Sparse Neural Networks

Dissertation

zur Erlangung des Doktorgrades

Dr. rer. nat.

der Fakultät für Ingenieurwissenschaften
und Informatik der Universität Ulm

von

Markus Thom

aus Weert, Niederlande

Ulm, im Jahr 2014

Amtierende Dekanin:	Prof. Dr. phil. Tina Seufert
Gutachter:	Prof. Dr. rer. nat. Günther Palm
Gutachter:	Prof. Dr. rer. nat. Heiko Neumann
Gutachterin:	Prof. Dr. rer. nat. Barbara Hammer
Tag der Promotion:	9. Januar 2015

Zusammenfassung

Diese Arbeit untersucht spärliche neuronale Netze, das heißt künstliche neuronale Informationsverarbeitungssysteme die in ihrer Struktur und Dynamik zur Ressourceneinsparung eingeschränkt werden. Das Spärlichkeitskonzept bezieht sich hier auf die Verbindungsstruktur der Netze, so dass jedes Neuron nur von einer beschränkten Anzahl anderer Neuronen eine Eingabe erhält, sowie auf den Netzzustand der den Aktivitätsgrad der Gesamtheit der Neuronen beschreibt, so dass zu jedem Zeitpunkt nur wenige Neuronen aktiv sind.

Das mathematische Werkzeug um diese Eigenschaften zu erhalten, ein Projektionsoperator der zu jedem gegebenen Vektor die beste Approximation findet, welche einen voreingestellten Spärlichkeitsgrad erzielt, wird umfassend analysiert und effiziente Algorithmen für seine Berechnung werden vorgeschlagen. Da die Zielmengen für die Projektion nicht konvex sind, können Standardverfahren hier nicht angewendet werden und neue Methoden zur Lösung dieses Optimierungsproblems müssen entwickelt werden. Auf dieser Theorie aufbauend werden neue Modelle zur Lösung von Klassifikationsaufgaben unter expliziten Spärlichkeitsbedingungen vorgeschlagen und ihre Eigenschaften untersucht. Dabei werden neue Ansätze zur Lösung der Probleme der ineffizienten Inferenz und der schwachen diskriminatorischen Eigenschaften spärlicher Repräsentationen entwickelt. Es wird gezeigt dass Spärlichkeit der Regularisierung dient, verglichen mit klassischen nicht-spärlichen Modellen können erheblich bessere Klassifikationsergebnisse erreicht werden. Darüber hinaus kann der spärliche Datenfluss verwendet werden um die Berechnungskomplexität der Anwendung eines trainierten Klassifikators um etwa eine Größenordnung zu senken.

Eine umfangreiche Auswertung erbringt den Nachweis der Signifikanz der Ergebnisse und ordnet die vorgeschlagenen Modelle in einer Vielzahl alternativer Mustererkennungsverfahren ein. Der Vergleich schließt mit der Überlegenheit spärlicher neuronaler Netze. Sonst können nur sehr komplexe und spezialisierte Modelle die Klassifikationsgenauigkeit auf Kosten einer um mehrere Größenordnungen gesteigerten Berechnungskomplexität verbessern. Der Beweis der durch die Spärlichkeit gesteigerten Leistung wird darüber hinaus durch Anwendung der in dieser Arbeit entwickelten Methoden zur Erkennung von Fußgängern bei Nacht von einem fahrenden Fahrzeug aus untermauert. Hier werden fortlaufend die Sensordaten von in unterschiedlichen Bereichen des Infrarot-Spektrums empfindlichen Kamerasystemen ausgewertet um gefährdete Verkersteilnehmer in großer Entfernung zu erkennen, was eine adäquate Warnung der Fahrzeugführer mit einem großen Sicherheitsabstand ermöglicht. Spärliche neuronale Netze eignen sich hervorragend für diese zeitkritische und praxisrelevante Aufgabe, sogar spezialisierte und für den verwendeten Rechner hochoptimierte Routinen werden übertroffen. Das Ergebnis ist ein echtzeitfähiges Erkennungssystem welches sogar in anspruchsvollen und hoch strukturierten Szenarien eine hohe Genauigkeit erzielt.

Abstract

This work investigates Sparse Neural Networks, which are artificial neural information processing systems restricted in their structure and dynamics to conserve resources. The concept of sparseness refers to the network's connectivity structure, such that each neuron receives inputs from only a limited number of other neurons, and to the network's state which describes the level of activity of the entire neural population, so that only few neurons are active at any one time.

The mathematical tool to achieve these properties, a projection operator that finds the best approximation to any given vector fulfilling a pre-defined sparseness degree, is extensively analyzed and efficient algorithms for its computation are proposed. Since the target sets for the projection are non-convex, standard approaches are not applicable and new methods must be developed to solve this optimization problem. Building upon this theory, original models suited to solving classification tasks subject to explicit sparseness constraints are proposed and their characteristics studied. In doing so, novel approaches to solving the problems of inefficient inference and poor discriminatory performance of sparse representations are developed. It is demonstrated that sparseness acts as a regularizer and significantly better classification results can be obtained compared to classical non-sparse models. Moreover, the sparse data flow can be exploited to reduce the computational complexity of the application of a trained classifier by approximately one order of magnitude.

A comprehensive evaluation shows the significance of the outcome and puts the proposed models in context with a multitude of alternative pattern recognition algorithms. The comparison concludes with the superiority of Sparse Neural Networks, only very complex and specialized models could improve on the classification accuracy, at the cost of an increase in computational complexity of several orders of magnitude. The evidence of boosted performance through the sparseness concept is furthermore substantiated by application of the methods developed in this work to night-time pedestrian detection from a moving vehicle. Here, the sensory data of camera systems sensitive in distinct regions of the infrared spectrum is continuously processed to detect vulnerable traffic participants at a considerable distance, allowing for an adequate warning with a large safety margin to vehicle drivers. Sparse Neural Networks excel in this time-critical real-world task, outperforming even specialized routines that were highly optimized for the computing machine the algorithms were run on. The result is a real-time capable detection system that provides high accuracy even in challenging, highly structured scenarios.

Contents

1 Introduction

Information processing systems are abstract descriptions of the process of transforming input information into output information. This conversion process is described in terms of an algorithm. In the field of neuroinformatics, artificial neural information processing systems are studied. These systems are mathematical models of processes that take place in the nervous systems of animal life-forms, including humans. In these *neural networks*, small-scale information processing units, called *neurons*, are connected with other neurons via junctions, called *synapses*. The level of abstraction of these models is very different with various approaches, reaching from models on the molecular level to more abstract models on the level that has been mathematically proved to be most concise yet universal.

A very popular and simple model is the *Multi-Layer Perceptron (MLP)*. It computes an operator between Euclidean spaces by factorization into functions that are computed by the individual layers of the model. In each layer, the input vector is multiplied with a weight matrix, a threshold vector is added to the product, and finally a nonlinear transfer function is applied. It was shown that an MLP with only two layers is sufficient to approximate sufficiently smooth functions with arbitrary precision. This is known as the *universal approximator* property and makes MLPs appealing for regression and classification tasks.

As this model is very simple and a universal approximator, it has been studied intensely over the past decades. The key property that is used in this work in combination with the MLP is *sparseness*. In mathematics, a vector or a matrix with only a very limited amount of nonzero entries is called *sparsely populated*. Sparse objects emerge in a natural manner in a variety of tasks, in particular in signal processing applications.

In biological neural networks, the term sparseness has a two-fold meaning. First, not every neuron is connected to every other neuron. Thus, there is a *sparse connectivity* in these systems. Second, at all times during the information conversion process, only a small fraction of neurons is active. Hence, the whole neuronal population features *sparse activity*. Sparseness has evolved naturally in biological nervous systems, for it provides several advantages over arbitrary structure, such as reduction of wiring costs and energy consumption.

This work is organized as follows. The following chapter gives an introduction to the sparse information processing paradigm and reviews the state of the art of formal sparseness measures and models that feature sparse connectivity or sparse activity. Chapter 3 introduces the *MNIST database of handwritten digits*. It is a benchmark data set for classification tasks and used throughout this work to assess classification capabilities. Chapter 4 analyzes the properties of a *normalized and scale-invariant sparseness measure*. Special attention is paid to the algorithmic computation of *projections* onto sets on which the investigated sparseness measure attains a desired target sparseness degree. This theoretical treatment is the key to optimization of Artificial Neural Networks subject to explicit sparseness constraints.

The theory developed in Chapter 4 is applied in the subsequent chapters. In the quest for a comprehensive model based on sparse information processing and suitable for classification tasks, three architectures with individual advantages on both the theoretical and the practical side are proposed. The *Sparse Coding for Fast Classification* model is developed in Chapter 5. It extends the classical *Non-Negative Matrix Factorization with Sparseness Constraints* with fast inference of sparse internal representations and classification capabilities. To allow processing of very large data sets, this model is simplified by neglecting sparse activity and focusing on *Sparsely Connected MLPs* in Chapter 6. This modification of MLPs exhibits excellent generalization performance while requiring only a small fraction of synaptic connections. In Chapter 7 this model is further extended by sparse activity through implementation of a sparseness-enforcing projection operator as neural transfer function. The newly proposed model, called *Supervised Online Auto-Encoder*, can achieve sparsely populated internal representations in a mathematically sound form and further improves classification capabilities.

Chapter 8 contains a comprehensive evaluation of the characteristics of the proposed models with regard to several aspects. A statistical analysis fortifies the empirical evidence of the superiority of sparseness concepts over classical non-sparse approaches. Moreover, the relationship between classification performance on the MNIST data set and the required computational resources is examined. This includes a multitude of competing learning algorithms for which results have been reported in the literature. The evaluation shows that sparseness is indeed an efficiency concept, for it helps to reduce both the number of misclassifications and the computational complexity by a significant amount. This is moreover demonstrated on the real-world task of *night-time pedestrian detection from a moving vehicle* in Chapter 9, where a complete system for real-time detection of vulnerable traffic participants based on a heterogeneous rejection cascade approach is proposed. Here, a challenging detection problem is investigated and solved by a combination of a variety of feature types and learning algorithms. The impact of sparse information processing is analyzed in terms of detection performance and the run-time of the pedestrian detection procedure on a real computing machine, and the findings of the previous chapters are confirmed. This work is concluded with a discussion of its main theses in Chapter 10.

For completeness, there are three appendixes to this work. Appendix A defines the notation by listing the mathematical symbols used. Appendix B provides a short collection of results from matrix algebra and matrix calculus which are used for deriving gradient information in this work. An overview of learning schemes for Artificial Neural Networks is given in Appendix C, which includes different strategies for efficient parameter tuning, and commonly used transfer functions and similarity measures.

2 The Sparse Information Processing Paradigm

Sparseness is an efficiency concept in neural networks, and exists in two variants in that context [1]. The *sparse connectivity* property means that each neuron is connected to only a limited number of other neurons. The *sparse activity* property means that only a small fraction of neurons is active at any time.

