langle \mathbf{a}_i, G(\mathbf{z}) \rangle)^2 + \lambda \|\mathbf{z}\|_2 \tag{5.5}$$

using L-BFGS with great success on important computer vision datasets (Asim et al., 2019).

Based on this prior research, we hypothesize that we can solve equation (5.3) using gradient descent methods also for generator networks that are not exclusively based on *ReLU* activation functions. We will experimentally validate this assumption in the next sections.

We we will now explore an exemplary gradient-based method for the solution of a regularized version of problem (5.3) and will evaluate its performance in Section 5.3.

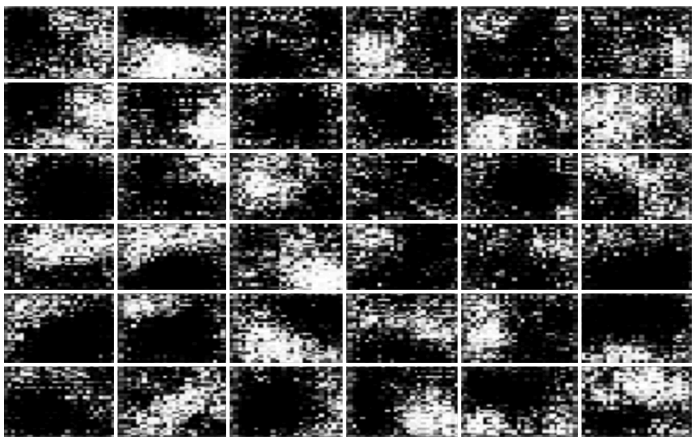


Figure 5.1: 6×6 samples from the generator range of a badly trained neural network on the Shepp-Logan dataset.

5.1.1 Deep Regularized Gradient Descent

Let $G : \mathbb{R}^k \to \mathbb{R}^n$ be a generator which has been trained in a way that its range approximately resembles the signal domain $\mathcal{X}$ of a signal $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^n$ and let $\mathbf{y} = \left[|\langle \mathbf{a}_1, \mathbf{x}\rangle|^2, ..., |\langle \mathbf{a}_m, \mathbf{x}\rangle|^2\right]^\top$ be the measurements. We are now interested in solving the following regularized optimization problem

$$\min_{\mathbf{z} \in \mathcal{D}(G)} \sum_{i=1}^{m} \left((y_i - |\langle \mathbf{a}_i, G(\mathbf{z})\rangle|^2)^2\right) + \lambda \|G(\mathbf{z})\|_{\mathrm{TV}} \tag{5.6}$$

or in matrix-vector notation

$$\min_{\mathbf{z} \in \mathcal{D}(G)} \|\mathbf{y} - |\mathbf{A}G(\mathbf{z})|^2\|_2^2 + \lambda \|G(\mathbf{z})\|_{\mathrm{TV}} \tag{5.7}$$

This problem is similar to the formulation of problem (5.3), but adds a *discrete anisotropic total variation norm* term $\lambda\|G(\mathbf{z})\|_{\mathrm{TV}}$ (see Definition 4.2.1) to account for and remove noise-like artifacts in the range of the generator (which can be the case especially when the generator has not been properly trained – Figure 5.1 shows examples of artifacts in the range of a badly trained generator).

Remark. In general, the quality of the generator, i.e. how well the generator is able to represent the signal domain, is upper-bounding the reconstruction quality. A badly trained (non-expressive) generator will mean that the best approximation $G(\mathbf{z}) \approx \mathbf{x}$ of a signal $\mathbf{x}$ might be poor (meaning that the model error $||\mathbf{x} - G(\mathbf{z})||$ will be large), which in turn will result in poor reconstruction results because the reconstruction *has* to be part of the generator range $\mathcal{R}(G)$.

The approach to solve (5.7) is based on the work by Asim et al. and Shamshad and Ahmed on generative prior-based reconstruction methods for

linear problems (Asim et al., 2019) and phase retrieval (Shamshad and Ahmed, 2018) and extends the latter by the additional regularization term $\lambda\|G(\mathbf{z})\|_{\mathrm{TV}}$ to improve the overall reconstruction results.

We will solve problem (5.7) using standard (sub-)gradient descent with a fixed step size as is suggested in Algorithm 5. However, the objective function being composed of two separate terms would also motivate the usage of a proximal method (Parikh et al., 2014), e.g. the *proximal gradient method* (otherwise also known as *forward backward splitting algorithm*), which we will not discuss in this thesis.

Algorithm 5: Deep Regularized Gradient Descent (DRGD) generalized phase retrieval algorithm

Data: measurements $\mathbf{y} = [y_1, ..., y_m]^\top \in \mathbb{R}^m$
sensing matrix $\mathbf{A} = [\mathbf{a}_1, ..., \mathbf{a}_m]^\top \in \mathbb{R}^{m \times n}$
differentiable generator network $G(\mathbf{z}) : \mathbb{R}^p \to \mathbb{R}^n$
step size η
regularization parameter λ
number of iterations $k_{\max}$
Result: reconstruction $\mathbf{x}^{(k_{\max})}$
Randomly initialize $\mathbf{z}^{(0)} \in \mathbb{R}^p$
for $k = 0$ **to** $k_{\max} - 1$ **do**
  $\mathbf{z}^{(k+1)} \leftarrow \mathbf{z}^{(k)} - \eta \nabla_{\mathbf{z}^{(k)}} \left(\|\mathbf{y} - |\mathbf{A}G(\mathbf{z}^{(k)})|^2\|_2^2 + \lambda \|G(\mathbf{z}^{(k)})\|_{\mathrm{TV}} \right)$
end
$\mathbf{x}^{(k_{\max})} \leftarrow G(\mathbf{z}^{(k_{\max})})$

5.2 Deep Generative Initialization

Given the shortcomings of the deep generative prior-based reconstruction methods described in Section 5.1.1 with respect to the generator's model error, we now want to investigate a *hybrid approach* that takes the reconstruction result of a generative prior-based method (e.g. *Deep Regularized Gradient Descent*) and uses that as the initializer of another method (e.g. *Truncated Wirtinger Flow* or the *Randomized Kaczmarz* method).

Conceptually, this hybrid algorithm works like this:

1. We first reconstruct an approximate $\tilde{\mathbf{x}}$ using a randomly initialized generative prior-based reconstruction method, which solves (5.3):

$$\tilde{\mathbf{x}} := G\left(\underset{\mathbf{z} \in \mathcal{D}(G)}{\operatorname{argmin}} \sum_{i=1}^{m} L(y_i, |\langle \mathbf{a}_i, G(\mathbf{z}) \rangle|^2) \right). \tag{5.8}$$

   As we have seen, the similarity of $\tilde{\mathbf{x}}$ to the true $\mathbf{x}$ is depending on the generator G's ability to correctly model the signal domain.

2. We then use $\tilde{\mathbf{x}}$ as the initializer $\mathbf{x}^{(0)}$ for one of the traditional reconstruction methods in order to solve

$$\underset{\mathbf{x} \in \mathbb{R}^n}{\operatorname{argmin}} \sum_{i=1}^{m} L(y_i, |\langle \mathbf{a}_i, \mathbf{x} \rangle|^2). \tag{5.9}$$

Doing this, we overcome the model error of the generator G but are still able to use the prior knowledge encoded into it without compromising reconstruction quality.

Our hypothesis is that based on this data-driven initialization $\mathbf{x}^{(0)}$ of the reconstruction method, we get better reconstruction results as we start closer to the true value of $\mathbf{x}$. We furthermore need less iterations to get to comparable reconstruction errors.

In comparison to the *spectral initialization* method employed by most reconstruction methods, our initialization method is also significantly faster. Figure 5.12 shows comparisons of both initialization methods with respect to runtime.

We want to highlight the combination of the *Deep Regularized Gradient Descent* and *Randomized Kaczmarz* algorithms, which we will consequently name *Deep Regularized Gradient Descent-initialized Randomized Kaczmarz (DRGD-RK)*. We will use this method as an example of *deep generative initialization* and will evaluate its performance in Section 5.3. The pseudocode of the algorithm can be found in Listing 6.

Algorithm 6: Deep Regularized Gradient Descent-initialized Randomized Kaczmarz (DRGD-RK) generalized phase retrieval algorithm

Data: measurements $\mathbf{y} = [y_1, ..., y_m]^\top \in \mathbb{R}^m$
sensing matrix $\mathbf{A} = [\mathbf{a}_1, ..., \mathbf{a}_m]^\top \in \mathbb{R}^{m \times n}$
differentiable generator network $G(\mathbf{z}) : \mathbb{R}^p \to \mathbb{R}^n$
step size η
regularization parameter λ
number of iterations of the initializer $i_{\max}$
number of iterations of the randomized Kaczmarz method $k_{\max}$
Result: reconstruction $\mathbf{x}^{(k_{\max})}$
Randomly initialize $\mathbf{z}^{(0)} \in \mathbb{R}^p$
for $i = 0$ **to** $i_{\max} - 1$ **do**
  $\mathbf{z}^{(i+1)} \leftarrow \mathbf{z}^{(i)} - \eta \nabla_{\mathbf{z}^{(i)}} \left(\|\mathbf{y} - |\mathbf{A}G(\mathbf{z}^{(i)})|^2\|_2^2 + \lambda \|G(\mathbf{z}^{(i)})\|_{\mathrm{TV}} \right)$
end
$\mathbf{x}^{(0)} \leftarrow G(\mathbf{z}^{(i_{\max})})$
for $k = 0$ **to** $k_{\max} - 1$ **do**
  $\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} + \left(\frac{\mathrm{sign}(\langle \mathbf{a}_{r(k+1)}, \mathbf{x}^{(k)} \rangle) \sqrt{y_{r(k+1)}} - \langle \mathbf{a}_{r(k+1)}, \mathbf{x}^{(k)} \rangle}{\|\mathbf{a}_{r(k+1)}\|_2^2} \right) \mathbf{a}_{r(k+1)}$
end

5.3 Numerical Experiments

In this section we will present an experimental numerical evaluation of our methods (*Deep Regularized Gradient Descent (DRGD)* and *Deep Regularized Gradient Descent-initialized Randomized Kaczmarz (DRGD-RK)*) against traditional generalized phase retrieval methods (*Wirtinger Flow (WF)*, *Truncated Wirtinger Flow (TWF)*, and *Randomized Kaczmarz (RK)*) on one standard test dataset and one synthetically generated dataset.

We evaluate the reconstructions based on *structural similarity index* SSIM and *peak signal-to-noise ratio* PSNR, two important quality metrics commonly used for image quality evaluation tasks.

Definition 5.3.1.

$$\mathrm{PSNR}(\hat{\mathbf{x}}, \mathbf{x}) := 10 \cdot \log_{10}\left(\frac{\mathrm{MAX}_{\mathbf{x}}^2}{\frac{1}{n}\sum_i (\hat{x}_i - x_i)^2}\right)$$

(where $\mathrm{MAX}_{\mathbf{x}}$ is the maximal value that each entry of the signal vector $\mathbf{x}$ can take) is the *peak signal-to-noise ratio* between a predicted or reconstructed signal $\hat{\mathbf{x}} = [\hat{x}_1, ..., \hat{x}_n]^\top \in \mathbb{R}^n$ and the original signal $\mathbf{x} = [x_1, ..., x_n]^\top \in \mathbb{R}^n$ (Hore and Ziou, 2010) (Madisetti, 1997, ch. 32.5).

Definition 5.3.2. The *structural similarity index* for image quality (Wang et al., 2004) is defined as

$$\mathrm{SSIM}(\mathbf{X}, \mathbf{Y}) := \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)},$$

where μ_x, μ_y are local means, σ_x, σ_y are local standard deviations and σ_{xy} is the local cross-covariance of images $\mathbf{X}, \mathbf{Y}$. c_1, c_2 are defined based on the dynamic range of the input image.[1]

5.3.1 MNIST dataset

The first dataset on which to evaluate the reconstruction methods is the well-known MNIST dataset (LeCun, 1998), which is a classic machine learning dataset consisting of $60,000$ handwritten digits represented as 28×28 pixel greyscale images. It is usually separated into $50,000$ images in the training set and $10,000$ images in the test set. Figure 5.2 shows samples from the MNIST dataset.

To evaluate the performance of our deep generative model-supported reconstruction algorithms, we trained two generative models on this dataset:

- *Variational Autoencoder:* Our variational autoencoder consists of two fully-connected leakyrelu (see (3.11)) layers as the encoder and a fully-connected leakyrelu and fully-connected sigmoid layer as the generator.

  We are using an ℓ^2-regularized ELBO (see (3.15))

$$L(q; \boldsymbol{\theta}) = \mathrm{ELBO}(q) + 0.01||\boldsymbol{\theta}||^2 \tag{5.10}$$

[1] The implementation used in this thesis follows the default values in `https://www.cns.nyu.edu/~lcv/ssim/ssim.m`.

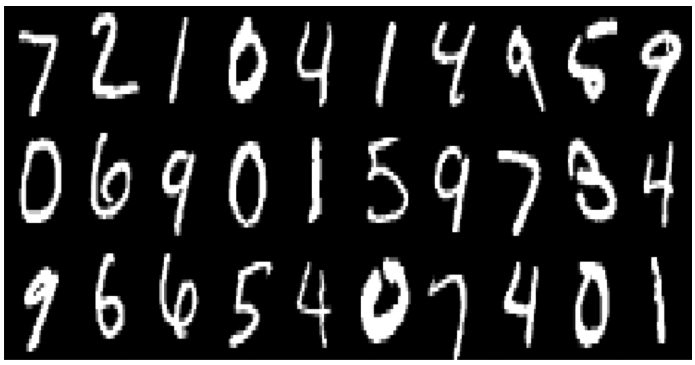


Figure 5.2: 30 samples taken from the MNIST dataset.

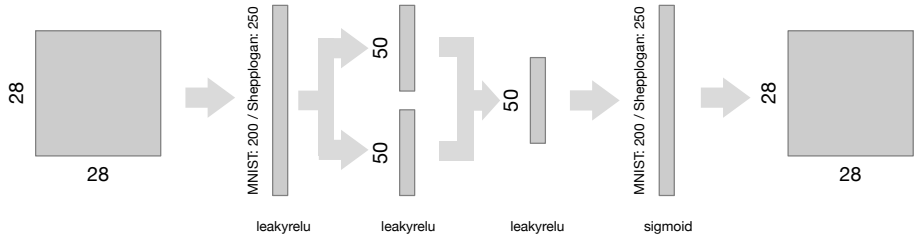


Figure 5.3: Architecture of the variational autoencoder trained on MNIST and the Shepp-Logan dataset.

as the loss function (where $\boldsymbol{\theta}$ is the vector of parameters of the generative model).

Figure 5.3 shows the architecture of the variational autoencoder trained for 50 epochs on the $50,000$ image MNIST training set.

- *Convolutional Variational Autoencoder:*

  For our evaluations, a convolutional variational autoencoder has been trained for 5 epochs.