Completely dense connectivity in a biological neuronal system would mean that each neuron is connected to every other neuron. In large systems like the human brain, there are about 10^{11} neurons [2]. The average neuron has about 1000 synaptic outputs, and receives up to 10 000 synaptic inputs, thus about 10^{14} synaptic connections emerge in the brain [2]. Connectivity is therefore sparse on a macroscopic scale. Furthermore, sparseness of connectivity increases quickly with brain volume, such that the human brain is about 170 times more sparse than a rat brain [3]. The brain forms local, functional networks, including ones for visual and acoustic perception, and dedicated networks for carrying out motoric activity [2]. Because of the locality, short-distance connections are predominant, justifying the so-called minimal axon length principle [3]. This principle in turn is motivated by the assumption that the brain conserves its biochemical resources, and hence reduces wiring costs [1]. On the microscopic scale of functional networks, connectivity is also sparse, as an analysis of rat visual cortex has shown: The probability of a connection between two neurons located within the same area ranges from about 1% to 10% [4, 5]. Moreover, connectivity in higher layers of information processing is sparser than in lower layers [6, 7]. In fact, there are small clusters of connected neurons that form functional sub-networks, introducing a neuronal structure of organization only slightly coarser than individual neurons [8].

Feed-forward neural networks are artificial neural information processing systems where information is propagated layer-wise in exactly one direction. The difference between dense and sparse connectivity is illustrated in Figure 2.1 for a simple network with only an input layer and an output layer. In networks with dense connectivity, the state of each neuron in the output layer depends on the state of all the neurons from the input layer. Sparse connectivity on the other hand incorporates a sparse data flow: The state of a neuron of the output layer depends on the state of only a small number of neurons from the input layer. Inference of the output layer state from the input layer state can thus be achieved with fewer computations and is more efficient. More formally, a network is called *sparsely connected* if and only if

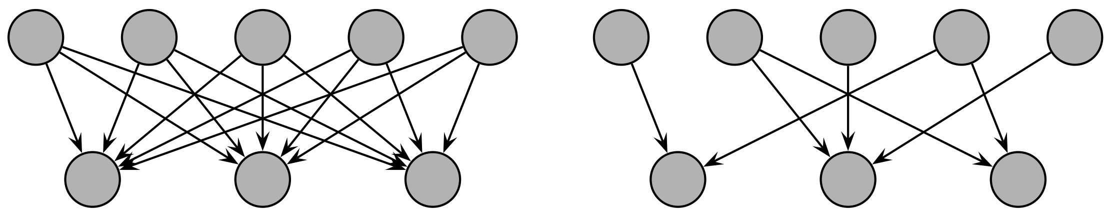

(a) Dense connectivity. (b) Sparse connectivity.

Figure 2.1: Principle of dense connectivity versus sparse connectivity. The arrows indicate which neurons from the bottom row, that is from the output layer, receive input from the neurons from the upper row, which forms the input layer. With sparse connectivity, the state of the output layer can be inferred more efficiently than with dense connectivity, as less information has to be processed.

its matrix of synaptic weights is sparsely populated, that is there are only few active synaptic connections. Here, a vanishing synaptic strength is considered to indicate that there is no synaptic connection between two neurons.

A neuron is called *active* if it emits an action potential within a given time frame. This activates subsequent neurons by means of a chemical or electrical transmitter. The degree of activity, and therefore the information that is stored within the neural network, is encoded through the frequency of the action potentials [2]. It was observed that there are neurons in the cat's primary visual cortex that are selective to very certain stimuli [9]. More precisely, these neurons have only shown activity when a light bar of certain orientation was projected onto the animal's retina. As there are multiple such neurons, where each has another preferred orientation, presentation of a certain pattern induces *sparse activity* among this neuronal population. In other words, the vector consisting of the activity of all neurons is sparsely populated [10]. This sparse coding principle is illustrated in Figure 2.2.

The striking advantage for the organism's metabolism is the conservation of energy, as inactive neurons consume much less energy than highly active ones [11]. This is because after every action potential, ionic balance has to be restored to repolarize the neuronal membrane [12]. An analysis of the energy that is consumed by one action potential, and considering the energy available to the entire visual cortex shows that only 1.5% to 4% of neurons can be active at any one time [12]. A similar conclusion can be drawn from a purely information theoretic point of view. Considering a population of binary neurons, representational capacity is maximized when half of the population is active at any time [13]. If additionally the energy consumption is minimized, while still maximizing representational capacity, then only 2% to 16% of all neurons should be active at any one time, where the exact number depends on the cost of generating an action potential [13].

Sparse connectivity and sparse activity are concepts that exist in biological neural networks. Since they constitute resource restraining mechanisms, they are appealing paradigms for Ar-

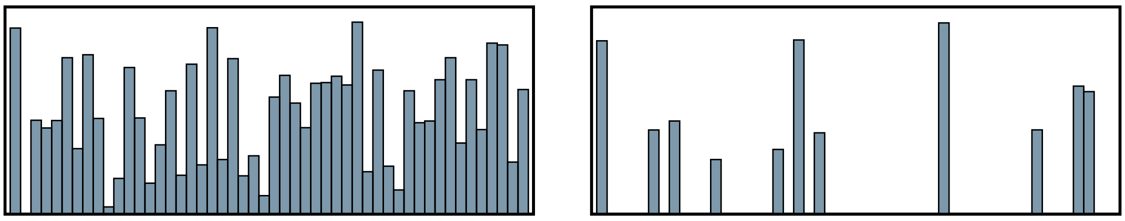

(a) Dense activity. (b) Sparse activity.

Figure 2.2: A comparison of dense activity versus sparse activity. The abscissa indicates the entries in an activation vector, and the ordinate indicates the amount of the activity. Dense activity means that every neuron is active to some degree. With sparse activity, only a few neurons are active, and the majority of the neuronal population is completely inactive.

tificial Neural Networks. As will be demonstrated in this work, sparseness can be exploited to save both computational time and storage requirements in an information processing setup, without any adverse effects.

The remainder of this chapter is structured as follows. First, a fundamental result of sparse coding is recapitulated, namely that learning bases with enforced sparse activity from natural image patches produces bases similar to receptive fields in mammalian early vision. Next, abstract sparseness measures and their properties are reviewed, and the commitment to one intuitive measure is motivated. Further, the advantages and drawbacks of explicit sparseness constraints versus implicit sparseness constraints are discussed. The chapter concludes with an overview of existing models that incorporate sparse information processing with respect to activity or connectivity and a discussion.

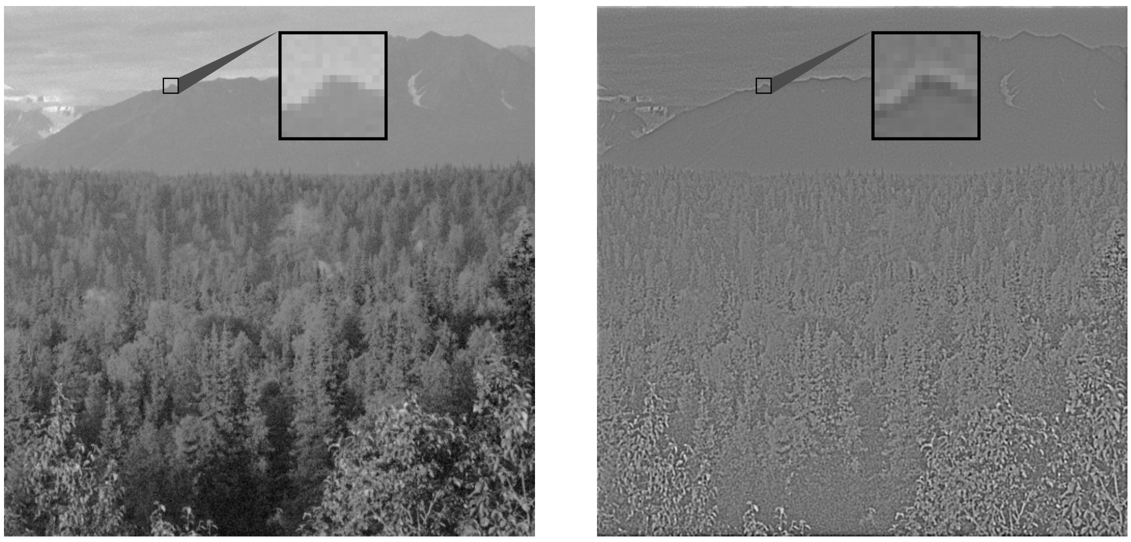
2.1 Sparse Coding of Natural Image Patches

Sparse coding denotes the idea that signals can be approximated by superposition of very few elements of a large dictionary. The selection of relevant elements and their weighting depend on the concrete signal, but the dictionary is assumed constant for all signals.

More formally, let Ω denote a sample space and let $X\colon \Omega \to \mathbb{R}^d$ denote a multivariate random variable that models the entirety of all possible signals. It is then assumed that there exists a *dictionary of bases*, that is $n \in \mathbb{N}$ vectors $w^{(1)}, \dots, w^{(n)} \in \mathbb{R}^d$ that are collectively denoted by $\mathcal{W}$, and a function $h_{\mathcal{W}}\colon \mathbb{R}^d \to \mathbb{R}^n$ that computes *code words* in dependence of the values of X. Let $e_i^T h_{\mathcal{W}}(X)$ denote the ith entry of $h_{\mathcal{W}}(X)$, then the condition for sparse coding is that

$$X \approx \sum_{i=1}^{n} e_i^T h_{\mathcal{W}}(X) \cdot w^{(i)}, \text{ where } h_{\mathcal{W}}(X) \text{ should be sparsely populated.}$$

If sparse connectivity is involved, $\mathcal{W}$ should also be sparsely populated.



(a) Raw image. (b) Whitened image.

Figure 2.3: Extraction of patches from natural images for the experiments of [10]. First, a bandpass filter is applied to the raw image to yield a whitened image. Next, small patches are extracted from random positions to yield the data set. Raw images courtesy of Bruno Olshausen [14].

The field of sparse coding has received a lot of attention since the pioneering work of Olshausen and Field [10]. They have shown that a sparse coding model applied to natural images yields a dictionary of bases $\mathcal{W}$ that features the same key properties as simple cells in mammalian primary visual cortex. In other words, they have found an algorithm that produces the same result as the biological development of low-level vision in mammalian brains. As a vast range of algorithms from the field of computer vision is inspired by biological archetypes, the results of Olshausen and Field motivated huge research interest.

To generate a data set, Olshausen and Field used ten images of size 512×512 pixels of natural surroundings in the American northwest. They applied a whitening filter to the raw images that performs a bandpass operation. This alleviates the emergence of a high number of low-frequency bases in $\mathcal{W}$, and the removal of very high frequencies attenuates sampling artifacts. From the whitened images, small image patches of size 16×16 pixels were extracted from random positions. Finally, the patches were vectorized to yield elements from $\mathbb{R}^{256}$. The process of data set generation is illustrated in Figure 2.3.

The objective function E_{OF} should then be optimized on this data set by variation of the bases and the code words [10]:

$$E_{\text{OF}} := \left\| x - \sum_{i=1}^{n} h_i w^{(i)} \right\|_2^2 + \alpha \left\| h \right\|_1 \overset{!}{\to} \min_{\mathcal{W},h}.$$

The first term measures the deviation between a sample $x \in \mathbb{R}^d$ from the data set and its reproduction. The reproduced sample is computed using a latent code word $h \in \mathbb{R}^n$ and n

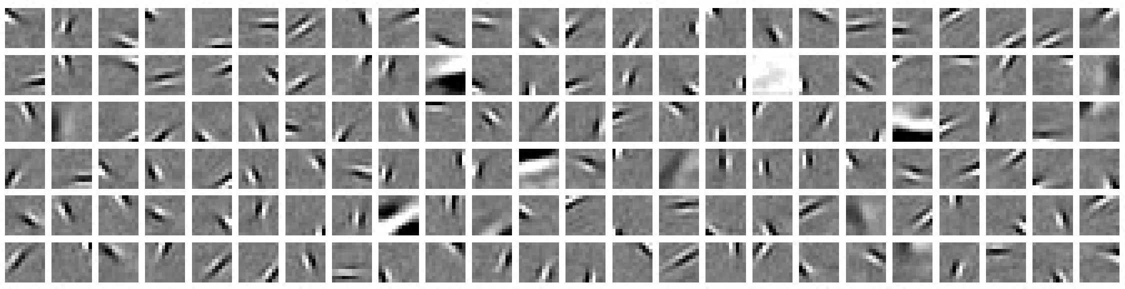


Figure 2.4: Optimal bases $w^{(1)}, \dots, w^{(144)} \in \mathbb{R}^{256}$ for reproduction of a natural image data set via sparsely populated code vectors. For display purposes, the bases have been reshaped to yield matrices from $\mathbb{R}^{16 \times 16}$. The entries were mapped to grayscales such that small values yield black and large values yield white, and a value of zero always yields the same level of gray. Image courtesy of Bruno Olshausen [14].

bases $w^{(1)}, \dots, w^{(n)} \in \mathbb{R}^d$. The second term penalizes non-sparseness of the code words, and $\alpha \in \mathbb{R}$ serves as trade-off constant for both terms. Algorithmic minimization of E_{OF} on the whole data set is accomplished as follows. First, a batch of samples is picked from the entire data set. Then their optimal code words are computed by minimizing E_{OF} subject to h while $\mathcal{W}$ is kept fixed. The function $h_{\mathcal{W}}$ for code word inference is thus computed by solving an optimization problem. Using the optimal code words, the gradient of E_{OF} subject to $\mathcal{W}$ is computed and averaged over the batch. Using gradient descent, $\mathcal{W}$ is then updated, and the procedure starts over again by picking a new batch of samples. The algorithm stops when the change in E_{OF} is sufficiently small to state convergence.

In this model, only non-sparseness of activity is penalized by an implicit, additive term. Sparse connectivity is neither preferred nor punished. The resulting bases computed from the patches from natural images are depicted in Figure 2.4. The bases resemble Gabor-like filters [15], that is they are spatially localized, oriented and bandpass [10]. Therefore, the bases share the same key properties as neurons in mammalian early vision [16, 17]. This motivates the conjecture that sparse coding is a necessary and sufficient condition for efficient biological vision.

The remainder of this chapter focuses on the expansion of the understanding of the sparseness concept, based on the model of Olshausen and Field. The next step in doing so is an analysis of how to measure sparseness in a mathematically proper way.

2.2 Sparseness Measures

For application of sparseness concepts in mathematical models and simulations, a formal sparseness measure is needed. This is a function that takes a real-valued vector as input and returns a real number that assesses the sparseness degree of the input vector. There is a large variety of such measures, and an elaborate discussion of their properties is given in [18]. This section focuses on the most important ones, namely the L_0 pseudo-norm, the L_1 norm, the slog function and Hoyer's sparseness measure σ. Contour plots for $d = 2$ dimensions are given in Figure 2.5 for orientation.

2.2.1 The Counting Norm and Combinatorial Problems

The L_0 pseudo-norm is a natural sparseness measure. It is computed by counting the number of non-vanishing entries in a vector:

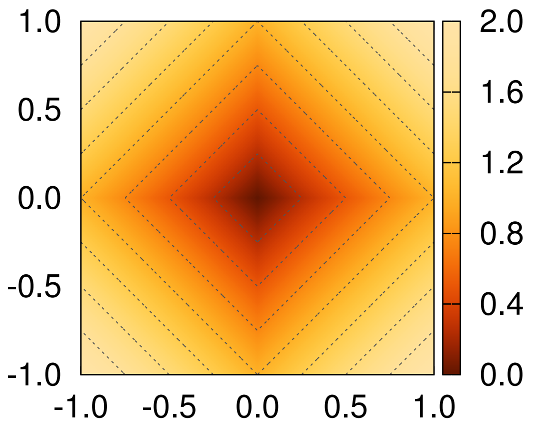
$$\|\cdot\|_0 \colon \mathbb{R}^d \to \{0,\ldots,d\}, \quad x \mapsto |\{\, i \in \{1,\ldots,d\} \mid x_i \neq 0 \}|\,.$$

It is a definite function and fulfills the triangle inequality. Because of $\|\lambda x\|_0 = \|x\|_0 \neq |\lambda| \cdot \|x\|_0$ for $\lambda \neq 0$ and $x \in \mathbb{R}^d$, the property of absolute homogeneity does not hold. Therefore this function is not a regular vector norm but merely a pseudo-norm.

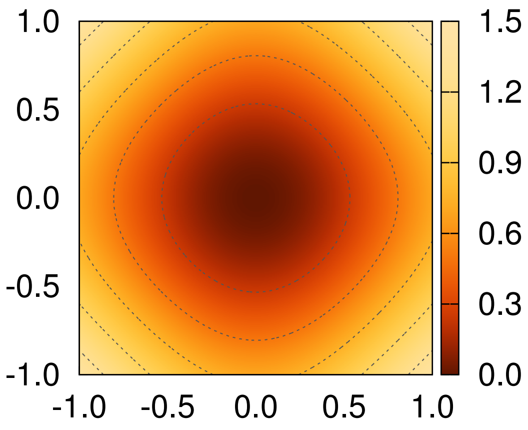
It has been shown that it can lead to biologically more plausible results compared to other sparseness measures [19], but it has the significant disadvantage of not being differentiable; it is in fact not even a continuous function. Gradient information thus cannot be incorporated for solving problems subject to the L_0 pseudo-norm. This renders a variety of machine learning problems very difficult to solve [20, 21], that is, they belong to the class of *NP-complete problems*.

As an example for such hard problems, consider the discrete rucksack problem [22, 23] from the domain of combinatorial optimization. It is known to be NP-complete, and the only deterministic algorithms known to solve it exactly have exponential run-time. Let $d \in \mathbb{N}$ be constant and consider d objects which should be packed into a rucksack. These objects are characterized by values $v_1,\ldots,v_d \in \mathbb{N}$ and weights $g_1,\ldots,g_d \in \mathbb{N}$. The task is to find a subset $I \subseteq \{1,\ldots,d\}$ of the objects such that the sum of the values is maximized, while the sum of the weights must be less than a constant $G \in \mathbb{N}$. In formal terms, the problem is given as $\max_{I \subseteq \{1,\ldots,d\}} \sum_{i \in I} v_i$ such that $\sum_{i \in I} g_i \leq G$. The trivial algorithm that enumerates each subset I and checks whether both conditions are fulfilled has exponential run-time.

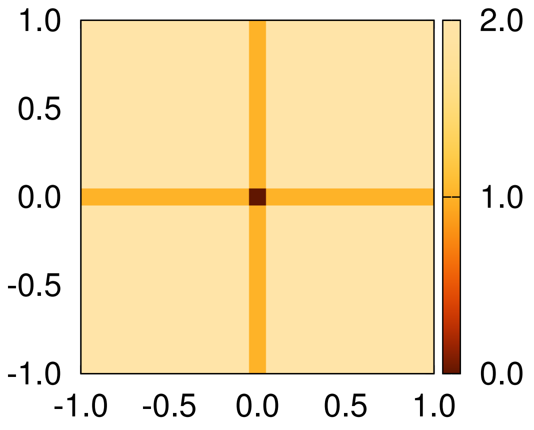
By definition, all NP-complete problems can be reduced in polynomial time to the rucksack problem. This means that a polynomial-time algorithm for solving *any* NP-complete problem could be used to provide a polynomial-time algorithm for *all* NP-complete problems. By the current state of science, it is unclear whether such an algorithm exists or not, or if it is even decidable in the sense of computability theory whether such an algorithm exists [23, 24]. Because of this unfortunate situation, it can currently not be expected that a solution to the rucksack problem, or any other NP-complete problem, can be found using a deterministic algorithm that terminates in reasonable time, even when the problem size is only moderate.



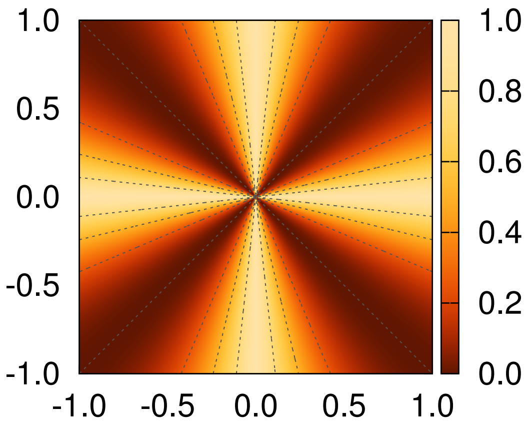
(a) L_1 norm.



(b) slog function.



(c) L_0 pseudo-norm.



(d) Hoyer's σ.

Figure 2.5: Visualization of sparseness measures in two dimensions. The abscissa and the ordinate specify the entries of a two dimensional vector, and the sparseness degree obtained by the respective measures is color coded. The gray dashed lines are contour levels at intervals of 0.25. The drawing of the L_0 pseudo-norm was exaggerated for illustrational purposes. While the L_1 norm and the slog function increase with the magnitude of the argument, the non-continuous L_0 pseudo-norm and Hoyer's sparseness measure are scale-invariant. The latter attains its minimum value of zero on the main diagonals, and the maximum value one is attained on the coordinate axes. Because of its shape and its well-behaving analytical properties, it can be considered a relaxation of the L_0 pseudo-norm.

2.2.2 Relaxation, the Manhattan Norm and the slog Function

As an exact solution is not always necessary, a common technique for avoiding these difficult problems is relaxation [23]. Hard constraints are replaced by softer ones, which are easier to fulfill. This results in an expansion of the set of feasible solutions, and can add helpful structure to it.

As an example, consider the fractional rucksack problem, which is a relaxation of the original rucksack problem by allowing objects to be dividable into smaller pieces. Formally, instead of finding a subset $I \subseteq \{1, \ldots, d\}$ the task is now to find a vector $a \in [0,\ 1]^d$ resembling the fraction of objects to pack into the rucksack. Here, $\sum_{i=1}^{d} a_i v_i$ should be maximized while $\sum_{i=1}^{d} a_i g_i \leq G$ holds. There exists a greedy algorithm for computing an exact solution of this problem [23]. The run-time of that algorithm is dominated by sorting the d objects, hence the computational complexity is quasilinear. This renders the solution of the relaxed problem very efficient compared to an exponential time algorithm for the discrete problem.

The result of applying the idea of relaxation to sparseness measures is the L_1 norm, which is also called Manhattan norm or the sum of the absolute entries:

$$\|\cdot\|_1 : \mathbb{R}^d \to \mathbb{R}_{\geq 0}, \quad x \mapsto \sum_{i=1}^{d} |x_i| .$$

It can be considered a convex relaxation of the L_0 pseudo-norm [25], and due to its differentiability, gradient information can be exploited in optimization tasks. Therefore, the enhanced structure helps to efficiently find solutions as it provides information which is not available in discrete spaces. The relaxation process can thus be employed as heuristics to find approximate solutions to problems which would otherwise be practically impossible to solve.

A drawback that still arises with all regular vector norms cast as sparseness measures is scale-invariance, which affects the L_1 norm in particular. In other words, because $\|\lambda x\| = |\lambda| \cdot \|x\|$ for all $\lambda \in \mathbb{R}$, all $x \in \mathbb{R}^d$ and every norm $\|\cdot\| : \mathbb{R}^d \to \mathbb{R}$, the measured sparseness depends strongly on the scale of the vectors, although one would admit that both x and λx possess the same degree of sparseness unless $\lambda = 0$.