  The convolutional variational autoencoder consists of four convolutional and two fully-connected tanh layers as the encoder, and one fully-connected tanh plus four convolutional layers as the generator. Figure 5.4 shows the architecture of the convolutional variational autoencoder trained on MNIST.

  The loss function used for the convolutional variational autoencoder is identical to the one used in the variational autoencoder (5.10).

5.3.2 Shepp-Logan dataset

This synthetically generated dataset is inspired by the well-known *Shepp-Logan phantom* (see Figure 5.5) by randomizing some of its parameters (see Appendix A.1 for the code that was used to generate the samples). Each image is again 28×28 pixels in size and greyscale. The overall dataset has $250,000$ randomly generated images. Figure 5.6 shows samples of this dataset.

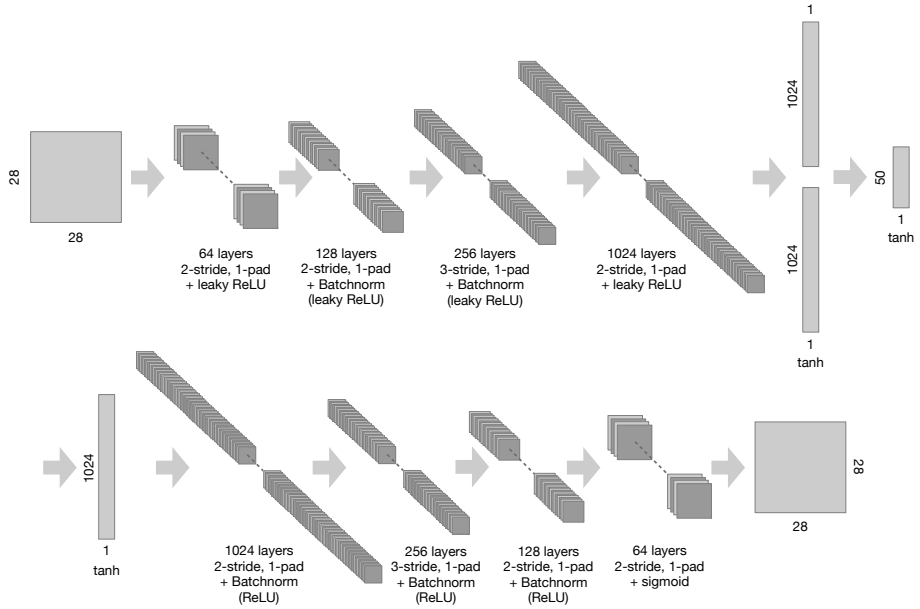


Figure 5.4: Architecture of the convolutional variational autoencoder trained on MNIST and the Shepp-Logan dataset.

Figure 5.5: Shepp-Logan phantom image as defined in (Shepp and Logan, 1974).

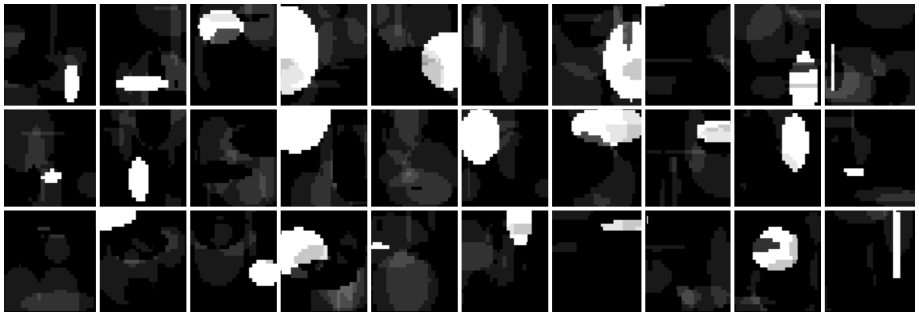


Figure 5.6: 30 samples taken from the synthetically generated Shepp-Logan dataset.

We again evaluated the performance of all algorithms on two generative models:

- *Variational Autoencoder:* We are using the same network design (but with different layer sizes, see Figure 5.3), the same number of epochs and the same regularized loss function (5.10) as for the MNIST dataset in Section 5.3.1.

- *Convolutional Variational Autoencoder:* For our evaluations, the convolutional variational autoencoder has been trained for 5 epochs.

  Again, we are using the same network design and loss function (5.10) as in Section 5.3.1.

5.3.3 Noise-free Measurements

All evaluations are performed under a noise-free measurement model $\mathbf{y} = |\mathbf{A}\mathbf{x}|^2$ using *complex* random Gaussian measurement matrices $\mathbf{A}$. We evaluate the methods based on their reconstruction quality and runtime.

The methods used in this evaluation are parameterized as follows:

1. *Wirtinger Flow* uses $k_{\max} = 50$ iterations.

2. *Truncated Wirtinger Flow* uses $k_{\max} = 200$ iterations.

3. *Randomized Kaczmarz* uses $k_{\max} = 100000$ iterations.[2]

4. *Deep Regularized Gradient Descent* uses $k_{\max} = 200$ iterations with a step size of $\eta = 0.1$ and a regularization factor of $\lambda = 0.1$.

5. *Deep Regularized Gradient Descent-initialized Randomized Kaczmarz* uses for the initializer $i_{\max} = 200$ iterations with a step size of $\eta = 0.1$ and a regularization factor of $\lambda = 0.1$, and for the *Randomized Kaczmarz* part an iteration count of $k_{\max} = 100000$.

[2]Note that *Randomized Kaczmarz*, unlike *Wirtinger Flow* or *Truncated Wirtinger Flow*, is not a gradient-based method and that iteration counts therefore are not comparable. In general, *Randomized Kaczmarz* iterations are much faster to execute than gradient steps.

Figures 5.7 and 5.8 visually show the reconstruction results for selected images from the MNIST and Shepp-Logan datasets and highlight the most important results.

Figures 5.9 to 5.11 show the evaluation results with respect to reconstruction quality under SSIM and PSNR for the MNIST and the Shepp-Logan dataset.

Main findings and observations

One can see that for both datasets, *Deep Regularized Gradient Descent-initialized Randomized Kaczmarz* and *Deep Regularized Gradient Descent* show superior reconstruction quality in the lower sampling regime (at sampling rates of 12.5% to around 400%). Reconstruction quality of *Deep Regularized Gradient Descent-initialized Randomized Kaczmarz* is slightly lower for the high sampling rate in comparison to *Truncated Wirtinger Flow*, *Wirtinger Flow* and *Randomized Kaczmarz*. This effect can be attributed to the fixed number of randomized Kaczmarz iterations (100000) in this experiment and is especially visible in the results with respect to the *peak signal-to-noise ratio* PSNR.

Because it only allows solutions that lie in the range of the generator, *Deep Regularized Gradient Descent* fails to deliver competitive reconstruction results for high sampling rates due to an inability of the (convolutional) variational autoencoder to model the original signal distribution well enough. This effect is not visible in reconstructions using the *Deep Regularized Gradient Descent-initialized Randomized Kaczmarz* algorithm, because in this method the final solution is not bound to be in the range of the generator network.

The difference in reconstruction quality between *Deep Regularized Gradient Descent(-initialized Randomized Kaczmarz)* based on variational autoencoders and *Deep Regularized Gradient Descent(-initialized Randomized Kaczmarz)* based on *convolutional* variational autoencoders is marginal in our experiments and is suspected to be due to the different model errors of the variational autoencoders and convolutional variational autoencoders. We see however that *Deep Regularized Gradient Descent(-initialized Randomized Kaczmarz)* based on *convolutional* variational autoencoders has a slightly better runtime behavior due to the efficiency of the convolution operations.

The evaluation results below clearly show the superior runtime performance of *Deep Regularized Gradient Descent-initialized Randomized Kaczmarz* and *Deep Regularized Gradient Descent* compared to *Truncated Wirtinger Flow* and *Wirtinger Flow* (see Figure 5.11). It also shows that *Deep Regularized Gradient Descent-initialized Randomized Kaczmarz* achieves preferable reconstruction quality over traditional generalized phase retrieval methods given a differentiable generator network which is able to model the signal domain well enough.

Spectral initialization vs. deep generative initialization

We also compare the runtime of the well-known spectral initialization method (compare the initialization from Algorithm 3) with a deep generative initialization (as introduced in Section 5.2). The results (averaged over 5 randomly chosen images) are shown in Figure 5.12.

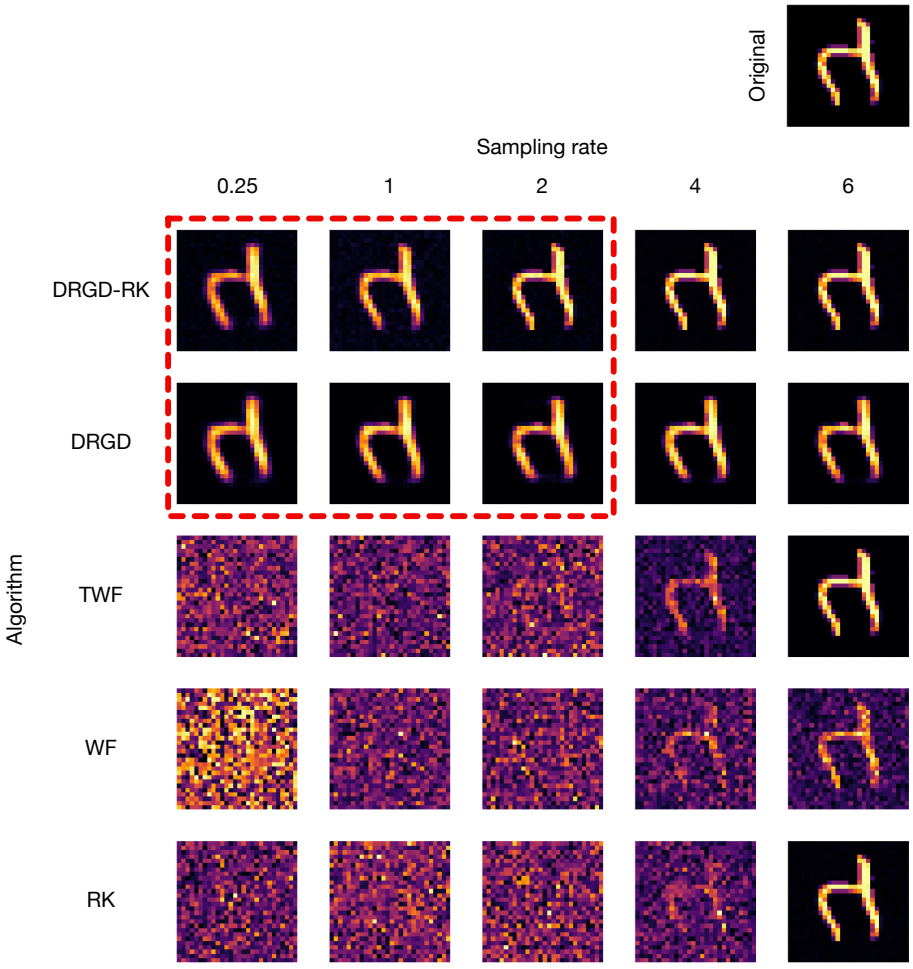


Figure 5.7: Results of the reconstruction process for a selected MNIST test image for all algorithms and selected sampling rates. *DRGD* and *DRGD-RK* make use of a trained variational autoencoder. Important results are highlighted with a dashed red box.

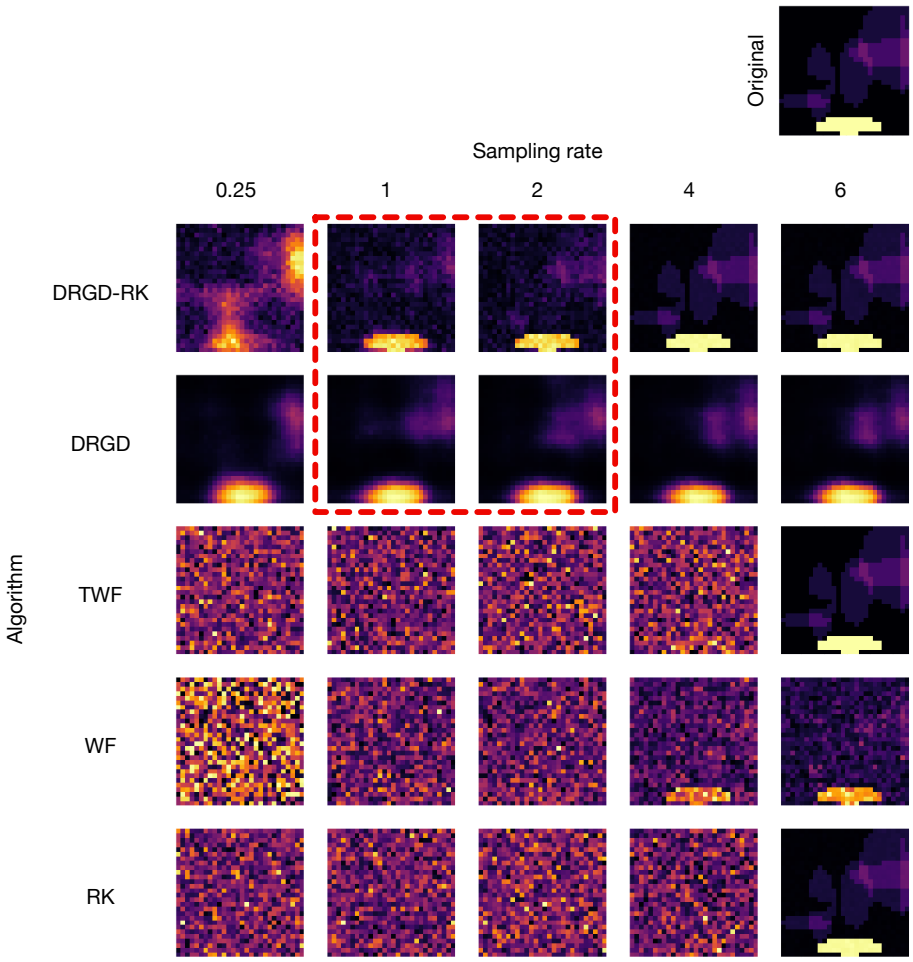


Figure 5.8: Results of the reconstruction process for a selected Shepp-Logan test image for all algorithms and selected sampling rates. *DRGD* and *DRGD-RK* make use of a trained variational autoencoder. One can see that *DRGD* is unable to perform a good reconstruction due to an inability of the variational autoencoder to model the original signal distribution well enough. Important results are highlighted with a dashed red box.