Another function commonly used as sparseness measure is the slog function:

$$\operatorname{slog}\colon \mathbb{R}^d \to \mathbb{R}_{\geq 0}, \quad x \mapsto \sum_{i=1}^{d} \log(1 + x_i^2).$$

Its shape is similar to that of the L_1 norm and it is not scale-invariant. Numerical experiments found no major difference between the results yielded using the L_1 norm and when the slog function is employed [10]. Because there are strong theoretical results that the L_1 norm can be used to achieve solutions that are also sparse in the L_0 sense [25], there is no justification why the slog function should be preferred over the L_1 norm.

2.2.3 Hoyer's Normalized Sparseness Measure

Throughout this work, Patrik Hoyer's smooth sparseness measure σ is used, which was first described in [26]:

$$\sigma \colon \mathbb{R}^d \setminus \{0\} \to [0,\ 1], \qquad x \mapsto \frac{\sqrt{d} - \|x\|_1/\|x\|_2}{\sqrt{d}-1}.$$

It is a normalized quotient of the L_1 norm and the L_2 norm of the argument. As σ evidently only depends on the ratio of two norms that fulfill absolute homogeneity, it is scale-invariant, that is $\sigma(\lambda x) = \sigma(x)$ for all $\lambda \neq 0$ and all $x \in \mathbb{R}^d \setminus \{0\}$. The normalization has been chosen such that σ computes numbers between zero and one. This follows from the basic inequalities $\|x\|_2 \leq \|x\|_1 \leq \sqrt{d}\,\|x\|_2$ that hold for all $x \in \mathbb{R}^d$ [27], which imply $1 \leq \|x\|_1/\|x\|_2 \leq \sqrt{d}$. As the bounds of the inequalities are tight, that is there are vectors for which equality holds, σ takes its extreme values of zero and one as well. For example, if $i \in \{1,\dots,d\}$ and $e_i \in \mathbb{R}^d$ is the ith canonical basis vector, then $\|e_i\|_1 = \|e_i\|_2$ and hence $\sigma(e_i) = 1$. On the other hand, if $e := \sum_{i=1}^d e_i \in \mathbb{R}^d$ denotes the vector where all entries are identical to unity, then $\|x\|_1 = d$ and $\|x\|_2 = \sqrt{d}$, hence $\sigma(e) = 0$. Because of the negative sign of the ratio of norms, higher numbers of σ indicate more sparse vectors.

Other notable properties of σ include the so-called "Robin Hood" and "Rising Tide" properties [18]. The first one states that sparseness should decrease when one large entry is reduced in favor for a smaller entry, that is $\sigma(x) > \sigma(x - \alpha e_i + \alpha e_j)$ should hold when $x_i > x_j$ and $0 < \alpha < 1/2 \cdot (x_i - x_j)$. The latter property means that sparseness should be decreased when a constant positive value is added to all entries, in formal terms $\sigma(x) > \sigma(x + \alpha e)$ for $\alpha > 0$. Note that the difference between $\sigma(x)$ and $\sigma(x + \alpha e)$ is small if α is small due to the continuity of σ, which renders σ robust to random noise. This property does not hold for the L_0 pseudo-norm since here $x + \alpha e$ is never sparse if $\alpha \neq 0$ even if x was sparse in the L_0 sense.

Further, Hoyer's sparseness measure attains a smaller value given a vector that is extended with vanishing entries. This is also known as the "Babies" property, that is when $x \in \mathbb{R}^d$ is extended to a vector $\tilde{x} \in \mathbb{R}^n$, $n > d$, by appending zero entries at its end, then $\sigma(\tilde{x}) > \sigma(x)$ should apply [18]. In other words, σ is not normalized as a function of dimensionality. More precisely, with $\|x\|_1 = \|\tilde{x}\|_1$ and $\|x\|_2 = \|\tilde{x}\|_2$ and from the definition of σ follows that

$$\sigma(\tilde{x}) = \frac{\sqrt{n} - \|\tilde{x}\|_1/\|\tilde{x}\|_2}{\sqrt{n}-1} = \frac{\sqrt{n} - \sqrt{d} + \sqrt{d} - \|x\|_1/\|x\|_2}{\sqrt{n}-1} = \frac{\sqrt{n}-\sqrt{d}}{\sqrt{n}-1} + \sigma(x) \cdot \frac{\sqrt{d}-1}{\sqrt{n}-1}.$$

This expression always has to be applied when comparing the sparseness of vectors of different dimensionality to compensate for the normalization factors. With $n > d$ it is obvious that $\sigma(\tilde{x}) > \sigma(x)$, therefore adding zero entries increases sparseness with respect to σ. In fact, when appending arbitrarily many zero entries at the end of x, maximum sparseness is achieved regardless of the original vector x, that is $\lim_{n\to\infty} \sigma(\tilde{x}) = 1$. Additional theoretical properties of Hoyer's σ and how to achieve solutions to optimization problems subject to sparseness constraints induced by this sparseness measure are discussed comprehensively in Chapter 4.

2.3 Implicit Sparseness versus Explicit Sparseness in Linear Generative Models

Once an appropriate formal sparseness measure has been chosen, an objective function with incorporated sparseness constraints can be formulated to suit a specific task. In this section, an information compression task using a *linear generative model* is discussed. For this, let $x^{(1)}, \dots, x^{(M)} \in \mathbb{R}^d$ be M data points of dimensionality $d \in \mathbb{N}$, and let $h^{(1)}, \dots, h^{(M)} \in \mathbb{R}^n$ be corresponding *code words* of dimensionality $n \in \mathbb{N}$. The objective is to make the code words carry as much information about the input data as possible, that is $x^{(\mu)}$ shall be able to be reproduced from $h^{(\mu)}$ with good precision for all $\mu \in \{1, \dots, M\}$. In a linear model, *bases* or *basis images* $w^{(1)}, \dots, w^{(n)} \in \mathbb{R}^d$ exist such that the rule for decoding a code word $h \in \mathbb{R}^n$ becomes a linear combination, $h \mapsto \sum_{i=1}^n h_i w^{(i)} \in \mathbb{R}^d$. From this point of view, the model of Olshausen and Field as discussed in Section 2.1 is a linear generative one.

It is convenient to write the occurring vectors in the columns of individual matrices so that matrix notation can be used. Hence, let $X := \left(x^{(1)}, \dots, x^{(M)}\right) \in \mathbb{R}^{d \times M}$ denote the *data matrix*, $H := \left(h^{(1)}, \dots, h^{(M)}\right) \in \mathbb{R}^{n \times M}$ is the *matrix of code words* and $W := \left(w^{(1)}, \dots, w^{(n)}\right) \in \mathbb{R}^{d \times n}$ denotes the *matrix of bases*. The rule for decoding a code word $h \in \mathbb{R}^n$ is then a matrix-vector product, $h \mapsto Wh$. Using the squared error similarity measure, the *reproduction error* E_R measures the deviation between the reproduced samples and the original data points:

$$E_R := \sum_{\mu=1}^M \left\| x^{(\mu)} - Wh^{(\mu)} \right\|_2^2 = \|X - WH\|_F^2 \overset{!}{\to} \min_{W,H}.$$

Here, $\|\cdot\|_F$ is the Frobenius norm of a matrix, that is an entrywise continuation of the Euclidean norm of a vector, see Appendix A.

Principal Component Analysis (PCA) [28] is a well-known representative of linear generative models. It additionally provides a rule for encoding, namely $x \mapsto W^T x$ for $x \in \mathbb{R}^d$. As the matrix of bases is employed for both encoding and decoding, PCA forms a *symmetric auto-encoder*. In order to pose a well-defined optimization criterion, the bases are required to be orthonormal, that is $\left\langle w^{(i)},\ w^{(j)} \right\rangle = 0$ for all $i, j \in \{1, \dots, n\}$ with $i \neq j$ and $\left\langle w^{(i)},\ w^{(i)} \right\rangle = 1$ for all $i \in \{1, \dots, n\}$.

In the context of *sparse coding*, a different regularization technique is employed to ameliorate non-uniqueness. The matrix of bases is unconstrained in the original formulation, and the code words shall satisfy a sparseness constraint with respect to the employed sparseness measure $s\colon \mathbb{R}^n \to \mathbb{R}$. In a model with *explicit sparseness*, see [26], [29], and [30] for examples, all code words have to meet a certain sparseness degree $\kappa \in \mathbb{R}$:

$$E_R \overset{!}{\to} \min_{W,H} \text{ such that } s\left(h^{(\mu)}\right) = \kappa \text{ for all } \mu \in \{1, \dots, M\}.$$

Note that this is quite the converse of a basis pursuit optimization problem [31], where the objective is to minimize $s\left(h^{(\mu)}\right)$ for all $\mu \in \{1, \dots, M\}$, such that $X = WH$. On the contrary, when *implicit sparseness* is involved, a sparseness error function is combined in an additive manner with the reproduction error to yield the overall objective function:

$$E_R + \alpha \sum\nolimits_{\mu=1}^{M} s\big(h^{(\mu)}\big) \overset{!}{\to} \min\nolimits_{W,H}\,.$$

Here, $\alpha \in \mathbb{R}$ is a constant for weighting the sparseness error. This optimization problem can also be considered a generalization of basis pursuit de-noising [31]. Besides the model from Section 2.1, the models from [32] and [33] also feature an implicit sparseness constraint. Using the method of Lagrange multipliers [34], the explicit form can be cast in implicit form to yield the Lagrangian

$$E_R + \sum\nolimits_{\mu=1}^{M} \lambda_\mu \left(s\big(h^{(\mu)}\big) - \kappa \right),$$

where $\lambda_1, \dots, \lambda_M \in \mathbb{R}$ are the Lagrange multipliers. Unfortunately, it is intractable to assess the correct values of the Lagrange multipliers in practice. A common heuristics is to assume all Lagrange multipliers to be equal, that is $\alpha := \lambda_1 = \dots = \lambda_M$, and ignore the concrete value of κ, such that the implicit form is yielded. To find a good choice for α, a variety of experiments has to be conducted which involves a large amount of computation time.

Another disadvantage of the implicit form over the explicit form is that the target degree of sparseness κ is neglected. In the implicit form, only the sparseness of the code words with respect to sparseness measure s is maximized. The resulting sparseness depends on the choice of the trade-off between reproduction error and sparseness error. If α is chosen too small, the reproduction error dominates the overall error, and the resulting sparseness is very low. On the contrary, if α is set too large, the influence of the reproduction error on the overall error becomes insignificant. In that case, the code words exhibit high sparseness, but they carry only very little information on the original data set.

In a model with explicit sparseness constraints on the other hand, the code words are guaranteed mathematically to attain the target sparseness κ in a solution of the optimization problem. However, finding the optimal choice for κ involves a variety of experiments as well. Additionally, there is no theoretical evidence whether forcing all code words to have the same sparseness is beneficial, or if an inhomogeneous distribution would yield better results. In this work, mainly models with explicit sparseness constraints are developed to benefit from exact target sparseness.

Algorithmic minimization of the objective function is typically achieved through *gradient descent*. Here, the sparseness measure s has to be differentiable for the model with implicit sparseness constraints. If explicit sparseness constraints are involved, a *projected gradient descent* algorithm is employed. This differs from ordinary gradient descent in that a projection operator is applied after updating the degrees of freedom. This projection operator finds the closest instance of the degrees of freedom satisfying the explicit sparseness constraints. Therefore, sparseness is guaranteed to be loop-invariant. Further details on optimization can be found in Appendix C, and an elaborate description of the projection with respect to Hoyer's sparseness measure σ is given in Chapter 4.

Table 2.1: Overview of various models subject to the sparse information processing paradigm.

Method	Sparse connectivity	Sparse activity	Generative	Inference	Discriminative	Explicit sparseness	○ Batch / • Online
ICA		•	○	○			•
NMF	○		•				○
NMFSC	•	•	•			•	○
SOCP	•	•	•		○	•	○
SESM	•	•	•	•			•
PSD		•	•	•			•
SDL		•	•		•		○
DSC		•	•		○		•
OBD	•				•	•	○
SCFC	•	•	•	•	•	•	○
SMLP	•			○	•	•	•
SOAE	•	•	•	•	•	•	•

2.4 Related Work

This section reviews work that is related to the sparse information processing paradigm. In doing so, each approach is analyzed and its capabilities with respect to certain features are determined. A comprehensive description is given in Table 2.1.

The investigated features are given as follows. For sparse connectivity and sparse activity, the approach must address the properties as depicted in Figure 2.1 and Figure 2.2, respectively. For the approach to be generative, it must be designed to be able to compute approximations of the input data set using the code words. The inference capability is fulfilled when it is feasible to efficiently compute code words from the input samples. When the approach is suitable inherently for classification tasks, the discriminative property holds. The differentiation between implicit and explicit sparseness as discussed in Section 2.3 is included in the comparison as well. The final criterion distinguishes whether one sample at a time is treated in an online algorithm, or if the entire data set is processed in a batch manner.

2.4.1 Independent Component Analysis (ICA)

A method motivated by information theoretical arguments and commonly used for blind source separation tasks is *Independent Component Analysis* [35, 36]. Considering a data set sampled from a multivariate, zero mean random variable X, the problem is to find a linear transformation via a matrix A such that the rows of $H := AX$ are stochastically as independent as possible from each other. In this formulation, ICA only features inference but no generative capabilities, that is it is ignored whether X can be reproduced from H.

Another formulation is purely generative: Here, a mixing matrix W is sought such that $X = WS$ holds, where the entries of S are assumed to be mutually independent. By setting A to the pseudoinverse [27] of W, the inference-only problem can be solved from the generative model. It can be shown [37] that in the generative model W can also be computed as maximization of the likelihood that the data set is generated by W.

This, in turn, is equivalent to maximization of the sparseness of S, and in fact it has been shown that ICA computes filters that are similar to those of Olshausen and Field when applied to natural image patches [38, 39]. ICA therefore incorporates implicit sparseness constraints. Note that in the generative formulation as posed above, overcomplete representations are not possible without technical modifications to the architecture [40], as then the encoding matrix A cannot be computed from W.

2.4.2 Non-Negative Matrix Factorization (NMF)

Non-Negative Matrix Factorization [41, 42, 43] is a linear generative model with the distinctive feature that all occurring quantities should be non-negative. Let $X \in \mathbb{R}_{\geq 0}^{d \times M}$ be a data matrix of M non-negative samples of dimensionality d. NMF aims to find a factorization of X into a non-negative matrix of bases $W \in \mathbb{R}_{\geq 0}^{d \times n}$ and a non-negative matrix of code words $H \in \mathbb{R}_{\geq 0}^{n \times M}$ such that the reproduction error in Frobenius norm is minimized:

$$E_{\text{NMF}} := \|X - WH\|_F^2 \overset{!}{\to} \min_{W,H} .$$

Alternative formulations use the Kullback-Leibler divergence instead of the Frobenius norm as the measure for the reproduction error [43]. Note that E_{NMF} is convex in either W or H, but not in both simultaneously. This model has become popular due to simple multiplicative update rules that are applied to W and H in an alternating manner, guaranteed to lead to a local minimum of E_{NMF} [43].

One of the most significant advantages of NMF is that it is known to produce sparsely populated bases on certain data sets. Due to the non-negativity constraints, the model is allowed to perform additive operations only to reproduce the input data set. Thus cancellations in the entries of the reproduction cannot occur, leading to a parts-based representation [42]. In fact, when applied to the CBCL data set of face images, the bases resemble distinctive parts of faces, such as the eyes, nose and mouth [42].

However, there are data sets, even from the same category of problems, where NMF fails to produce sparse representations, such as the ORL face image data set [44, 26]. This is supposed to result from the enhanced complexity of the ORL data set compared to the CBCL data set, where the faces have been well aligned within their respective bounding boxes. By modification of the NMF objective function and algorithm to incorporate sparseness, a parts-based representation can be computed even for complex data sets [44, 26].

2.4.3 Non-Negative Matrix Factorization with Sparseness Constraints (NMFSC)

The idea of controlling the representation's sparseness degree within the NMF framework leads to the definition of *Non-Negative Matrix Factorization with Sparseness Constraints* [26]. Using Hoyer's sparseness measure σ, the objective is to minimize E_{NMF} under the explicit sparseness constraints

$$\sigma(We_i) \overset{!}{=} \sigma_W \text{ for all } i \in \{1, \dots, n\} \text{ and } \sigma(H^T e_i) \overset{!}{=} \sigma_H \text{ for all } i \in \{1, \dots, n\}$$

for fixed degrees of sparseness $\sigma_W, \sigma_H \in (0,\ 1) \subseteq \mathbb{R}$.

The sparseness of the individual bases, that is the quantities $We_i \in \mathbb{R}^d$ for $i \in \{1, \dots, n\}$, is controlled by σ_W. Hence, the sparseness of each base can be explicitly forced to achieve a controllable degree. The number σ_H controls the fraction of samples each base contributes to by imposing a sparseness constraint onto the rows of the matrix of code words, that is the quantities $H^T e_i \in \mathbb{R}^M$ for $i \in \{1, \dots, n\}$. A high value of σ_H indicates that each base influences the reproduction of a small number of samples only. Thus a representation is yielded where information is distributed evenly among the bases, resulting in sparse activity.

This linear generative model with explicit sparseness constraints is optimized using a projected gradient descent algorithm. The sparseness-enforcing projection operator that is employed for this is discussed in Chapter 4. Chapter 5 provides more details of the optimization algorithm, and proposes an extension of NMFSC that is suitable for real-time classification tasks. Section 5.1.2 gives arguments why NMFSC is not inherently suitable for classification tasks.

2.4.4 Supervised NMFSC with Second Order Cone Programs (SOCP)

Closely related with NMFSC, the authors of [29] cast the original optimization problem as a second order cone program and develop efficient algorithms to solve it. For increased discriminative capabilities, a new constraint is added. It enforces that every code word, that is a column He_j of the code word matrix, which belongs to class c is near the mean value μ_c of all code words belonging to the same class:

$$\left\|\mu_c - He_j\right\|_2 \leq \lambda \left\|\mu_c\right\|_1 \text{ for all } j \in \{1,\dots,M\} \text{ belonging to class } c.$$

Here, λ is used to relax this constraint that minimizes the within-class scatter, similar to Fisher NMF [45]. Using their second order cone framework, this additional constraint can be handled in an efficient way. However, classification capabilities are only incorporated implicitly in the model, classifiers have to be trained separately. See Section 5.1.2 for a discussion on this model's discriminative capabilities.

2.4.5 Sparse Encoding Symmetric Machine (SESM)

Sparse Encoding Symmetric Machine [33] is an architecture which involves an encoder and a decoder part. Like in Principal Component Analysis, both parts employ the same matrix of bases $W \in \mathbb{R}^{d\times n}$. Using a nonlinearity $g\colon \mathbb{R}^n \to \mathbb{R}^n$ and thresholds $\theta_{\text{enc}} \in \mathbb{R}^n$ for encoding and $\theta_{\text{dec}} \in \mathbb{R}^d$ for decoding, the model dynamics are determined by two functions:

$$\begin{aligned} f_{\text{enc}}\colon \mathbb{R}^d \to \mathbb{R}^n, &\quad x \mapsto W^T x + \theta_{\text{enc}}, \text{ and} \\ f_{\text{dec}}\colon \mathbb{R}^n \to \mathbb{R}^d, &\quad h \mapsto Wg(h) + \theta_{\text{dec}}. \end{aligned}$$

The compatibility between a sample $x \in \mathbb{R}^d$ and a potential decomposition $h \in \mathbb{R}^n$ is measured by the energy function $E(x,h) := \alpha_e \left\|h - f_{\text{enc}}(x)\right\|_2^2 + \left\|x - f_{\text{dec}}(h)\right\|_2^2$, which is a sum of reproduction error and inference error. Using the slog function as sparseness penalty term, the objective function

$$E_{\text{SESM}} := E(x,h) + \alpha_s \operatorname{slog}(h) + \alpha_r \left\|W\right\|_1 \overset{!}{\to} \min\nolimits_{W,h}$$

is minimized in a sample after sample fashion via alternating gradient descent on h for finding the optimal decomposition and on the remaining parameters to update the machine. In this context, α_e, α_s and α_r are constants for weighting the individual penalty terms.

2.4.6 Predictive Sparse Decomposition (PSD)

In contrast to SESM, *Predictive Sparse Decomposition* [46] uses two different filter matrices for encoding and decoding, $W_{\text{enc}} \in \mathbb{R}^{n\times d}$ and $W_{\text{dec}} \in \mathbb{R}^{d\times n}$, respectively. Inference involves a nonlinearity using the mapping $x \mapsto \operatorname{diag}(p) \cdot \tanh\left(W_{\text{enc}}x + \theta_{\text{enc}}\right)$, where $p \in \mathbb{R}^n$ compensates for the scaling of inferred decompositions. Optimization of the objective function

$$\begin{aligned} E_{\text{PSD}} := {} & \left\|x - W_{\text{dec}}h\right\|_2^2 + \lambda \left\|h\right\|_1 \\ & + \alpha \left\|h - \operatorname{diag}(p) \cdot \tanh\left(W_{\text{enc}}x + \theta_{\text{enc}}\right)\right\|_2^2 \overset{!}{\to} \min\nolimits_{W_{\text{enc}}, W_{\text{dec}}, h, p, \theta_{\text{enc}}} \end{aligned}$$

takes place by alternating stochastic gradient descent. Non-sparseness of code words h is penalized implicitly by the additive term of $\|h\|_1$ scaled by a constant $\lambda > 0$. Since non-sparseness of the filter matrices is neither penalized nor preferred, sparse connectivity is not featured in this model.