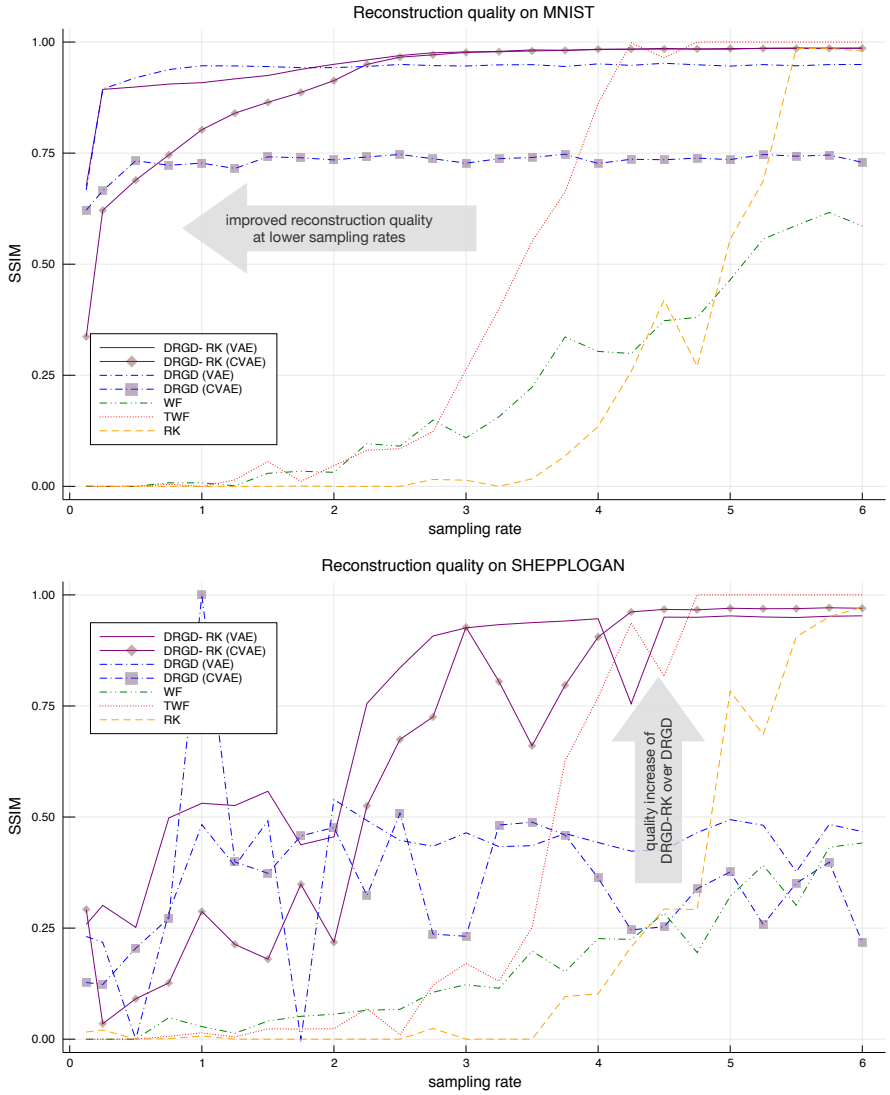


Figure 5.9: Evaluation results for the MNIST and Shepp-Logan datasets with respect to the *structural similarity index for image quality* SSIM. All results are averaged over five different images. The term in brackets (*VAE/CVAE*) describes the type of generative model used as the data prior for the respective reconstruction method. The reconstruction quality of *DRGD-RK (VAE)*, *DRGD-RK (CVAE)* and *RK* is upper-bounded by the number of iterations of the *Randomized Kaczmarz algorithm*. One can see that *DRGD (VAE)* and *DRGD (CVAE)* fail to deliver competitive reconstruction results for high sampling rates.

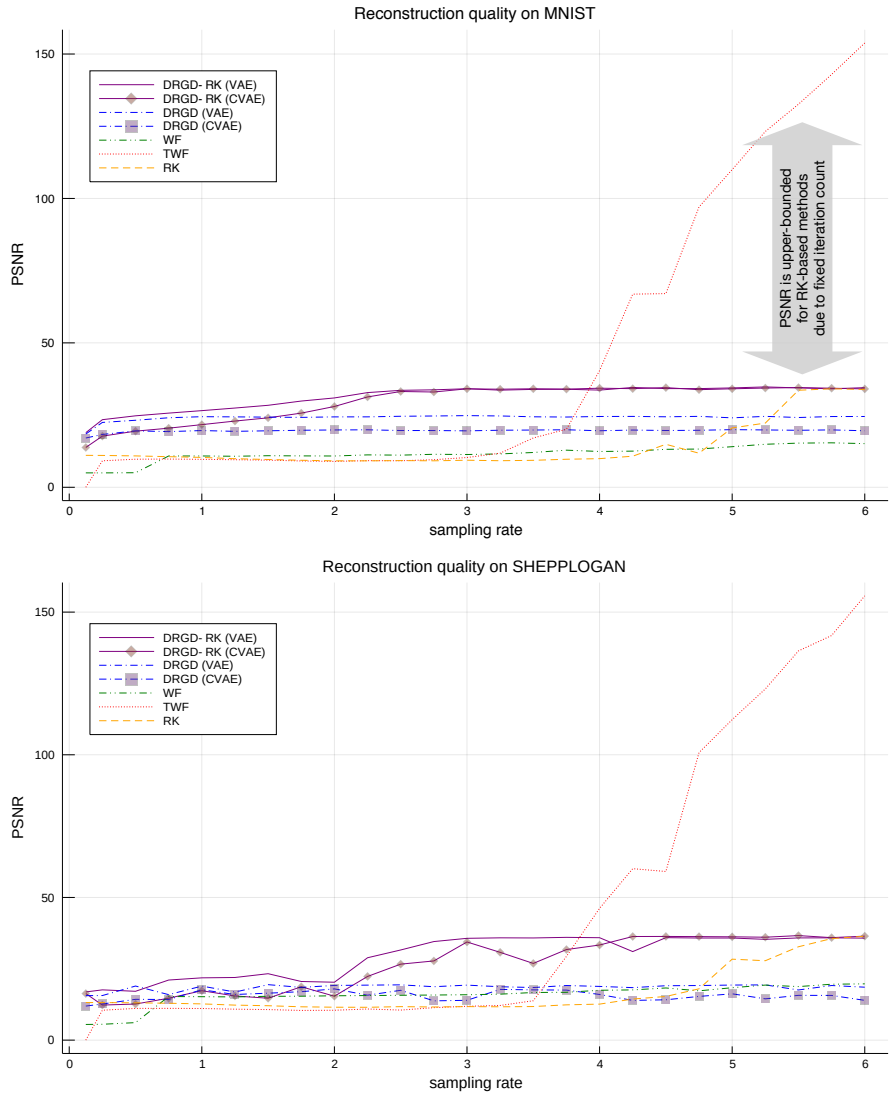


Figure 5.10: Evaluation results for the MNIST and Shepp-Logan datasets with respect to the *peak signal-to-noise ratio* PSNR. All results are averaged over five different images. The term in brackets (*VAE/CVAE*) describes the type of generative model used as the data prior for the respective reconstruction method. Note that the reconstruction quality of *DRGD-RK (VAE)*, *DRGD-RK (CVAE)* and *RK* is upper-bounded by the number of iterations of the *Randomized Kaczmarz algorithm*, which is visible by the quality plateau that all of these algorithms reach.

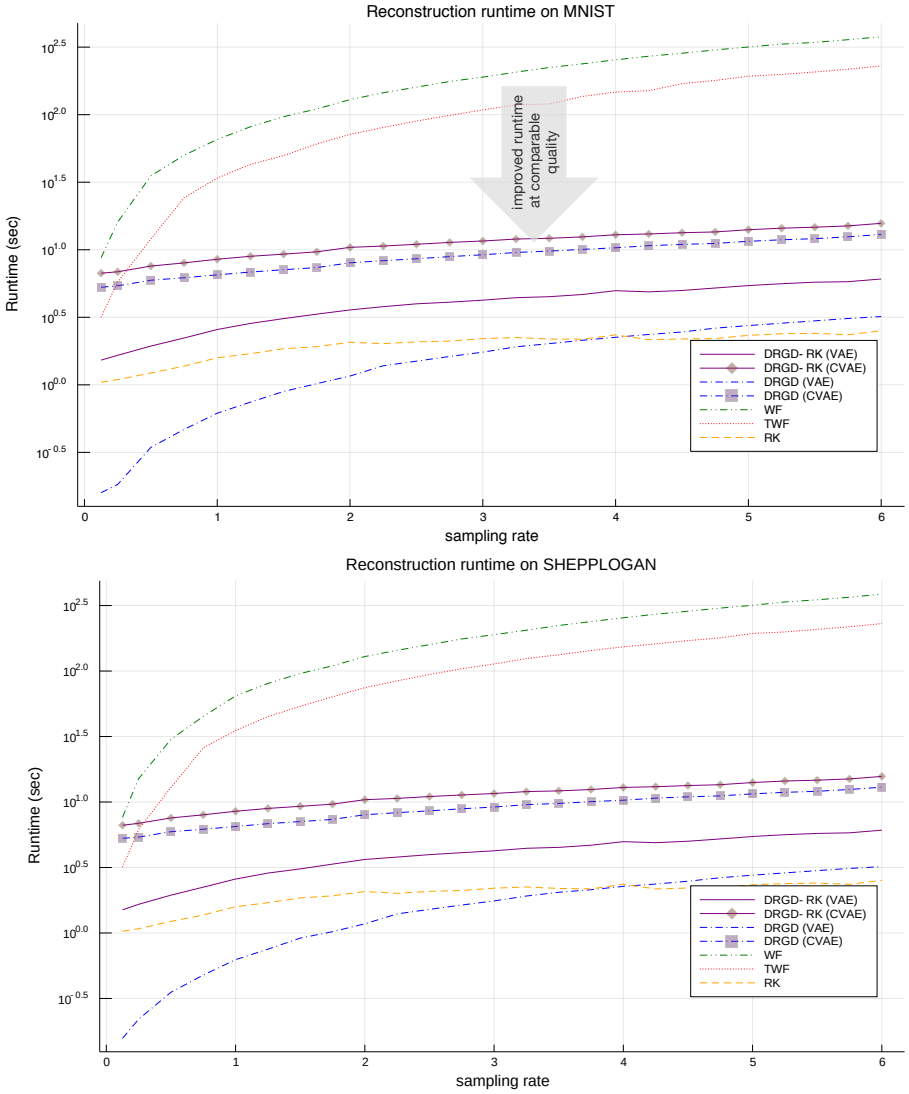


Figure 5.11: Evaluation results for the MNIST and Shepp-Logan datasets with respect to the reconstruction time. Note the log scale of the vertical axis. All results are averaged over five different images. The term in brackets (*VAE/C-VAE*) describes the type of generative model used as the data prior for the respective reconstruction method.

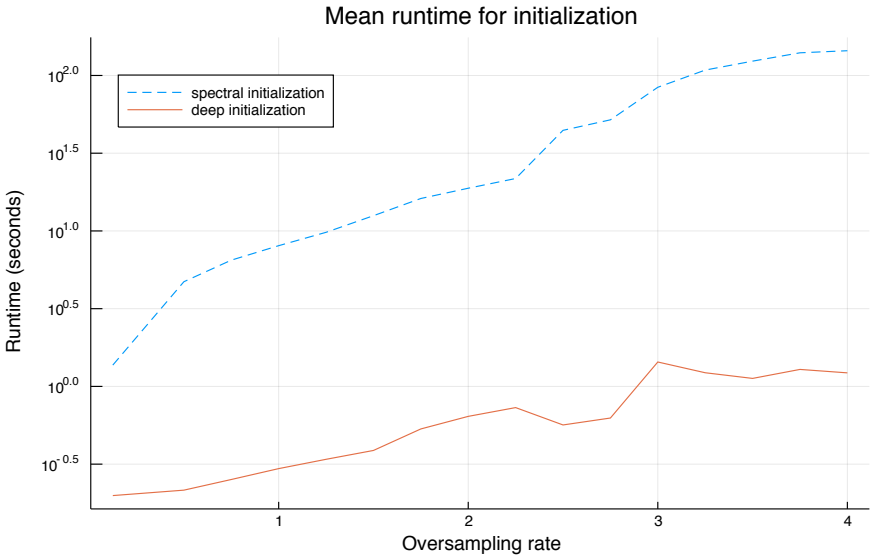


Figure 5.12: Comparison of the runtime of spectral initialization against deep generative initialization. Note the logarithmic scale on the vertical axis. All results are averaged over five different images.

Chapter 6

Deep Generative Models for Terahertz Single-Pixel Phase Retrieval

After having discussed both traditional and deep generative prior-supported reconstruction algorithms for generalized phase retrieval from a theoretical point of view and having numerically validated their performance with random *complex* Gaussian measurements, this chapter builds the bridge to a practical application of the proposed algorithms for a simulated experimental setup.

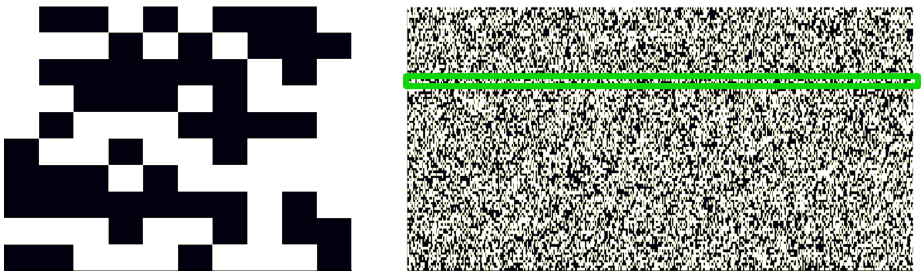
A *single-pixel imaging device* is a system that allows to take an image of a scene using only one single detector (instead of a 2D detector array as is usual in most optical systems, for example in digital cameras). There are many reasons why choosing only a single detector might be advantageous: detectors might be very expensive and building an array of them might not be economically reasonable, or building a detector array might not be technically feasible due to the degree of miniaturization that would be needed for a practical application.

In the optical realm, such a system is called a *single-pixel camera*, and has gained a lot of interest in research and practical applications in the last decade after work done by Baraniuk et al. showed that such a system is possible using the methods of *compressed sensing* (as introduced in Chapter 4.1.1) (Baraniuk, 2007) (Duarte et al., 2008).

Terahertz Single-Pixel Imaging Device

In this chapter we will look at a similar system in the *terahertz* regime. Terahertz radiation is of special interest in many different applications, most notably security, as it is non-ionizing and at the same time able to penetrate many non-conducting materials. This suggests its usage for imaging and motivates the following setup.

We are interested in reconstructing the transmission of a scene illuminated with terahertz radiation (at a wavelength of 0.000856m, which equals to approximately 0.35 THz). However, since we restrict ourselves to only have a single detector cell, we illuminate the scene with a *random* but *known* radiation pattern and collect the transmission radiation through a *collecting optics*



(a) (b)

Figure 6.1: Bernoulli random masks represented as row vectors in a measurement matrix.