2.4.7 Supervised Dictionary Learning (SDL)

Two approaches that add discriminatory capabilities to the sparse coding model were proposed by [47], both called variations of *Supervised Dictionary Learning*. The simpler approach of the two was proposed as minimization of

$$\begin{aligned} E_{\text{SDL-G}} :=& \operatorname{Fermi}\left(t \cdot (w_{\text{out}}^T h + \theta_{\text{out}})\right) + \lambda_2 \left\| (w_{\text{out}}^T,\ \theta_{\text{out}})^T \right\|_2^2 \\ &+ \lambda_0 \|x - Wh\|_2^2 + \lambda_1 \|h\|_1 \overset{!}{\to} \min_{W,\,h,\,w_{\text{out}},\,\theta_{\text{out}}}. \end{aligned}$$

Here, $x \in \mathbb{R}^d$ is the input sample, $W \in \mathbb{R}^{d \times n}$ is the matrix of bases and $h \in \mathbb{R}^n$ is the associated code word. Classification is done using a linear model in a logistic regression setting using the Fermi transfer function, see Section C.5, where the classification path involves the teacher signal $t \in \mathbb{R}$ associated with x, and degrees of freedom $w_{\text{out}} \in \mathbb{R}^n$ and $\theta_{\text{out}} \in \mathbb{R}$. The constants $\lambda_0, \lambda_1, \lambda_2 \in \mathbb{R}$ control the weight of the reproduction error, the sparseness of the code words and the regularization on the classification model's parameters, respectively. Inference of sparse code words is achieved by clamping all parameters but h, and then searching for a code word that minimizes $E_{\text{SDL-G}}$. The degrees of freedom are optimized by alternating gradient descent. First, optimal code words h are inferred, and then the remaining degrees of freedom W, w_{out} and θ_{out} are updated all together. As explicit inference of code words is not intended, a cost-intensive optimization problem has to be solved for every sample.

The more complicated approach aims at maximizing the difference of the loss function with respect to the correct teacher signal and the negated teacher signal. Consider

$$S := \operatorname{Fermi}\left(t \cdot (w_{\text{out}}^T h + \theta_{\text{out}})\right) + \lambda_0 \|x - Wh\|_2^2 + \lambda_1 \|h\|_1,$$

which is $E_{\text{SDL-G}}$ except for the classification path regularization. While keeping all other degrees of freedom fixed, let h_y^* denote the minimum of S with respect to h with assumed teacher signal y, that is h_t^* is the code word for the correct teacher signal and h_{-t}^* is the code word for the confused teacher signal. With $\mu \in [0,\ 1]$ controlling the trade-off between reproduction error and classification error, the overall objective function

$$E_{\text{SDL-D}} := \mu \cdot \operatorname{Fermi}(S|_{h_t^*} - S|_{h_{-t}^*}) + (1-\mu) \cdot S|_{h_t^*} + \lambda_2 \left\| (w_{\text{out}}^T,\ \theta_{\text{out}})^T \right\|_2^2 \overset{!}{\to} \min_{W,\,h,\,w_{\text{out}},\,\theta_{\text{out}}},$$

is optimized using an alternating gradient descent procedure.

2.4.8 Differentiable Sparse Coding (DSC)

The *Differentiable Sparse Coding* model [48] minimizes a squared reproduction error and uses the Kullback-Leibler divergence, see Section C.6, as implicit sparseness penalty term:

$$E_{\text{DSC}} := \|x - Wh\|_2^2 + \lambda \operatorname{KL}(\kappa e, h) \overset{!}{\to} \min_{W,h}.$$

Here, e is the vector of only ones. $\kappa \in \mathbb{R}$ is chosen such that $\|\kappa e\|_1 = \mathbb{E}(\|h\|_1)$, that is the L_1 norm of κe should match the expected L_1 norm of the code words. The matrix of bases is optimized via stochastic gradient descent, and the code words are optimized using exponentiated gradient descent [49]. The authors claim that their sparseness penalty term is smoother than an L_1 norm constraint, and hence the code words are more stable to subtle changes of the input. They demonstrate that their method is beneficial in a classification task, especially when adding a second phase backpropagation step. However, discriminative capabilities are not incorporated explicitly in the model.

2.4.9 Optimal Brain Damage (OBD)

Optimal Brain Damage [50] is not a sparse coding model, but a technique devoted to sparse connectivity in Artificial Neural Networks for classification. The idea is to remove synaptic connections after a network has been trained to minimize the classification error. After this pruning, the network is re-trained using the remaining connections to make it re-adapt to the classification task. The key feature of OBD lies in the determination which connections should be removed.

The authors propose to use a measure of *saliency*, which estimates the impact of one synaptic connection on the training error. The smaller the saliency is, the more likely the network will produce the same output without the corresponding connection. Instead of measuring saliency by the magnitude of the synaptic connection, the authors propose to directly consider the objective function. The approach of temporarily deleting one parameter after another, and then re-evaluating the objective function on the whole data set is intractable.

Therefore, the proposal is to consider the effect of small changes of the weight vector on the overall error. Let $E\colon \mathbb{R}^n \to \mathbb{R}$ be the classification error, and let $w \in \mathbb{R}^n$ be a vector with all the weights of the network. Let $\delta \in \mathbb{R}^n$ be a small perturbation of the weights. Assuming E is locally quadratic, a Taylor expansion yields $E(w+\delta) \approx E(w) + E'(w)\delta + 1/2 \cdot \delta^T E''(w)\delta$, where $E'(w) \in \mathbb{R}^{1 \times n}$ is the gradient and $E''(w) \in \mathbb{R}^{n \times n}$ is the Hessian of E evaluated at w. Because pruning takes place on an already trained network, it can be assumed that the network's weights are located in a local minimum of the objective function, that is $E'(w) = 0$. To avoid having to compute the full Hessian whose size is quadratic in the number of weights, its non-diagonal entries are neglected. Hence let $g \in \mathbb{R}^n$ be the diagonal of $E''(w)$. Using these simplifications, the change in the objective function when applying the perturbation δ to the weights becomes $E(w+\delta)-E(w) \approx 1/2 \cdot \sum_{i=1}^{n} \delta_i^2 g_i$. If the jth weight, $j \in \{1,\dots,n\}$, is to be completely removed, then $\delta_j = -w_j$ must hold. Hence the saliency becomes $s_j := 1/2 \cdot g_j w_j^2$.

Using this information, the OBD technique consists of training the network and pruning connections with low saliency in an alternating manner. For computation of the saliency, the diagonal of the Hessian has to be computed as well. This can for example be done using a modified variant of the backpropagation algorithm [50]. In the special case of a two-layer MLP, simple expressions for the Hessian's diagonal can be derived, see Section 6.2.

OBD is extended by the *Optimal Brain Surgeon (OBS)* method [51]. Here, it is not assumed that the Hessian of the classification error is diagonal. Using the Taylor series of E, this method yields a more general measure of saliency, $\tilde{s}_j := 1/2 \cdot \tilde{g}_j w_j^2$, where $\tilde{g}_j := \left(e_j^T E''(w)^{-1} e_j\right)^{-1}$ is the reciprocal of the jth entry on the main diagonal of the Hessian's inverse. When $E''(w)$ is diagonal, then clearly $\tilde{g}_j = g_j$ and hence $\tilde{s}_j = s_j$, and the saliency of OBD and OBS are identical.

Further, an optimal weight change formula using the entire inverse of the Hessian was derived by [51] that can be employed in updating the network after deletion of a synaptic connection so as to minimize the objective function increase after pruning. According to the authors, this alleviates the need for retraining that OBD requires. Despite this advantage, usage of OBS is not realistic in scenarios with neural networks that have a large number of synaptic connections. This is because storing and computing the entire Hessian requires at least quadratic complexity in the number of the degrees of freedom, and this is intractable in practice. If [51] would have used no special matrix inversion formula, time complexity would even be up to cubic for computation of the Hessian's inverse.

2.4.10 New Models for Sparse Information Processing: SCFC, SMLP and SOAE

The final three approaches from Table 2.1 are novel methods proposed in this work. SCFC stands for Sparse Coding for Fast Classification, which extends NMFSC by inference and classification capabilities. It is discussed in Chapter 5. Chapter 6 proposes a Multi-Layer Perceptron with sparse connectivity (SMLP), which is able to handle very large data sets and achieve excellent classification capabilities, while the computational complexity is very low during the recall phase. Supervised Online Auto-Encoder (SOAE) uses Hoyer's sparseness-enforcing projection operator as neural transfer function to realize a hybrid of an auto-encoder network and a two-layer MLP, see Chapter 7. All three approaches have their individual advantages on both the theoretical and the practical side, which will be discussed comprehensively in the individual chapters. A prerequisite for this is an elaborate analysis of the projection onto sparseness constraints induced by Hoyer's σ which is given in Chapter 4, since the newly proposed methods rely on explicit sparseness constraints.

2.5 Discussion

This chapter introduced the sparse information processing paradigm for artificial neural information processing systems. In doing so, the two key properties sparse connectivity and sparse activity were characterized. It was noted that these properties can be observed in mammalian brains, and conversely that restricting an artificial system to produce sparse solutions leads to phenomena also encountered in mammalian brains. Then, several formal sparseness measures were discussed to serve as mathematical tools in obtaining sparse representations. It was concluded that Hoyer's sparseness measure is the most appropriate because it is a smooth relaxation of the natural L_0 pseudo-norm. The consideration of the pros and cons of explicit sparseness constraints have shown that from a user's point of view, explicit constraints are easier to handle as they yield representations with a guaranteed sparseness degree.

A review of existing models that support sparse information processing subject to general desirable properties has shown that none possesses all properties and that most models are in fact quite cumbersome. For example, SOCP implements discriminative capabilities as supervised clustering, which cannot be considered a state-of-the-art approach. SESM and PSD are not suited to classification tasks and only make use of implicit sparseness constraints, such that the outcome of learning depends strongly on trade-off parameters that are non-trivial to tune. SDL provides no easy way of inferring sparse code words from arbitrary input data, such that for every sample that should be processed a cost-intensive optimization has to be carried out during run-time. DSC simply replaces an L_1 norm sparseness penalty with the Kullback-Leibler divergence. This replacement seems ad hoc, as this does not result in sparsely populated vectors with a low L_0 pseudo-norm but rather in vectors where all entries are nonzero and only a few have a very strong magnitude compared to others. The saliency measure of OBD is merely an approximation with strong assumptions. Here, the neural network always has to be trained until convergence to ensure that the gradient of the objective function vanishes before synaptic connections can be pruned. Further, neglecting off-diagonal entries of the Hessian seems to oversimplify the problem, while computation of the entire Hessian as required by OBS is intractable for reasonable-sized neural networks.

This discussion clearly underlines the need for a comprehensive theory of sparseness in Artificial Neural Networks that can be used both for generative and discriminative tasks and where the way in which sparseness is achieved has a strong theoretical foundation. This work provides such a theory by rigorous analysis of an appealing sparseness measure and its incorporation in comprehensive models of neural information processing. It is further demonstrated by empirical evidence that these models exploit the advantage the sparseness concept provides and thus attain superior performance compared to non-sparse models.

3 A Benchmark Data Set for Classification Tasks

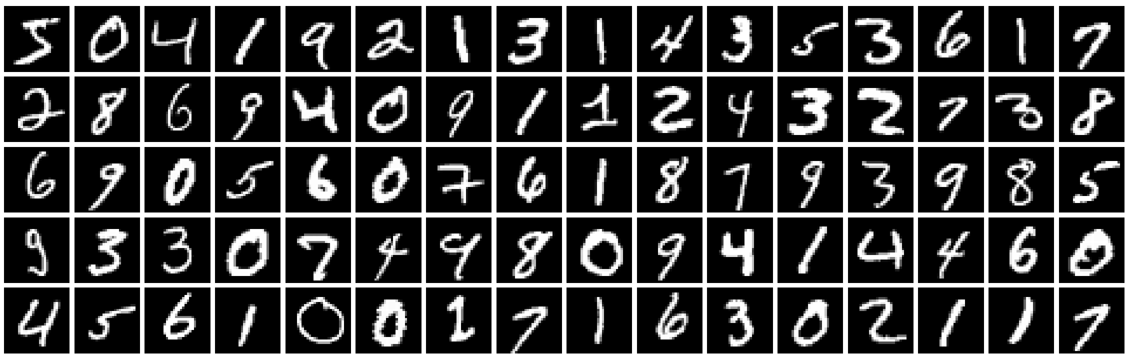
The data set of natural image patches as described in Section 2.1 does not contain any label information and is therefore not suited for classification tasks. In this work, the *MNIST database of handwritten digits* [52] is used to evaluate learning algorithms. As the MNIST database is a popular benchmark data set, numerous results have already been reported in the literature, such that an extensive comparison is possible. Learning algorithms subject to the sparse information processing paradigm are proposed in Chapters 5, 6, and 7. These chapters are concluded with experiments on the MNIST data set to assess classification capabilities. Finally, Chapter 8 contains a comprehensive evaluation of all algorithms and in addition a review of the computational complexity necessary for classification of unknown samples.

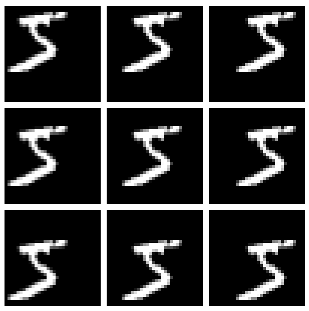
The MNIST database consists of 70 000 samples, divided into a learning set of 60 000 samples and an evaluation set of 10 000 samples. Each sample represents a digit of size 28×28 pixels and has a class label from $\{0, \ldots, 9\}$ associated with it. Therefore the input dimensionality is $28^2 = 784$, and the output dimensionality is 10. The first samples of the learning set are depicted in Figure 3.1a. In the experiments, the classification error is always given as a percent of all 10 000 evaluation samples by convention, hence 0.01% corresponds to a single misclassified digit. Although in real-world applications for handwritten digit recognition it is beneficial to reject samples rather than force a classification decision if the confidence of the classifier is below a certain threshold [53, 54], this approach is not used frequently in the scientific literature and hence was not applied in this work to allow for a fair comparison.

This chapter is structured as follows. First, the generation of virtual training samples is discussed to augment the original learning set. Next, basic unsupervised algorithms as described in Chapter 2 are used to provide a first analysis of the MNIST data set. The chapter concludes with a discussion and various full-page graphics that depict the results of the unsupervised techniques which are useful to assess the characteristics of this data set.

3.1 Augmented Learning Sets

This section describes certain methods for increasing the learning set size by computing virtual training samples from the original learning set. The purpose is to enhance the generalization


(a) The first 80 samples of the MNIST learning set.


(b) The first sample of the data set jittered in eight directions (dramatization).

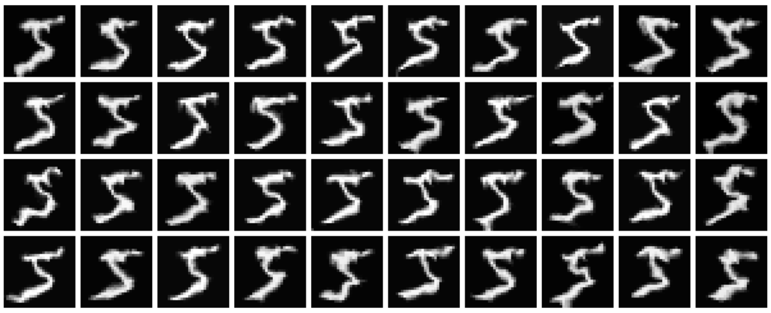

(c) Artificial learning samples created from the first sample of the data set using elastic distortions. The distortion effect was amplified to be clearly visible in the illustration.

Figure 3.1: Samples from the MNIST data set [52]. Subfigures (b) and (c) sketch the appearance of artificial training samples created from original samples.

capabilities of the classifiers trained on these augmented learning sets. The evaluation set was left unchanged throughout this work to provide results that can be compared to the literature. The only pre-processing applied to the data set is an affine-linear transformation of all individual samples to obtain zero mean and unit variance.

To generate the original data set, the digits were placed based on their barycenter [52]. Because of sampling and rounding errors, the localization uncertainty can hence be assumed to be less than one pixel in both directions. To account for this uncertainty, a variant of the learning set was generated by augmenting it with samples jittered in each of eight possible directions by one pixel. This yielded 540 000 samples for learning in total, that is the learning set size increased by factor 9. A sketch of the jitter is depicted in Figure 3.1b, where the effect was exaggerated for illustrational purposes. This augmented learning set is denoted the *jittered MNIST learning set* in this work.

As noted by [55], the learning problem is not permutation-invariant due to the jitter, as information on the neighborhood of the pixels is implicitly incorporated in the learning set.

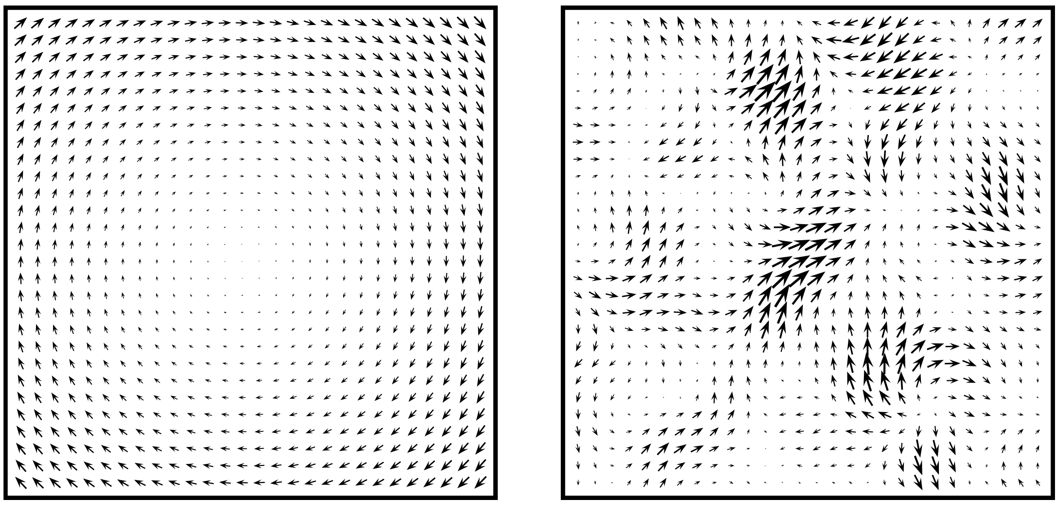

(a) Rotational displacement. (b) Elastic distortion.

Figure 3.2: Displacement fields for creating artificial training samples to augment the learning set. The arrows indicate orientation and magnitude of pixel-wise displacements.

However, classification results improve dramatically when this prior knowledge is used. This was demonstrated by [56] using the virtual support vector method, which improved a Support Vector Machine with polynomial kernel of degree five from an error of 1.4% to 1.0% by jittering the support vectors by one pixel in four principal directions. This result was extended by [57], where a Support Vector Machine with a polynomial kernel of degree nine was improved from an error of 1.22% to 0.68% by jittering in all possible eight directions.

Further improvements can be achieved by generation of artificial training samples using *displacement fields* [58]. Here, two real-valued matrices with the same size as the input image are generated. One matrix determines the displacement in the horizontal direction, and the other matrix determines the displacement in the vertical direction. A warped image is then computed from the original image by pixel-wise application of the displacements, using bilinear interpolation for sub-pixel access to the input image. A displacement field for rotation around the image center is depicted in Figure 3.2a. The magnitude of the displacement increases with the distance from the rotation center.

Elastic distortions are special displacements that model uncontrolled oscillations of the hand muscles [58]. The entries of the matrices for horizontal and vertical displacements are first initialized randomly by sampling from a uniform distribution with values in $[-1,\ 1] \subseteq \mathbb{R}$. The matrices are then low-pass filtered using a convolution with a Gaussian filter with standard deviation $s \in \mathbb{R}$. Finally, all entries are scaled by a constant gain factor $\alpha \in \mathbb{R}$. An example for such a displacement field is depicted in Figure 3.2b. The smoothness of the displacement field increases with the Gaussian's standard deviation. Effect intensity is controlled via the gain factor. Example images of elastic distortions applied to a sample from the MNIST learning set are shown in Figure 3.1c. A large variety of artificial samples can be generated by random resampling of the displacement matrices prior to the convolution and the scaling. In this work, 25 elastically distorted samples were generated from every sample of the original learning set using the parameters $s := 2$ and $\alpha := 8$. Additionally, all samples were jittered by one pixel, yielding a learning set with $9 \cdot 25 \cdot 60\,000 = 13\,500\,000$ samples in total, which is called the *elastically distorted learning set*.

As this technique simulates hand muscle oscillations, it is ideal for generating large learning sets of handwritten digits. Using a two-layer MLP with 800 hidden units, the evaluation error on the MNIST database can be decreased from 1.1% using samples generated by affine transformations to 0.7% by using elastic distortions [58]. Since new samples can always be generated from new random numbers, it is theoretically possible to create learning sets with an infinite number of distinct samples. This is beneficial when very big neural networks with a large number of adaptable weights are trained. For example, a six-layer MLP with twelve million synaptic connections can achieve an evaluation error as low as 0.35% using elastically distorted samples [59]. The current record error of 0.23% is held by an approach that combines distorted samples with a committee of Convolutional Neural Networks [60]. This is an architecture that has been optimized exclusively for input data representing images, where the neighborhood of the pixels is hard-wired in the classifier. A more detailed discussion of a variety of learning algorithms applied to MNIST is given in Chapter 8.

3.2 Properties of the MNIST Data Set

This section analyzes fundamental properties of the MNIST data set. In doing so, several unsupervised learning algorithms are applied to discover the basic structure of the samples. Further, the sparseness of the samples is investigated to estimate the parameters for learning algorithms that support sparse connectivity. Finally it is demonstrated that while not obvious, information on the pixel topology can also be extracted from pixel intensities.

3.2.1 Principal Component Analysis

As a first experiment, Principal Component Analysis [28] was applied to the entire MNIST database. The first 160 principal components are shown in Figure 3.7a, in descending order of importance for reproduction. It can be seen that low frequencies are predominant in the most important principal components. As component relevance decreases, the frequencies of the components increase, until only small checkerboard-like filters remain.

In a second experiment, the data set was split depending on the labels of the samples to form ten data sets, one per class. Principal Component Analysis was applied independently to all ten data sets. This yielded the bases shown in Figure 3.7b. While the underlying class of each data set can easily be recognized from the bases, the intraclass variance appears very high.

Figure 3.3 shows how many principal components are required to achieve a certain degree of information preservation in the projections. When the first 20 principal components are used, then roughly 75% of the information with respect to the squared error similarity measure is available. Furthermore, 120 principal components are sufficient for 95% of information preservation, and 468 principal components are needed for 99.9% reproduction capabilities. As 468 bases are only about 60% of the total 784 bases, the intrinsic dimensionality of the manifold the samples reside on is rather low. Thus, when 1000 filters are used for classification of the MNIST digits, the internal representation can be considered two times overcomplete.

3.2.2 Independent Component Analysis

Independent Component Analysis was used to further analyze the data set, using an implementation of the FastICA algorithm [36]. The algorithm computes a mixing matrix and an encoded representation of the original samples with respect to the independent components. Figure 3.4 shows a histogram of the sparseness of each independent signal as measured over the entire data set. It can be seen that almost the entire range of sparse activity is present, ranging from 0.10 to 1.0. The distribution is bimodal, with peaks at sparsenesses of about 0.30 and 0.95. A subset of the resulting bases, the columns of the mixing matrix, is depicted in Figure 3.8. Here, the bases were sorted according to their induced sparseness of activity. For low sparseness, the bases resemble individual strokes and the digit "eight" can be recognized in some of the bases. The bases which induce high sparseness resemble whole digits.