(e.g. a lens) which focuses the transmitted radiation into a single detector cell that is able to measure the *intensity* (i.e. squared amplitude) of the incoming radiation. This process is repeated with multiple different patterns to obtain multiple measurements, which are then used to reconstruct the original signal (this is also often referred to as a *structured illumination* approach). This follows the system setup from Augustin et al. (Augustin et al., 2017).

The random radiation patterns are achieved by the usage of so-called *masks* applied by a *spatial light modulator* (also called *optical switch*), a special device that modulates a radiation beam in a way that it only allows radiation to pass through at certain selectable areas (which are defined by the masks). The masks are 2-dimensional patterns which can be discretely represented as matrices $\{0,1\}^{n_1 \times n_2}$ (see Figure 6.1(a)). These 2-dimensional matrices $\{0,1\}^{n_1 \times n_2}$ can also be represented as vectors $\{0,1\}^{n_1 \cdot n_2}$ (where $n_1 \cdot n_2 = n$, see Figure 6.1(b)). We will call these vectors the *real-valued* measurement vectors $\mathbf{a}_i \in \{0,1\}^n$.

Figure 6.2 shows a schematic view of the experimental setup.

It is, however, of importance to note that every electromagnetic wave is subject to diffraction effects when propagating through space after hitting an obstacle or propagating through an aperture (see Figure 6.3) in the size similar to its wavelength. In the *optical* regime (e.g. using a standard optical camera) these diffraction effects are negligible for many practical applications due to the extremely short wavelength of the visible light (between approximately 380nm and 740nm). However, in our setup, we will assume our target of interest to be of a size that is similar to the wavelength, especially will we assume that the *pixel size* of the image of the scene that we want to recover is approximately the wavelength of the radiation.

Technically, this means that we will have to model the diffraction effects taking place between the *spatial light modulator* and the scene and between the scene and the detector. Diffraction effects between the terahertz source and the spatial light modulator can be neglected as the radiation can be seen as coherent.

We will model the diffraction effects using the *Discrete Diffraction Transformations* introduced by Katkovnik et al. (Katkovnik et al., 2009) (Katkovnik et al., 2008) similarly to how it has been used by Nickel (Nickel, 2018), which

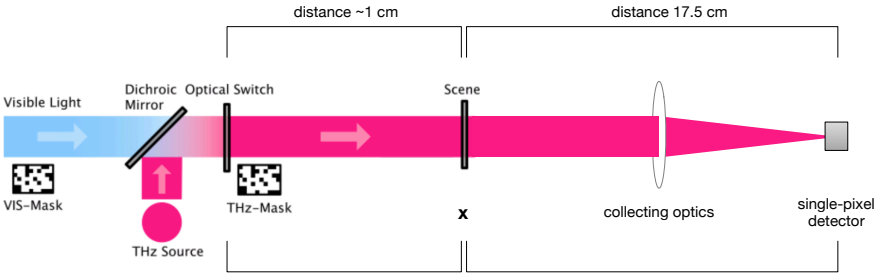


Figure 6.2: Schematic view of the experimental setup. The optical switch is controlled by visible light and allows the transmission of the terahertz radiation at the parts of the mask which are set to `1`, thereby imposing the pattern on the transmitted terahertz radiation. The terahertz pattern is propagated to the scene $\mathbf{x}$ and its transmission is collected using a collecting optics that focuses the radiation onto a singe detector cell. Figure adapted from work by Augustin et al. (Augustin et al., 2017).

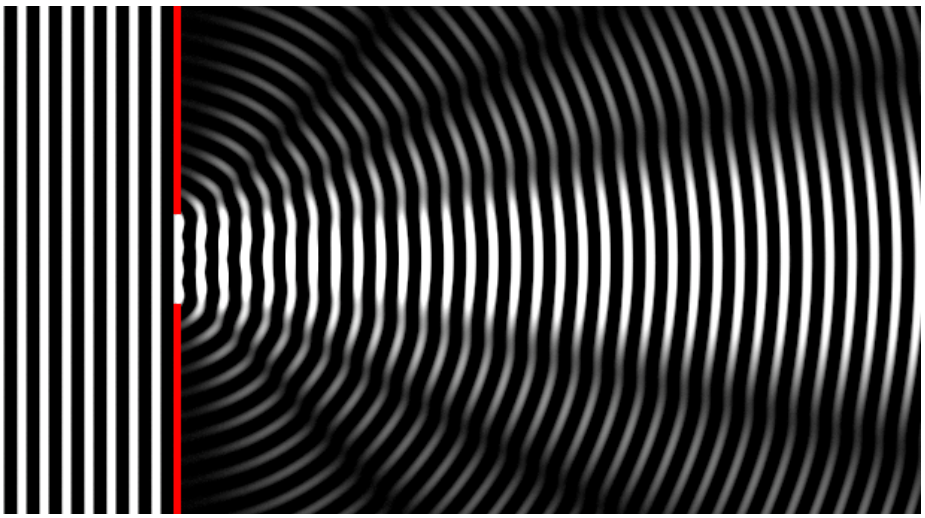


Figure 6.3: Schematic view of the phenomenon of diffraction of a coherent wave coming from the left side passing an aperture of the size of four wavelengths. Adapted from (Lyon, 2010).

will allow us to approximate the diffraction effects using *complex-valued* linear transformations (in the form of complex matrix multiplications) on the incoming wave by specifying the wavelength, propagation distance and pixel sizes at the planes before and after the propagation (see Figure 6.4).

Parameterization of diffraction matrices $D_{\mathrm{M}\to\mathrm{S}}$ and $D_{\mathrm{S}\to\mathrm{D}}$

We will define the diffraction effects *between the spatial light **M**odulator and the **S**cene* using a diffraction matrix $D_{\mathrm{M}\to\mathrm{S}} \in \mathbb{C}^{n\times n}$. This matrix is generated according to the simplified construction method[1] from Katkovnik et al. (Katkovnik et al., 2009) by assuming a propagation distance of 1cm (according to the system setup in Figure 6.2), and 28×28 quadratic pixels of edge size 0.5mm (both before and after the propagation, see Figure 6.4) at a wavelength of $0.856\cdot10^{-3}$m.

We will name the second diffraction matrix, modeling the effects *between the **S**cene and the **D**etector*, $D_{\mathrm{S}\to\mathrm{D}} \in \mathbb{C}^{n\times n}$ and will generate it analogously assuming a propagation distance of 17.5cm.

Signal model

For our simulation, we will model the measurement at the detector as follows: a uniform illumination $[1,\dots,1]^\top$ hits the spatial light modulator which applies the mask $\mathrm{diag}(\mathbf{a}_i)$. After that, the wave propagates from the spatial light modulator to the scene while being subject to diffraction $D_{\mathrm{M}\to\mathrm{S}}$ before it hits the scene $\mathrm{diag}(\mathbf{x})$ and propagates further from the scene to the detector being subject to diffraction $D_{\mathrm{S}\to\mathrm{D}}$. At the detector it is summed up ($\langle\cdot,[1,\dots,1]\rangle$) and its intensity $|\cdot|^2$ is measured.

This leads us to the following (noise-free) signal model:

$$y_i = \left|\sum_{j=1}^{n}\left(D_{\mathrm{S}\to\mathrm{D}}\mathrm{diag}(\mathbf{x})D_{\mathrm{M}\to\mathrm{S}}\mathrm{diag}(\mathbf{a}_i)[1,\dots,1]^\top\right)_j\right|^2 \tag{6.1}$$

$$= \left|\sum_{j=1}^{n}\left(D_{\mathrm{S}\to\mathrm{D}}\mathrm{diag}(\mathbf{x})D_{\mathrm{M}\to\mathrm{S}}\mathbf{a}_i\right)_j\right|^2 \tag{6.2}$$

$$= \left|\sum_{j=1}^{n}\left(D_{\mathrm{S}\to\mathrm{D}}(\mathrm{diag}(\mathbf{a}_i)D^{\mathrm{H}}_{\mathrm{M}\to\mathrm{S}}\mathbf{x})^{\mathrm{H}}\right)_j\right|^2 \tag{6.3}$$

$$= \left|\langle\overline{D_{\mathrm{S}\to\mathrm{D}}(\mathrm{diag}(\mathbf{a}_i)D^{\mathrm{H}}_{\mathrm{M}\to\mathrm{S}}\mathbf{x})^{\mathrm{H}}},[1,\dots,1]^\top\rangle\right|^2 \tag{6.4}$$

$$= \left|\langle\mathrm{diag}(\mathbf{a}_i)D^{\mathrm{H}}_{\mathrm{M}\to\mathrm{S}}\mathbf{x},D^{\top}_{\mathrm{S}\to\mathrm{D}}[1,\dots,1]^\top\rangle\right|^2 \tag{6.5}$$

$$= \left|\langle\mathbf{x},(\mathrm{diag}(\mathbf{a}_i)D^{\mathrm{H}}_{\mathrm{M}\to\mathrm{S}})^{\mathrm{H}}D^{\top}_{\mathrm{S}\to\mathrm{D}}[1,\dots,1]^\top\rangle\right|^2 \tag{6.6}$$

$$= \left|\langle\mathbf{x},D_{\mathrm{M}\to\mathrm{S}}\mathrm{diag}(\mathbf{a}_i)^{\mathrm{H}}D^{\top}_{\mathrm{S}\to\mathrm{D}}[1,\dots,1]^\top\rangle\right|^2 \tag{6.7}$$

$$= \left|\langle\mathbf{x},D_{\mathrm{M}\to\mathrm{S}}\mathrm{diag}(\mathbf{a}_i)D^{\top}_{\mathrm{S}\to\mathrm{D}}[1,\dots,1]^\top\rangle\right|^2 \tag{6.8}$$

[1] The implementation is based on the code accompanying the (Katkovnik et al., 2009) paper available at `http://www.cs.tut.fi/~lasip/DDT/`.

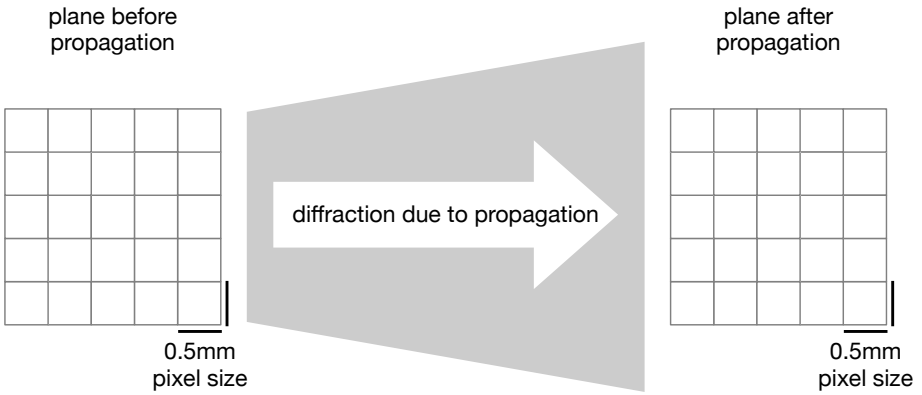


Figure 6.4: Schematic view of the pixel sizes at the planes before and after propagation. For the evaluation we simulate pixel sizes approximately the size of the wavelength both *before* and *after* the propagation.

$$= \left| \langle \overline{D_{\mathrm{M}\to\mathrm{S}} \mathrm{diag}(\mathbf{a}_i) D^{\top}_{\mathrm{S}\to\mathrm{D}}[1,\dots,1]^{\top}}, \mathbf{x} \rangle \right|^2 \tag{6.9}$$

$$= \left| \langle \overline{D_{\mathrm{M}\to\mathrm{S}}} \mathrm{diag}(\mathbf{a}_i) D^{\mathrm{H}}_{\mathrm{S}\to\mathrm{D}}[1,\dots,1]^{\top}, \mathbf{x} \rangle \right|^2 \tag{6.10}$$

$$=: \left| \langle \tilde{\mathbf{a}}_i, \mathbf{x} \rangle \right|^2 \tag{6.11}$$

and finally to the optimization problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \sum_{i=1}^{m} L(y_i, |\langle \tilde{\mathbf{a}}_i, \mathbf{x} \rangle|^2), \tag{6.12}$$

which for a quadratic loss results in

$$\min_{\mathbf{x}\in\mathbb{R}^n} \sum_{i=1}^{m} (y_i - |\langle \tilde{\mathbf{a}}_i, \mathbf{x} \rangle|^2)^2. \tag{6.13}$$

This problem is especially sensitive to changes of the distance between the spatial light modulator and the scene $D_{\mathrm{M}\to\mathrm{S}}$ (also referred to as the *stand-off distance*), because the masks commanded at the spatial light modulator drastically degrade while propagating to the scene (see Figure 6.5). This is caused by the diffraction matrix $D_{\mathrm{M}\to\mathrm{S}}$ losing rank with increasing propagation distance (an effect that is covered in more detail in (Katkovnik et al., 2009)). This results in a blurring effect, which, depending on the distance, can be up to a degree that the original signal can no longer be recovered. We will therefore investigate the reconstruction quality of (6.13) with respect to sensitivity to changes in the distance between the spatial light modulator and the scene for *simulated data*, which will be the topic of the next section. A similar problem has been investigated experimentally by Augustin et al (Augustin et al., 2019).

It is important to note that there are even more difficulties that affect the reconstruction quality in this setup. As an example, the shape of the beam might in fact be almost Gaussian due to the diagonal horn shape of the antenna of the terahertz transmitter (Augustin et al., 2019). Furthermore, radiation

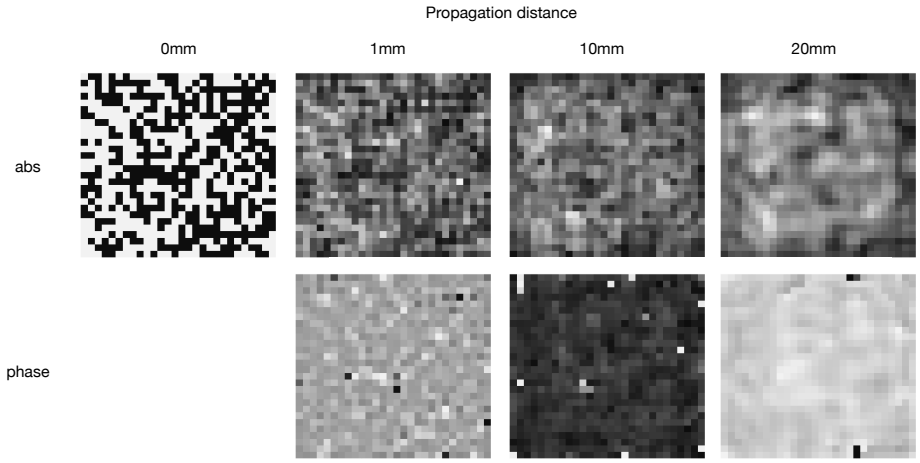


Figure 6.5: Commanded Bernoulli mask (0mm) and degraded masks after 1mm, 10mm and 20mm of simulated free-space propagation.

divergence takes place: the radiation is not completely parallel, which has an influence on the effective masks at the scene. Investigating these effects is out of scope for this thesis. In our simulations we are also not investigating the effect of a change of pixel sizes or the number of pixels for our reconstruction.

6.1 Sensitivity Analysis

We will now investigate the sensitivity of the reconstruction quality for different stand-off distances and sampling rates $\frac{m}{n}$. Our evaluation will be done for both traditional (*Truncated Wirtinger Flow (TWF)*) as well as deep generative prior-supported reconstruction algorithms (*Deep Regularized Gradient Descent* and *Deep Regularized Gradient Descent-initialized Randomized Kaczmarz*, both using the *variational autoencoders* defined in Sections 5.3.1 and 5.3.2 as their underlying generative models) and will be executed on both the MNIST dataset as well as on the synthetic Shepp-Logan dataset. For the *Truncated Wirtinger Flow* algorithm (defined in Section 4.4.2) we use parameters ($a_z^{\mathrm{lb}} = 0.001$, $a_z^{\mathrm{ub}} = 500$) different to the usual defaults.

The following pages contain the results of these numerically simulated experiments for stand-off distances between 0.00125m and 0.08m.

Figures 6.6 and 6.7 visually show the results of the reconstruction process for selected MNIST and Shepp-Logan samples.

Figures 6.8 to 6.13 show the reconstruction quality of the different algorithms for varying stand-off distances and sampling rates with respect to the *structural similarity index* SSIM and the *peak signal-to-noise ratio* PSNR.

Main findings and observations

The most important result from this experiment is the observation that *Deep Regularized Gradient Descent-initialized Randomized Kaczmarz* has a reconstruction quality which is superior to *Truncated Wirtinger Flow* for virtually all

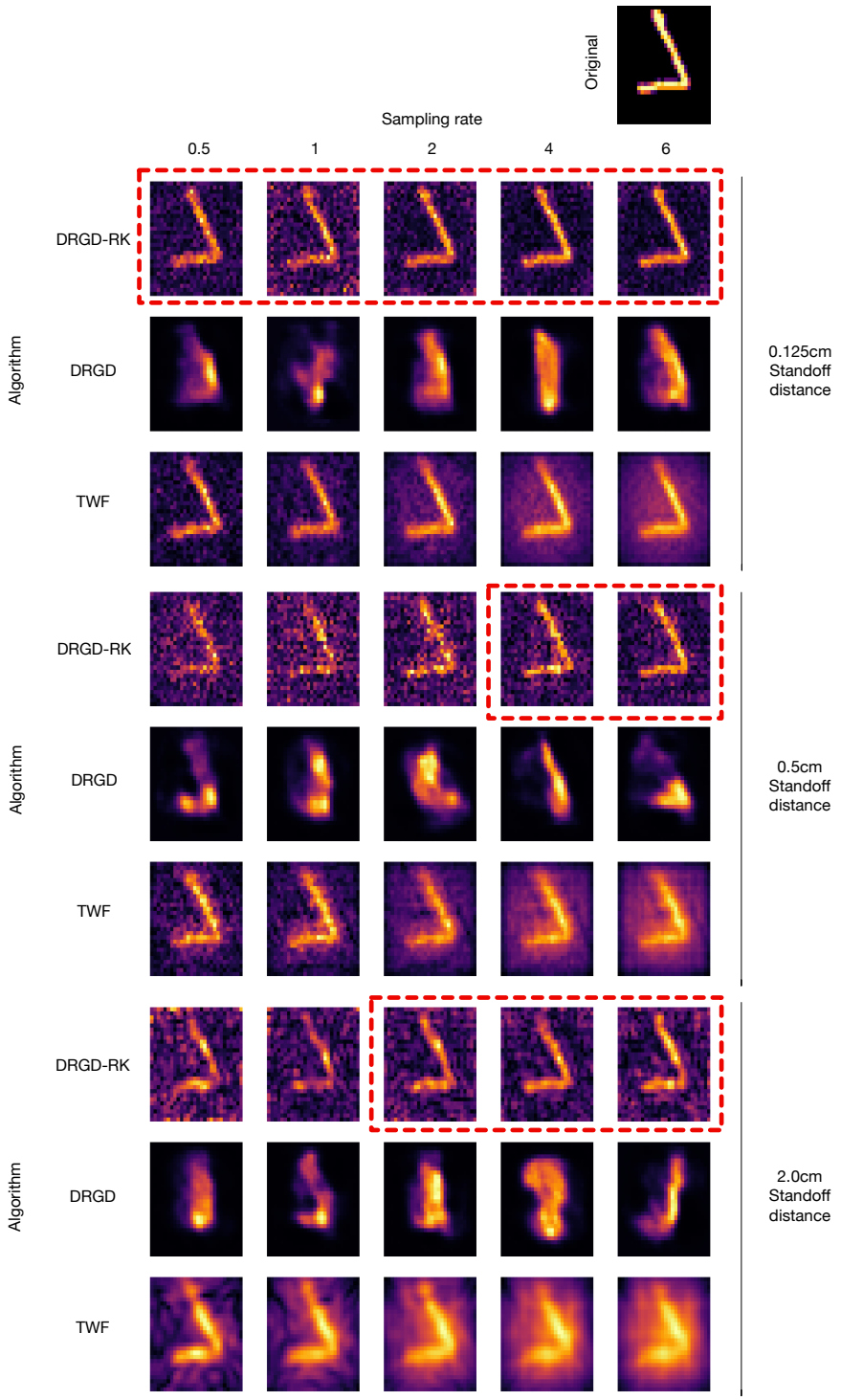


Figure 6.6: Results of the reconstruction process for a selected MNIST test image for selected sampling rates at 0.125cm, 0.5cm and 2cm stand-off distances. Important results are highlighted with a dashed red box.

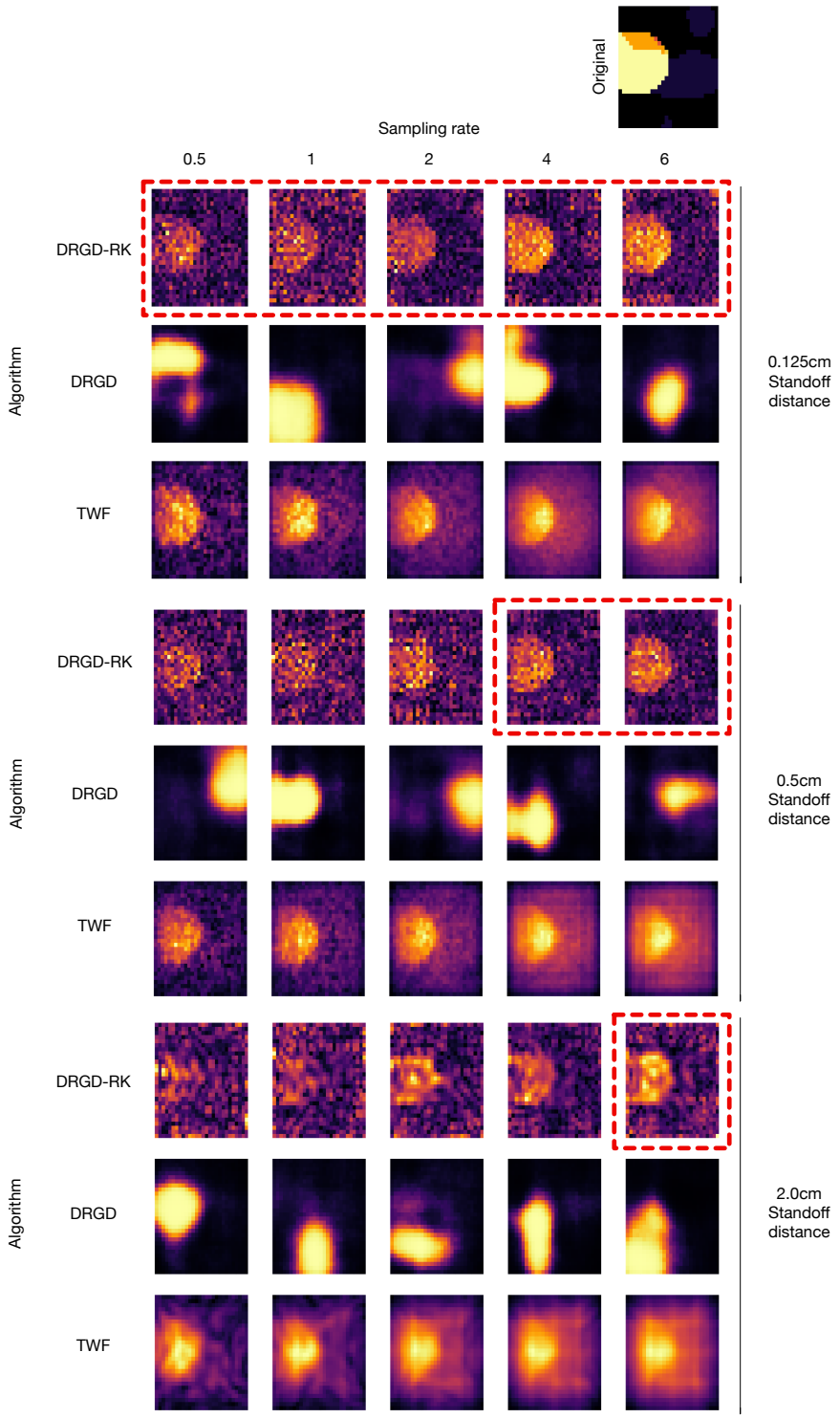


Figure 6.7: Results of the reconstruction process for a sample from the Shepp-Logan dataset for selected sampling rates at 0.125cm, 0.5cm and 2cm stand-off distances. Important results are highlighted with a dashed red box.

evaluated sampling rates and stand-off distances. This is especially visible in the evaluation results with respect to the *structural similarity index* SSIM (see Figures 6.8 and 6.12), while the *peak signal-to-noise ratio* PSNR (see Figures 6.9 and 6.13) shows less pronounced differences between the two methods. These quantitative results are in line with the qualitative results provided by visual comparison in Figures 6.6 and 6.7.

One can also see that the reconstruction quality decreases when increasing the stand-off distance $D_{\mathrm{M}\to\mathrm{S}}$. This is an expected effect which is caused by the diffraction matrix losing rank with increasing propagation distance (Katkovnik et al., 2009).

Deep Regularized Gradient Descent is unable to perform meaningful reconstructions for almost all sampling rates and stand-off distances. This is attributable to the variational autoencoder being unable to model the signal distribution well enough.

One can also observe that *Truncated Wirtinger Flow* shows slightly declining reconstruction quality with increased sampling rate. This is due to an effect caused by the spectral initializer used in this method.

The evaluation results experimentally confirm that *Deep Regularized Gradient Descent-initialized Randomized Kaczmarz* is able to outperform *Truncated Wirtinger Flow* for generalized phase retrieval in simulations of physical systems where diffraction effects occur. This suggests that *Deep Regularized Gradient Descent-initialized Randomized Kaczmarz* can prove valuable as a generalized phase retrieval algorithm in real-world physical image reconstruction problems.

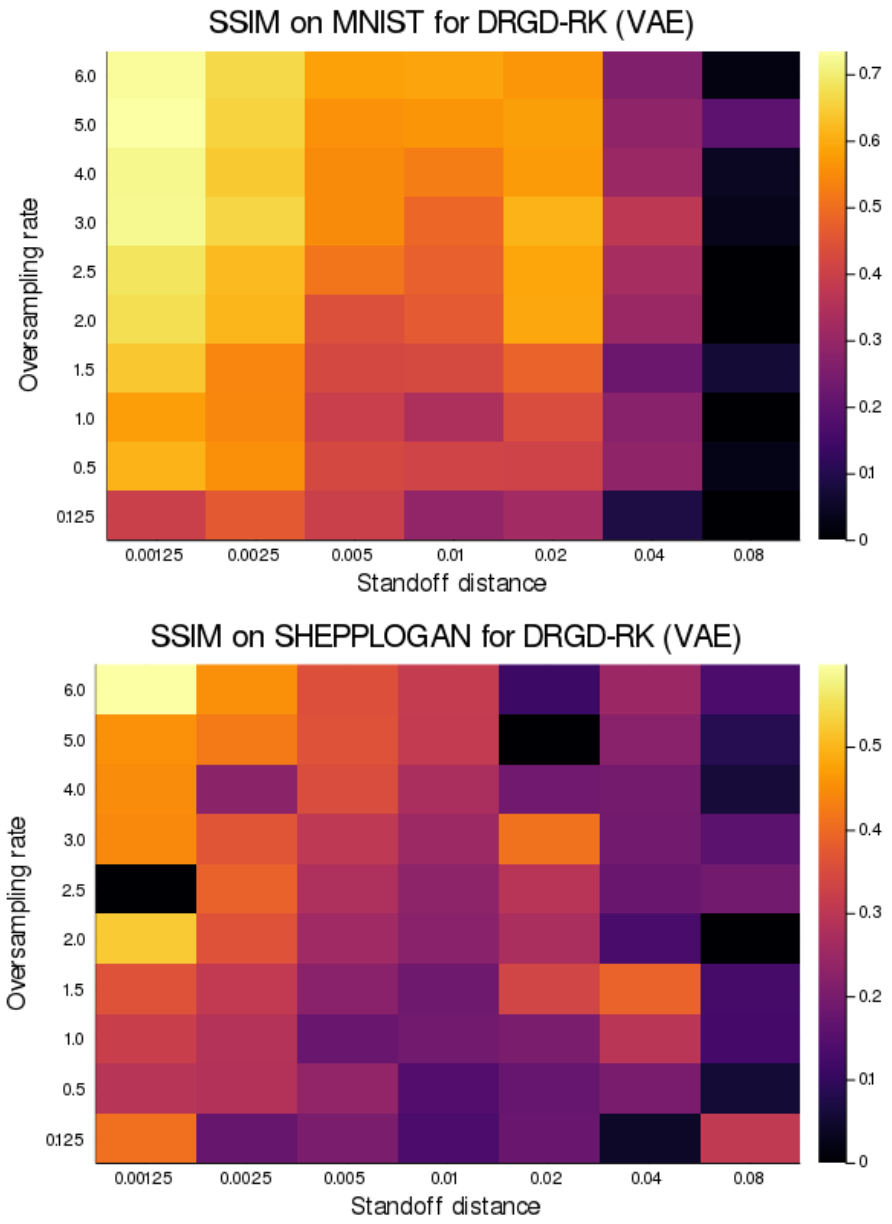


Figure 6.8: Evaluation results of the practical experiment for the *Deep Regularized Gradient Descent-initialized Randomized Kaczmarz* method (using a variational autoencoder as generator) on the MNIST and Shepp-Logan datasets with respect to reconstruction quality measured using the *structural similarity index* SSIM. All results are averaged over five different images. Stand-off distance in meters.

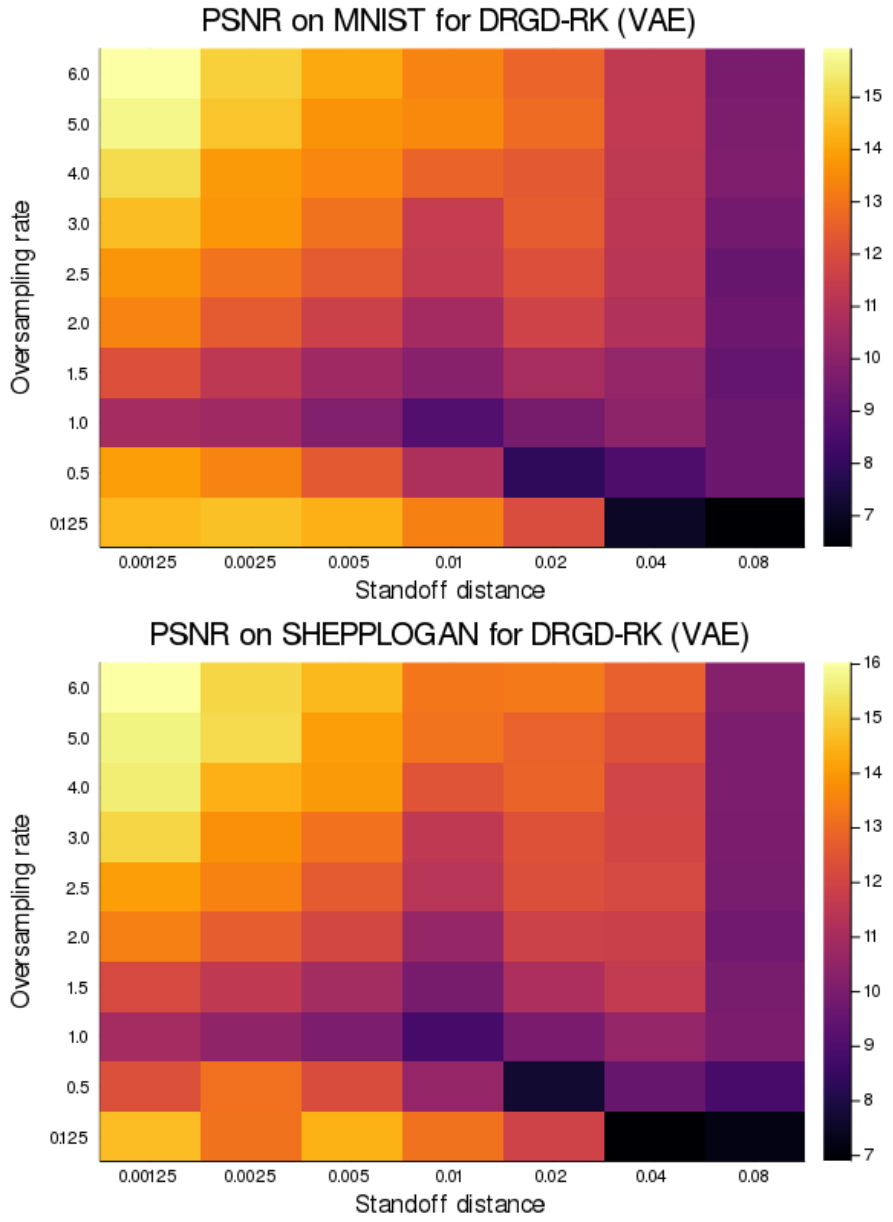


Figure 6.9: Evaluation results of the practical experiment for the *Deep Regularized Gradient Descent-initialized Randomized Kaczmarz* method (using a variational autoencoder as generator) on the MNIST and Shepp-Logan datasets with respect to reconstruction quality measured using the *peak signal-to-noise ratio* PSNR. All results are averaged over five different images. Stand-off distance in meters.

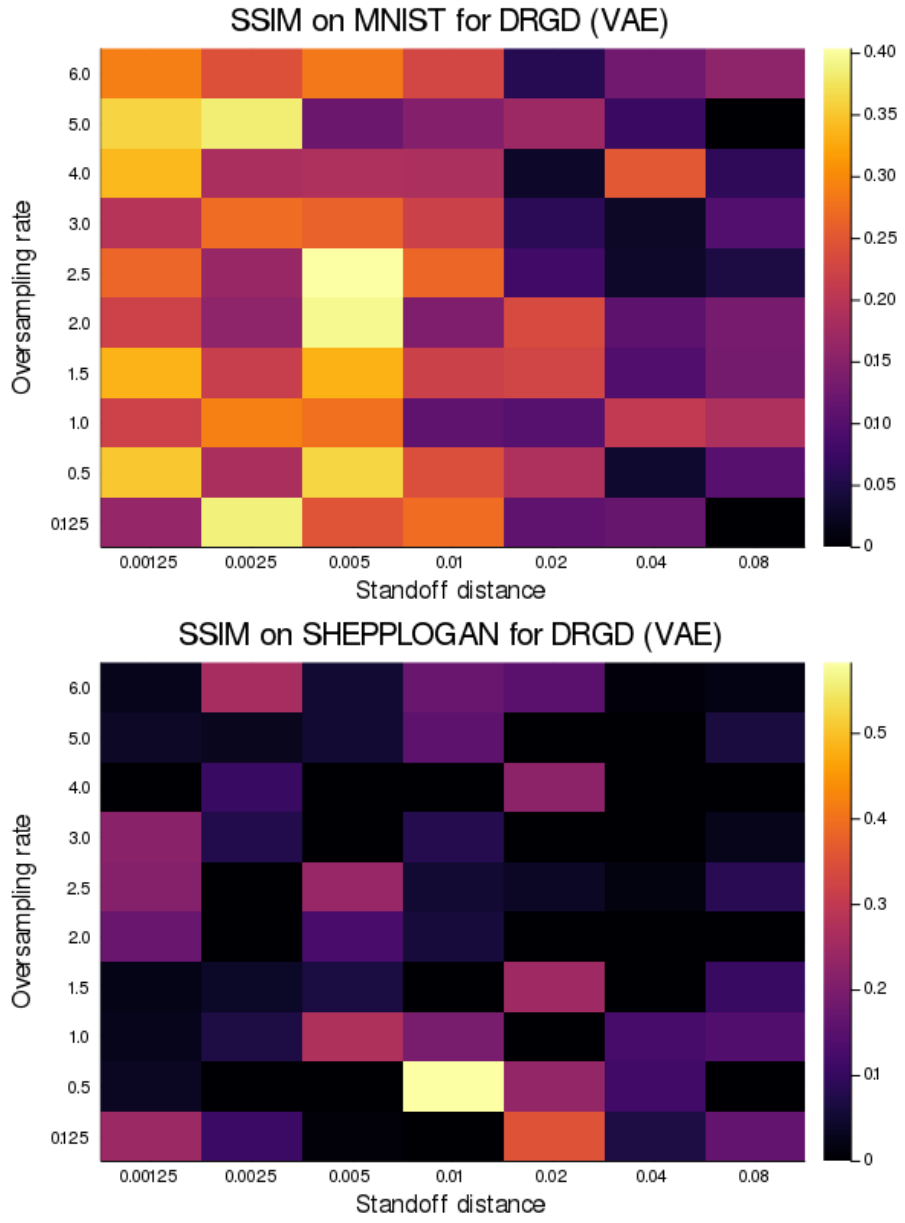


Figure 6.10: Evaluation results of the practical experiment for the *Deep Regularized Gradient Descent* method (using a variational autoencoder as generator) on the MNIST and Shepp-Logan datasets with respect to reconstruction quality measured using the *structural similarity index* SSIM. All results are averaged over five different images. Stand-off distance in meters.

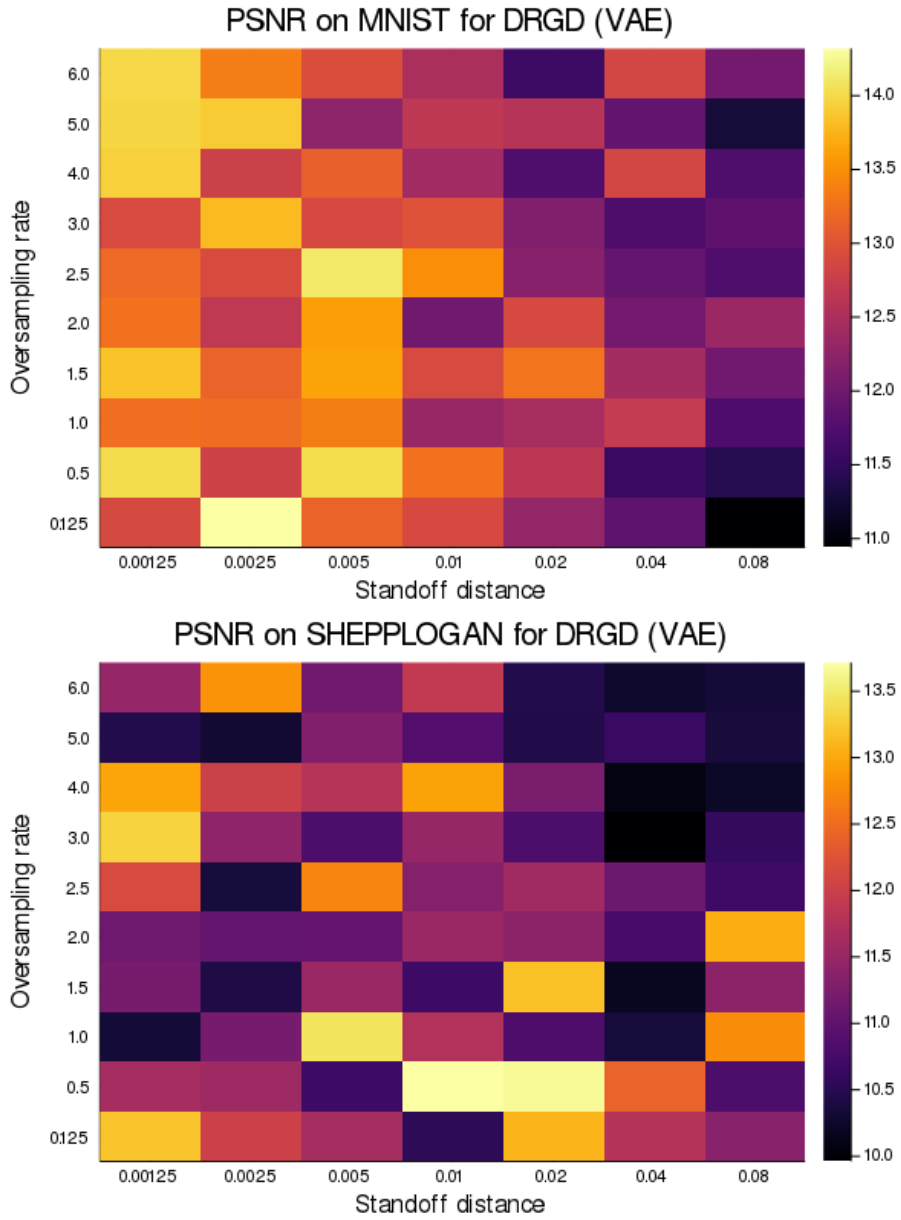


Figure 6.11: Evaluation results of the practical experiment for the *Deep Regularized Gradient Descent* method (using a variational autoencoder as generator) on the MNIST and Shepp-Logan datasets with respect to reconstruction quality measured using the *peak signal-to-noise ratio* PSNR. All results are averaged over five different images. Stand-off distance in meters.

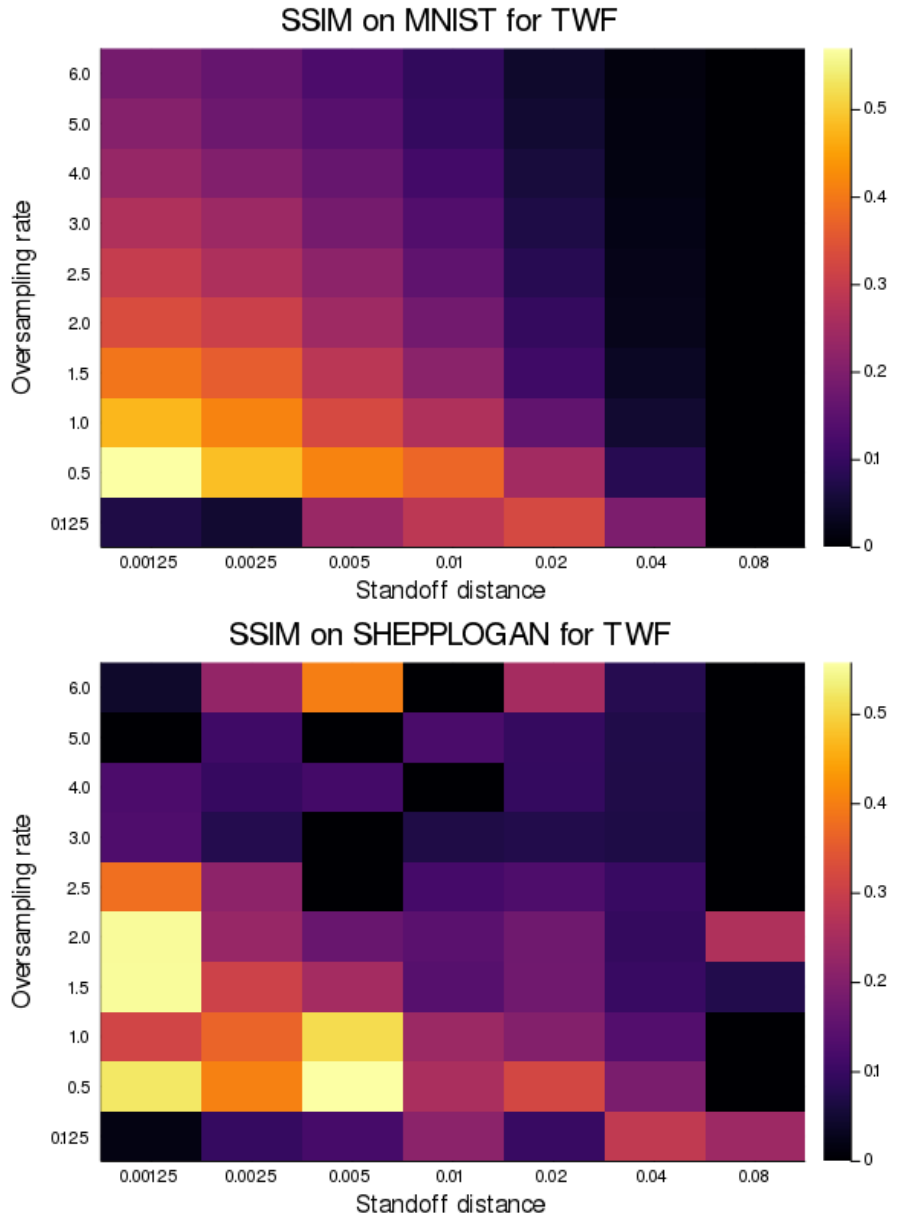


Figure 6.12: Evaluation results of the practical experiment for the *Truncated Wirtinger Flow* method on the MNIST and Shepp-Logan datasets with respect to reconstruction quality measured using the *structural similarity index* SSIM. All results are averaged over five different images. Stand-off distance in meters.

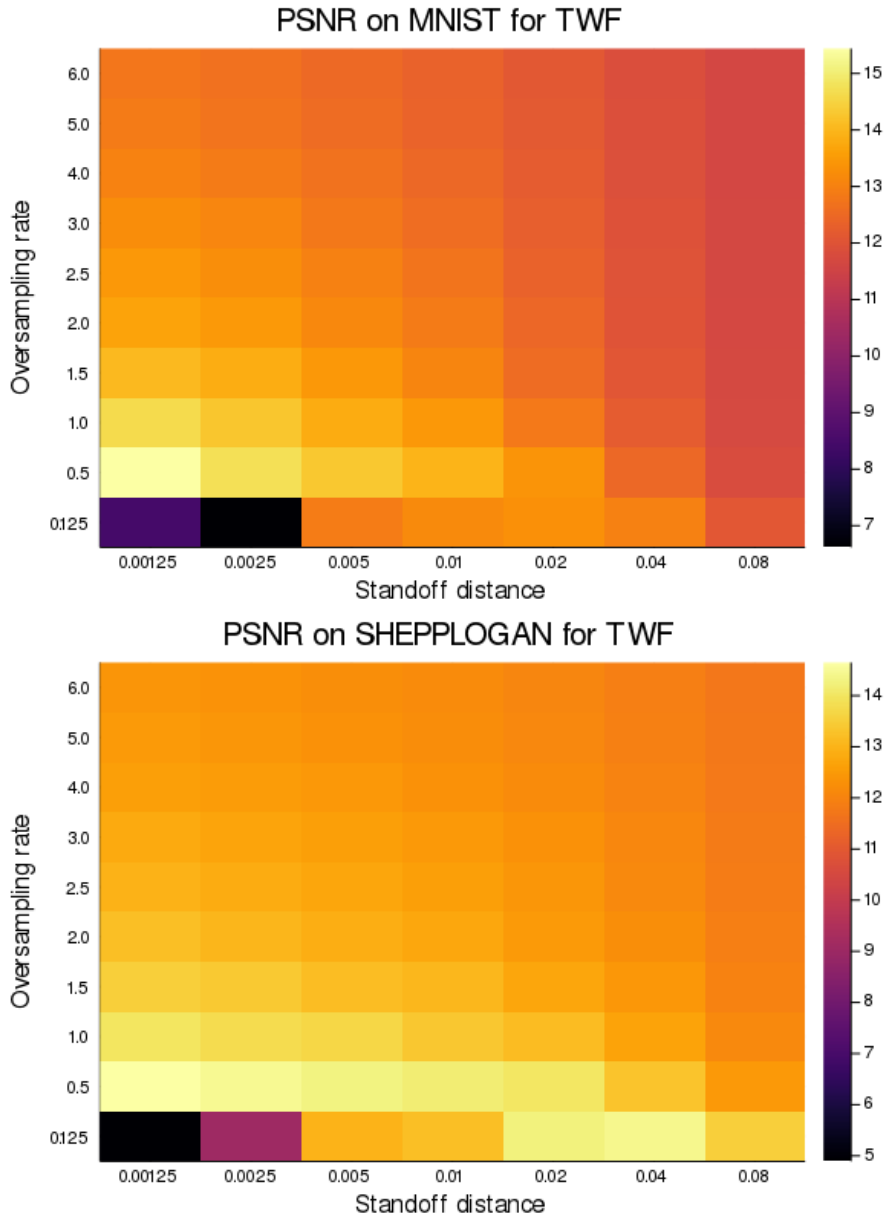


Figure 6.13: Evaluation results of the practical experiment for the *Truncated Wirtinger Flow* method on the MNIST and Shepp-Logan datasets with respect to reconstruction quality measured using the *peak signal-to-noise ratio* PSNR. All results are averaged over five different images. Stand-off distance in meters.

Chapter 7

Conclusion

This thesis explored the applicability of deep learning in the field of nonlinear inverse problems. Special focus was given to the question how deep generative models (such as the generator part of a *variational autoencoder*) can be leveraged to support the solution of the *generalized phase retrieval* problem, a very prominent example of a nonlinear inverse problem, which has applications in a lot of disciplines.

We introduced both traditional methods of solving the generalized phase retrieval problem as well as already existing and new approaches to incorporate deep generative models into the reconstruction process. We evaluated the reconstruction quality and runtime of these algorithms on the famous *MNIST* dataset as well as on a more complex synthetic dataset based on the well-known *Shepp-Logan phantom* and we were able to show that deep generative prior-based reconstruction methods are often able to reconstruct images better than traditional methods while requiring fewer measurements. We were also able to show that generator model error plays a significant role in the reconstruction quality of deep generative prior-based methods, resulting in poor reconstruction quality if the deep model has not been trained well or is not expressive enough to capture the full complexity of the signal domain.

The thesis introduced two new algorithms: *Deep Regularized Gradient Descent* is a total variation-regularized extension of the deep generative prior-based gradient descent algorithm introduced by Shamshad and Ahmed (Shamshad and Ahmed, 2018). This method allows to find a solution to the generalized phase retrieval problem in the range of a differentiable generator network, but suffers from poor reconstruction quality when the generator network is not properly able to model the signal domain. *Deep Regularized Gradient Descent-initialized Randomized Kaczmarz* also incorporates signal domain information using deep generative priors but does not suffer from reconstruction quality degradation caused by generator model error. This is because the data prior is used only during the initialization (performed by executing the *Deep Regularized Gradient Descent* part of the method) while the actual reconstruction is performed using *Randomized Kaczmarz* iterations. The thesis empirically showed that *Deep Regularized Gradient Descent-initialized Randomized Kaczmarz* achieves a reconstruction quality which is higher than the ones provided by traditional methods at low sampling rates (which is a result of the generative prior-based optimization used as the initializer). The method furthermore achieves a recon-

struction quality comparable to traditional methods (*Truncated Wirtinger Flow*, *Wirtinger Flow* and *Randomized Kaczmarz*) at higher sampling rates (achieved by the well-initialized *Randomized Kaczmarz* part), all while having superior runtime performance.

For the practically motivated application of reconstructing an image from measurements performed by a simulated *terahertz single-pixel imaging device* in Chapter 6, the thesis experimentally showed that the *Deep Regularized Gradient Descent-initialized Randomized Kaczmarz* method achieves reconstruction quality that is superior to *Truncated Wirtinger Flow*, which gives first experimental evidence that the method is suited for real-world image reconstruction scenarios in which degradation of the scene caused by diffraction plays an important role.

Appendix A

Code Listings

A.1 Randomized Shepp-Logan-style phantoms

This is the Julia[1] 1.1 code for the generation of samples for the synthetic Shepp-Logan dataset used in sections 5.3.2 and 5.6.

```
function shepp_logan(M,N)
  # This code is based on https://raw.githubusercontent.com/JuliaImages/
  # Images.jl/dd15028375f682affb61e64293a2b519f7225203/src/algorithms.jl

  P = zeros(M,N)
  x = range(-1, stop=1, length=M)'
  y = range(1, stop=-1, length=N)

  centerX = rand(10) .*2 .-1
  centerY = rand(10) .*2 .-1
  majorAxis = rand(10).*0.9
  minorAxis = rand(10).*0.9
  theta = rand(10)

  grayLevel = [2, -0.98, -0.02, -0.02, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01]

  for l=1:length(theta)
    P += grayLevel[l] * (
        ((cos(theta[l] / 360*2*π) * (x .- centerX[l]) .+
          sin(theta[l] / 360*2*π) * (y .- centerY[l])) / majorAxis[l] ).^2 .+
        ((sin(theta[l] / 360*2*π) * (x .- centerX[l]) .-
          cos(theta[l] / 360*2*π) * (y .- centerY[l])) / minorAxis[l] ).^2 .< 1)
  end

  return P
end
```

[1] https://julialang.org/downloads/

A.2 Deep Regularized Gradient Descent

This is the Julia 1.1 code for the deep generative model-based gradient descent algorithm with discrete anisotropic total variation norm regularization (*Deep Regularized Gradient Descent*).

```
using Distributions
using Flux

# grad(f, z) is a function that takes the gradient of f at z
# using algorithmic differentiation, e.g. Tracker.gradient from the Flux library
grad(f, z) = Tracker.gradient(f, z)[1]

function drgd(generator,
              generator_dimension,
              forward_model,
              y;
              eta=0.1,
              iters=50,
              lambda=0.1,
              image_shape=(28,28))

    z = rand(Normal(0,1), generator_dimension)

    objective_function = (x) -> (norm(y - forward_model(generator(x))) +
            lambda*(sum(abs.(diff(reshape(generator(x), image_shape), dims=1))) +
            sum(abs.(diff(reshape(generator(x), image_shape), dims=2)))))

    for i in 1:iters
        gradient = grad(objective_function, z)
        z = z - gradient .* eta
    end

    return generator(z)
end
```

A.3 Deep Regularized Gradient Descent-initialized Randomized Kaczmarz

This is the Julia 1.1 code for the randomized Kaczmarz method using the deep generative model-based gradient descent algorithm with discrete anisotropic total variation norm regularization as an initializer (*Deep Regularized Gradient Descent-initialized Randomized Kaczmarz*).

```
using Distributions
using Flux

# grad(f,z) is function that takes the gradient of f at z
# using algorithmic differentiation, e.g. Tracker.gradient from the Flux library
grad(f, z) = Tracker.gradient(f, z)[1]

function drgd_rk(generator,
                 generator_dimension,
                 forward_model,
                 A,
                 y;
                 eta_initializer = 0.1,
                 iters_initializer = 50,
                 lambda_initializer = 0.1,
                 iters_kaczmarz = 1000000,
                 image_shape = (28,28))

    z = rand(Normal(0,1), generator_dimension)

    objective_function = (x) -> (norm(y - forward_model(generator(x))) +
        lambda_initializer * (sum(abs.(diff(reshape(generator(x), image_shape),
        dims=1))) + sum(abs.(diff(reshape(generator(x), image_shape), dims=2)))))

    for i in 1:iters_initializer
        gradient = grad(objective_function, z)
        z = z - gradient .* eta_initializer
    end

    x_approx = generator(z)

    m,n = size(A)
    x = x_approx * maximum([norm(A[i,:],2) for i in 1:size(A)[1]])
    for i in 1:iters_kaczmarz
        rk = rand(1:size(A)[1])
        a  = A[rk,:] / norm(A[rk,:],2)
        η  = sign(dot(a, x)) * y[rk] - dot(a, x)
        x += η * a
    end

    return abs.(x / maximum([norm(A[i,:],2) for i in 1:size(A)[1]]))
end
```

Bibliography

M. Asim, A. Ahmed, and P. Hand. Invertible generative models for inverse problems: mitigating representation error and dataset bias. *arXiv:1905.11672 [cs]*, May 2019. URL http://arxiv.org/abs/1905.11672. arXiv: 1905.11672.

S. Augustin, S. Frohmann, P. Jung, and H.-W. Hubers. An optically controllable 0.35 THz single-pixel camera for millimeter resolution imaging. In *2017 42nd International Conference on Infrared, Millimeter, and Terahertz Waves (IRMMW-THz)*, pages 1–2, Cancun, Mexico, Aug. 2017. IEEE. ISBN 978-1-5090-6050-4. doi: 10.1109/IRMMW-THz.2017.8066996. URL http://ieeexplore.ieee.org/document/8066996/.

S. Augustin, P. Jung, S. Frohmann, and H.-W. Huebers. Terahertz dynamic aperture imaging at stand-off distances using a Compressed Sensing protocol. *arXiv:1902.07935 [physics]*, Feb. 2019. URL http://arxiv.org/abs/1902.07935. arXiv: 1902.07935.

A. S. Bandeira, J. Cahill, D. G. Mixon, and A. A. Nelson. Saving phase: Injectivity and stability for phase retrieval. *Applied and Computational Harmonic Analysis*, 37(1):106–125, July 2014. ISSN 10635203. doi: 10.1016/j.acha.2013.10.002. URL https://linkinghub.elsevier.com/retrieve/pii/S1063520313000936.

R. Baraniuk, M. Davenport, R. DeVore, and M. Wakin. A Simple Proof of the Restricted Isometry Property for Random Matrices. *Constructive Approximation*, 28(3):253–263, Dec. 2008. ISSN 0176-4276, 1432-0940. doi: 10.1007/s00365-007-9003-x. URL http://link.springer.com/10.1007/s00365-007-9003-x.

R. G. Baraniuk. Compressive sensing. *IEEE signal processing magazine*, 24(4), 2007. doi: 10.1109/MSP.2007.4286571. URL https://ieeexplore.ieee.org/document/4286571.

A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *arXiv:1502.05767 [cs, stat]*, Feb. 2015. URL http://arxiv.org/abs/1502.05767. arXiv: 1502.05767.

D. P. Bertsekas. *Nonlinear programming*. Athena Scientific, Belmont, Massachusetts, third edition, 2016. ISBN 978-1-886529-05-2. OCLC: 988741359.

A. Bora, A. Jalal, E. Price, and A. G. Dimakis. Compressed Sensing using Generative Models. *arXiv:1703.03208 [cs, math, stat]*, Mar. 2017. URL http://arxiv.org/abs/1703.03208. arXiv: 1703.03208.

P. Bouboulis. Wirtinger's Calculus in general Hilbert Spaces. *arXiv:1005.5170 [cs, math]*, May 2010. URL `http://arxiv.org/abs/1005.5170`. arXiv: 1005.5170.

S. Boyd. Subgradient Methods. Notes for EE364b, Stanford University, Spring 2013–14, May 2014. URL `http://web.mit.edu/6.976/www/notes/subgrad_method.pdf`.

S. P. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, Cambridge, UK; New York, 2004. ISBN 978-0-521-83378-3.

E. Candes and T. Tao. Decoding by linear programming. *arXiv preprint math/0502327*, 2005.

E. J. Candes and T. Tao. Near-Optimal Signal Recovery From Random Projections: Universal Encoding Strategies? *IEEE Transactions on Information Theory*, 52(12):5406–5425, Dec. 2006. ISSN 0018-9448. doi: 10.1109/TIT.2006.885507. URL `http://ieeexplore.ieee.org/document/4016283/`.

E. J. Candes, T. Strohmer, and V. Voroninski. PhaseLift: Exact and Stable Signal Recovery from Magnitude Measurements via Convex Programming. *arXiv:1109.4499 [cs, math]*, Sept. 2011. URL `http://arxiv.org/abs/1109.4499`. arXiv: 1109.4499.

E. J. Candes, X. Li, and M. Soltanolkotabi. Phase Retrieval via Wirtinger Flow: Theory and Algorithms. *IEEE Transactions on Information Theory*, 61(4):1985–2007, Apr. 2015. ISSN 0018-9448, 1557-9654. doi: 10.1109/TIT.2015.2399924. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7029630`.

A. Chambolle, V. Caselles, M. Novaga, D. Cremers, and T. Pock. An introduction to Total Variation for Image Analysis. working paper, Nov. 2009. URL `https://hal.archives-ouvertes.fr/hal-00437581`.

S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic Decomposition by Basis Pursuit. *SIAM Journal on Scientific Computing*, 20(1):33–61, Jan. 1998. ISSN 1064-8275, 1095-7197. doi: 10.1137/S1064827596304010. URL `http://epubs.siam.org/doi/10.1137/S1064827596304010`.

Y. Chen and E. J. Candes. Solving Random Quadratic Systems of Equations Is Nearly as Easy as Solving Linear Systems. *arXiv:1505.05114 [cs, math, stat]*, May 2015. URL `http://arxiv.org/abs/1505.05114`. arXiv: 1505.05114.

Y. Chen, Y. Chi, J. Fan, and C. Ma. Gradient Descent with Random Initialization: Fast Global Convergence for Nonconvex Phase Retrieval. *Mathematical Programming*, 176(1-2):5–37, July 2019. ISSN 0025-5610, 1436-4646. doi: 10.1007/s10107-019-01363-6. URL `http://arxiv.org/abs/1803.07726`. arXiv: 1803.07726.

L. Condat. Discrete Total Variation: New Definition and Minimization. *SIAM Journal on Imaging Sciences*, 10(3):1258–1290, Jan. 2017. ISSN 1936-4954. doi: 10.1137/16M1075247. URL `https://epubs.siam.org/doi/10.1137/16M1075247`.

D. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, Apr. 2006. ISSN 0018-9448. doi: 10.1109/TIT.2006.871582. URL http://ieeexplore.ieee.org/document/1614066/.

M. F. Duarte, M. A. Davenport, D. Takhar, J. N. Laska, T. Sun, K. F. Kelly, and R. G. Baraniuk. Single-pixel imaging via compressive sampling. *IEEE Signal Processing Magazine*, 25(2):83–91, Mar. 2008. ISSN 1053-5888. doi: 10.1109/MSP.2007.914730. URL http://ieeexplore.ieee.org/document/4472247/.

R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern classification*. Wiley, New York, 2nd ed edition, 2001. ISBN 978-0-471-05669-0.

M. L. Eaton. A Note on Symmetric Bernoulli Random Variables. *The Annals of Mathematical Statistics*, 41(4):1223–1226, Aug. 1970. ISSN 0003-4851. doi: 10.1214/aoms/1177696897. URL http://projecteuclid.org/euclid.aoms/1177696897.

Y. C. Eldar and G. Kutyniok, editors. *Compressed sensing: theory and applications*. Cambridge University Press, Cambridge; New York, 2012. ISBN 978-1-107-00558-7.

Y. C. Eldar and S. Mendelson. Phase retrieval: Stability and recovery guarantees. *Applied and Computational Harmonic Analysis*, 36(3):473–494, May 2014. ISSN 10635203. doi: 10.1016/j.acha.2013.08.003. URL https://linkinghub.elsevier.com/retrieve/pii/S1063520313000717.

H. W. Engl and P. Kügler. Nonlinear inverse problems: theoretical aspects and some industrial applications. In *Multidisciplinary methods for analysis optimization and control of complex systems*, pages 3–47. Springer, 2005. ISBN 978-3-540-27167-3.

C. Fienup and J. Dainty. Phase retrieval and image reconstruction for astronomy. In *Image Recovery: Theory and Application*, pages 231–275. Academic Press, 1987. ISBN 0-12-663940-X.

J. R. Fienup. Phase retrieval algorithms: a comparison. *Applied Optics*, 21(15):2758, Aug. 1982. ISSN 0003-6935, 1539-4522. doi: 10.1364/AO.21.002758. URL https://www.osapublishing.org/abstract.cfm?URI=ao-21-15-2758.

T. Goldstein, C. Studer, and R. Baraniuk. A Field Guide to Forward-Backward Splitting with a FASTA Implementation. *arXiv:1411.3406 [cs]*, Nov. 2014. URL http://arxiv.org/abs/1411.3406. arXiv: 1411.3406.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts, 2016. ISBN 978-0-262-03561-3.

A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, Second Edition*. Society for Industrial and Applied Mathematics, second edition, Jan. 2008. ISBN 978-0-89871-659-7 978-0-89871-776-1. doi: 10.1137/1.9780898717761. URL http://epubs.siam.org/doi/book/10.1137/1.9780898717761.

I. Gulrajani, K. Kumar, F. Ahmed, A. A. Taiga, F. Visin, D. Vazquez, and A. Courville. PixelVAE: A Latent Variable Model for Natural Images. *arXiv:1611.05013 [cs]*, Nov. 2016. URL `http://arxiv.org/abs/1611.05013`. arXiv: 1611.05013.

P. Hand and V. Voroninski. Global Guarantees for Enforcing Deep Generative Priors by Empirical Risk. *arXiv:1705.07576 [cs, math]*, May 2017. URL `http://arxiv.org/abs/1705.07576`. arXiv: 1705.07576.

P. Hand, O. Leong, and V. Voroninski. Phase Retrieval Under a Generative Prior. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 9136–9146. Curran Associates, Inc., 2018. URL `http://papers.nips.cc/paper/8127-phase-retrieval-under-a-generative-prior.pdf`.

R. W. Harrison. Phase problem in crystallography. *Journal of the Optical Society of America A*, 10(5):1046, May 1993. ISSN 1084-7529, 1520-8532. doi: 10.1364/JOSAA.10.001046. URL `https://www.osapublishing.org/abstract.cfm?URI=josaa-10-5-1046`.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York, New York, NY, 2009. ISBN 978-0-387-84857-0. doi: 10.1007/978-0-387-84858-7. URL `http://link.springer.com/10.1007/978-0-387-84858-7`.

A. Hore and D. Ziou. Image quality metrics: PSNR vs. SSIM. In *2010 20th International Conference on Pattern Recognition*, pages 2366–2369. IEEE, 2010. doi: 10.1109/ICPR.2010.579.

T. Jebara. *Machine Learning: Discriminative and Generative*. Springer US, Boston, MA, 2004. ISBN 978-1-4419-9011-2. URL `http://dx.doi.org/10.1007/978-1-4419-9011-2`. OCLC: 853259270.

M. E. Jerrell. Automatic Differentiation and Interval Arithmetic for Estimation of Disequilibrium Models. *Computational Economics*, 10(3):295–316, Aug. 1997. ISSN 1572-9974. doi: 10.1023/A:1008633613243. URL `https://doi.org/10.1023/A:1008633613243`.

T. Karras, S. Laine, and T. Aila. A Style-Based Generator Architecture for Generative Adversarial Networks. *arXiv:1812.04948 [cs, stat]*, Dec. 2018. URL `http://arxiv.org/abs/1812.04948`. arXiv: 1812.04948.

V. Katkovnik, J. Astola, and K. Egiazarian. Discrete diffraction transform for propagation, reconstruction, and design of wavefield distributions. *Applied Optics*, 47(19):3481, July 2008. ISSN 0003-6935, 1539-4522. doi: 10.1364/AO.47.003481. URL `https://www.osapublishing.org/abstract.cfm?URI=ao-47-19-3481`.

V. Katkovnik, A. Migukin, and J. Astola. Backward discrete wave field propagation modeling as an inverse problem: toward perfect reconstruction of wave field distributions. *Applied optics*, 48(18):3407–3423, 2009. URL `http://www.cs.tut.fi/~lasip/DDT/MATRIX_DDT.pdf`.

F. Krahmer and Y.-K. Liu. Phase Retrieval Without Small-Ball Probability Assumptions. *IEEE Transactions on Information Theory*, 64(1):485–500, Jan. 2018. ISSN 0018-9448, 1557-9654. doi: 10.1109/TIT.2017.2757520. URL `http://arxiv.org/abs/1604.07281`. arXiv: 1604.07281.

F. Krahmer and D. Stöger. Complex phase retrieval from subgaussian measurements. *arXiv:1906.08385 [cs, math, stat]*, June 2019. URL `http://arxiv.org/abs/1906.08385`. arXiv: 1906.08385.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. doi: 10.1145/3065386.

Y. LeCun. The MNIST database of handwritten digits, 1998. URL `http://yann.lecun.com/exdb/mnist/`.

J. D. Lee, M. Simchowitz, M. I. Jordan, and B. Recht. Gradient Descent Converges to Minimizers. *arXiv:1602.04915 [cs, math, stat]*, Feb. 2016. URL `http://arxiv.org/abs/1602.04915`. arXiv: 1602.04915.

D. Lyon. Wave diffracton 4lambda Slit, 2010. URL `https://commons.wikimedia.org/wiki/File:Wave_Diffraction_4Lambda_Slit.png`.

V. Madisetti. *The digital signal processing handbook*. CRC press, 1997. ISBN 978-0-8493-8572-8.

C. A. Metzler, P. Schniter, A. Veeraraghavan, and R. G. Baraniuk. prDeep: Robust Phase Retrieval with a Flexible Deep Network. *arXiv:1803.00212 [cs, stat]*, Feb. 2018. URL `http://arxiv.org/abs/1803.00212`. arXiv: 1803.00212.

R. P. Millane. Phase retrieval in crystallography and optics. *JOSA A*, 7(3): 394–411, 1990. URL `http://xrm.phys.northwestern.edu/research/pdf_papers/1990/millane_josaa_1990.pdf`.

J. L. Mueller and S. Siltanen. *Linear and Nonlinear Inverse Problems with Practical Applications*. Society for Industrial and Applied Mathematics, Philadelphia, PA, Oct. 2012. ISBN 978-1-61197-233-7 978-1-61197-234-4. doi: 10.1137/1.9781611972344. URL `http://epubs.siam.org/doi/book/10.1137/1.9781611972344`.

L. Nickel. Phase Retrieval in Single Detector Cameras. Master's thesis, Westfaelische Wilhelmsuniversitaet Muenster, Muenster, Oct. 2018.

M. O'Neill. Neural Network for Recognition of Handwritten Digits, 2006. URL `https://www.codeproject.com/Articles/16650/Neural-Network-for-Recognition-of-Handwritten-Digi`.

E. Osherovich. Numerical methods for phase retrieval. *arXiv:1203.4756 [astro-ph, physics:physics]*, Mar. 2012. URL `http://arxiv.org/abs/1203.4756`. arXiv: 1203.4756.

N. Parikh, S. Boyd, and others. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014. doi: 10.1561/2400000003.

G. Pisier. Subgaussian sequences in probability and Fourier analysis. *arXiv:1607.01053 [math]*, July 2016. URL `http://arxiv.org/abs/1607.01053`. arXiv: 1607.01053.

M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein. On the Expressive Power of Deep Neural Networks. *arXiv:1606.05336 [cs, stat]*, June 2016. URL `http://arxiv.org/abs/1606.05336`. arXiv: 1606.05336.

Y. Romano, M. Elad, and P. Milanfar. The Little Engine That Could: Regularization by Denoising (RED). *SIAM Journal on Imaging Sciences*, 10(4): 1804–1844, Jan. 2017. ISSN 1936-4954. doi: 10.1137/16M1102884. URL `https://epubs.siam.org/doi/10.1137/16M1102884`.

F. Shamshad and A. Ahmed. Robust Compressive Phase Retrieval via Deep Generative Priors. *arXiv:1808.05854 [cs, stat]*, Aug. 2018. URL `http://arxiv.org/abs/1808.05854`. arXiv: 1808.05854.

Y. Shechtman, A. Beck, and Y. C. Eldar. GESPAR: Efficient Phase Retrieval of Sparse Signals. *arXiv:1301.1018 [cs, math]*, Jan. 2013. URL `http://arxiv.org/abs/1301.1018`. arXiv: 1301.1018.

Y. Shechtman, Y. C. Eldar, O. Cohen, H. N. Chapman, J. Miao, and M. Segev. Phase Retrieval with Application to Optical Imaging. *arXiv:1402.7350 [cs, math]*, Feb. 2014. URL `http://arxiv.org/abs/1402.7350`. arXiv: 1402.7350.

L. A. Shepp and B. F. Logan. The Fourier reconstruction of a head section. *IEEE Transactions on Nuclear Science*, 21(3):21–43, June 1974. ISSN 0018-9499, 1558-1578. doi: 10.1109/TNS.1974.6499235. URL `http://ieeexplore.ieee.org/document/6499235/`.

N. Z. Shor. *Minimization methods for non-differentiable functions*. Number 3 in Springer series in computational mathematics. Springer-Verlag, Berlin ; New York, 1985. ISBN 978-0-387-12763-7.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, and others. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017. doi: 10.1038/nature24270.

T. Strohmer and R. Vershynin. A randomized Kaczmarz algorithm with exponential convergence. *arXiv:math/0702226*, Feb. 2007. URL `http://arxiv.org/abs/math/0702226`. arXiv: math/0702226.

Y. S. Tan and R. Vershynin. Phase Retrieval via Randomized Kaczmarz: Theoretical Guarantees. *arXiv:1706.09993 [cs, math, stat]*, June 2017. URL `http://arxiv.org/abs/1706.09993`. arXiv: 1706.09993.

V. Vapnik. Principles of risk minimization for learning theory. In *Advances in neural information processing systems*, pages 831–838, 1992. URL `http://papers.nips.cc/paper/506-principles-of-risk-minimization-for-learning-theory.pdf`.

G. Wang, G. B. Giannakis, and Y. C. Eldar. Solving Systems of Random Quadratic Equations via Truncated Amplitude Flow. *arXiv:1605.08285 [cs, math, stat]*, May 2016. URL `http://arxiv.org/abs/1605.08285`. arXiv: 1605.08285.

Z. Wang, A. C. Bovik, H. R. Sheikh, S. Member, E. P. Simoncelli, and S. Member. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing*, 13:600–612, 2004. URL `https://ece.uwaterloo.ca/~z70wang/publications/ssim.pdf`.

K. Wei. Solving systems of phaseless equations via Kaczmarz methods: A proof of concept study. *arXiv:1502.01822 [math]*, Feb. 2015. URL `http://arxiv.org/abs/1502.01822`. arXiv: 1502.01822.

W. Wirtinger. Zur formalen Theorie der Funktionen von mehr komplexen Veränderlichen. *Mathematische Annalen*, 97(1):357–375, 1927. URL `http://www.digizeitschriften.de/download/PPN235181684_0097/PPN235181684_0097___log19.pdf`.

K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang. Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155, July 2017. ISSN 1057-7149, 1941-0042. doi: 10.1109/TIP.2017.2662206. URL `http://arxiv.org/abs/1608.03981`. arXiv: 1608.03981.