3.2.3 Non-Negative Matrix Factorization

Known to produce sparsely connected representations on some data sets, Non-Negative Matrix Factorization [41] was applied to the MNIST database. A subset of the resulting bases is depicted in the top row of Figure 3.9. Clearly, Non-Negative Matrix Factorization does produce a sparsely connected representation, but unfortunately all bases resemble blobs without particular structure and do not provide any deeper insight into the data set. In subsequent experiments, Non-Negative Matrix Factorization with Sparseness Constraints [26] was used where sparseness of activity was enforced but sparseness of connectivity was ignored. Subsets of resulting bases for various choices of the sparseness degree of activity σ_H are depicted in Figure 3.9. For $0.10 \leq \sigma_H \leq 0.55$, the results do not differ significantly from those of plain Non-Negative Matrix Factorization. When σ_H equals 0.60 or 0.65, the blobs morph into individual strokes of the handwritten digits. For $\sigma_H \geq 0.70$, the bases resemble individual digits and hence do not feature sparse connectivity. For the bases depicted in Figure 3.10, the sparseness degree of activity was set to $\sigma_H := 0.80$ and the sparseness of connectivity σ_W was

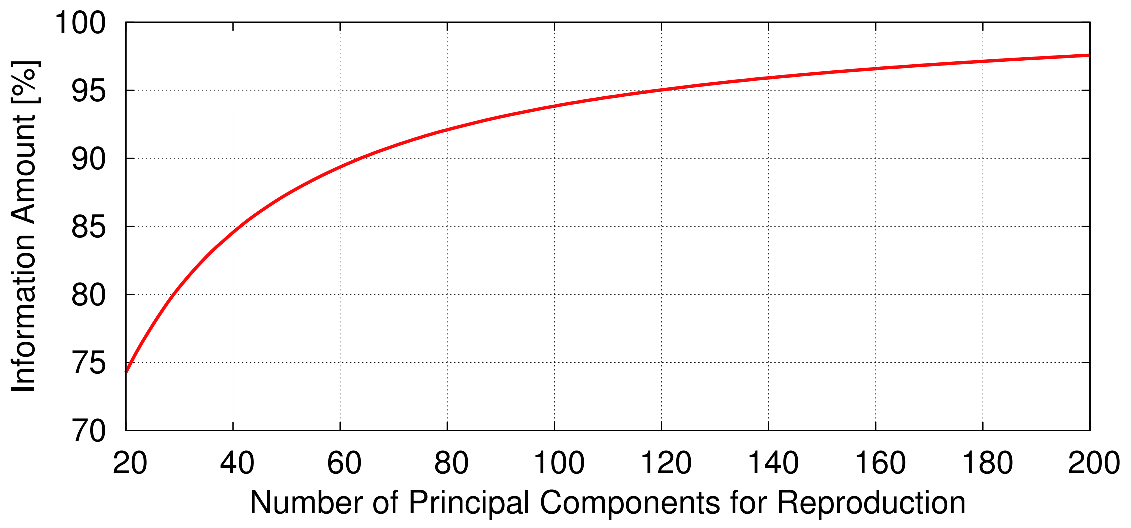


Figure 3.3: Number of principal components required so that the projection carries a specified amount of the original data information with respect to the squared error. For example, if 120 principal components are used then 95% of the original information is still accessible.

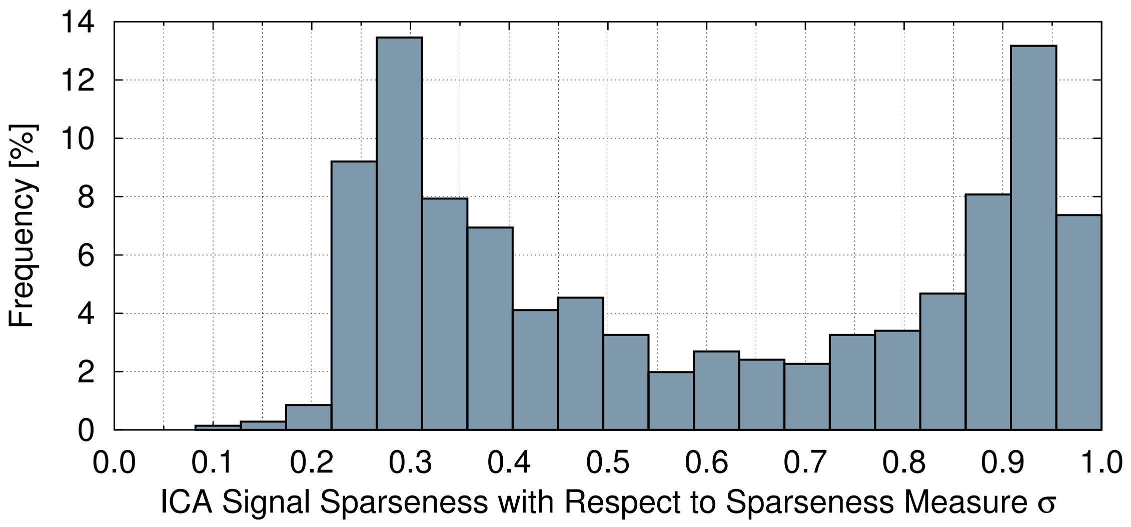


Figure 3.4: Histogram of the sparseness of activity for the independent signals discovered by Independent Component Analysis. Almost the entire range of possible sparseness values is present. The distribution is bimodal with peaks at about 0.30 and 0.95.

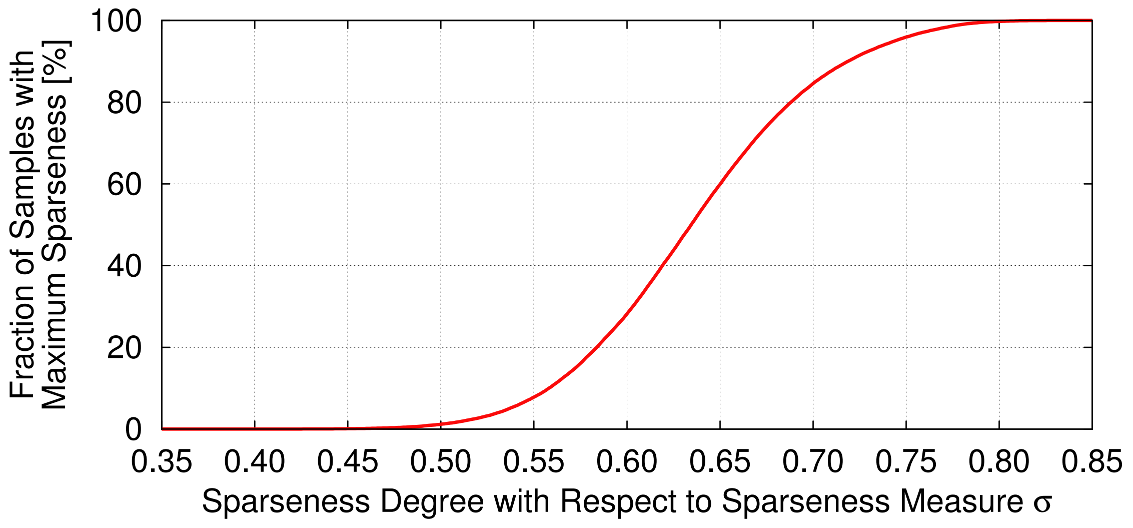


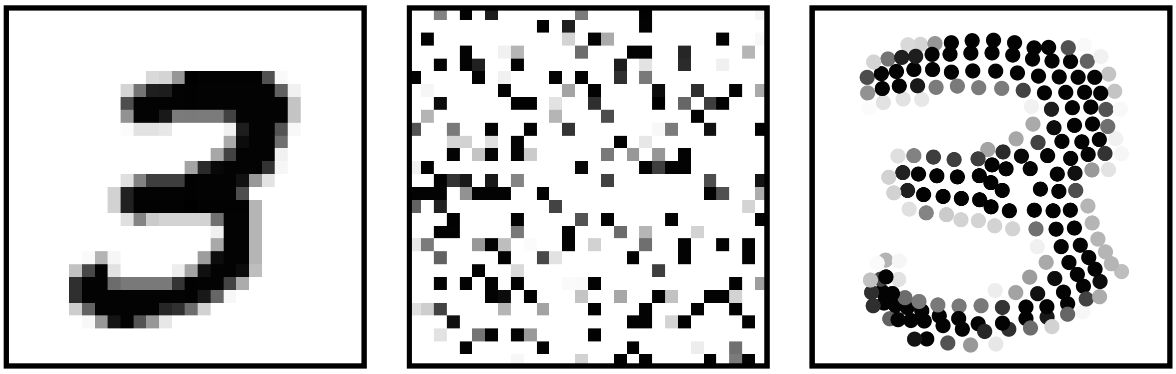
Figure 3.5: Empirical cumulative distribution function of the sparseness of the samples from the MNIST database. About 60% of all samples have a sparseness of less than 0.65 with respect to Hoyer's sparseness measure σ, and about 96% of all samples have a sparseness of less than 0.75.

varied. The entire variety of possible bases could be achieved from this experiment, as low values of σ_W induce bases that look like entire digits, medium values like $\sigma_W = 0.75$ produce a stroke-based representation and for high values only local blobs result.

It can therefore be concluded that in contrast to Independent Component Analysis, where the degree of sparse activity is merely random, Non-Negative Matrix Factorization with Sparseness Constraints can be used to produce representations where sparseness degrees are explicitly controllable. By experimentation with the parameter σ_H, representations tailored for specific use cases can be generated. It is crucial not to set σ_H too low, because then hardly any useful structure can be observed, while setting σ_H too high results in a method similar to fuzzy clustering which does not result in a distributed representation. Further, tuning of σ_W can be used to explicitly control the number of non-vanishing entries in the resulting bases.

3.2.4 Sparseness of Individual Samples

The sparseness of the samples of the MNIST database was analyzed to find a feasible range for sparse connectivity. The empirical cumulative distribution function of the sample sparseness with respect to Hoyer's sparseness measure σ is depicted in Figure 3.5. It is evident that 96% of all samples have a sparseness of at most 0.75. Moreover, from Figure 3.10 can be seen that lower values do not yield a parts-based representation. Therefore, if the sparseness of connectivity of the models developed in this work, by convention denoted by parameter



(a) Original sample $x \in \mathbb{R}^{784}$. (b) Sample $y := P_\tau x \in \mathbb{R}^{784}$ after pixels have been permuted. (c) Reconstruction using learned pixel arrangement.

Figure 3.6: Sketch of a method to discover pixel topology. A random permutation is applied to the (a) original samples to yield (b) permuted samples. The permuted samples are then fed into an algorithm that estimates the arrangement of the individual pixels in two-dimensional space. By pigmenting the found pixel positions with the permuted pixel intensities a (c) reconstructed image is yielded. This reconstruction may not be an exact copy of the original image, but the main characteristics are still visible. Pixel values were inverted for illustrational purposes.

σ_W, is set to at least 0.75 when the MNIST data set is used, a representation with true sparse connectivity will result. Explicitly forcing sparse connectivity with a degree lower than 0.75 is not advisable because that results in bases which are difficult to interpret since they are then mixtures of numerous samples.

3.2.5 Reconstruction of Pixel Topology

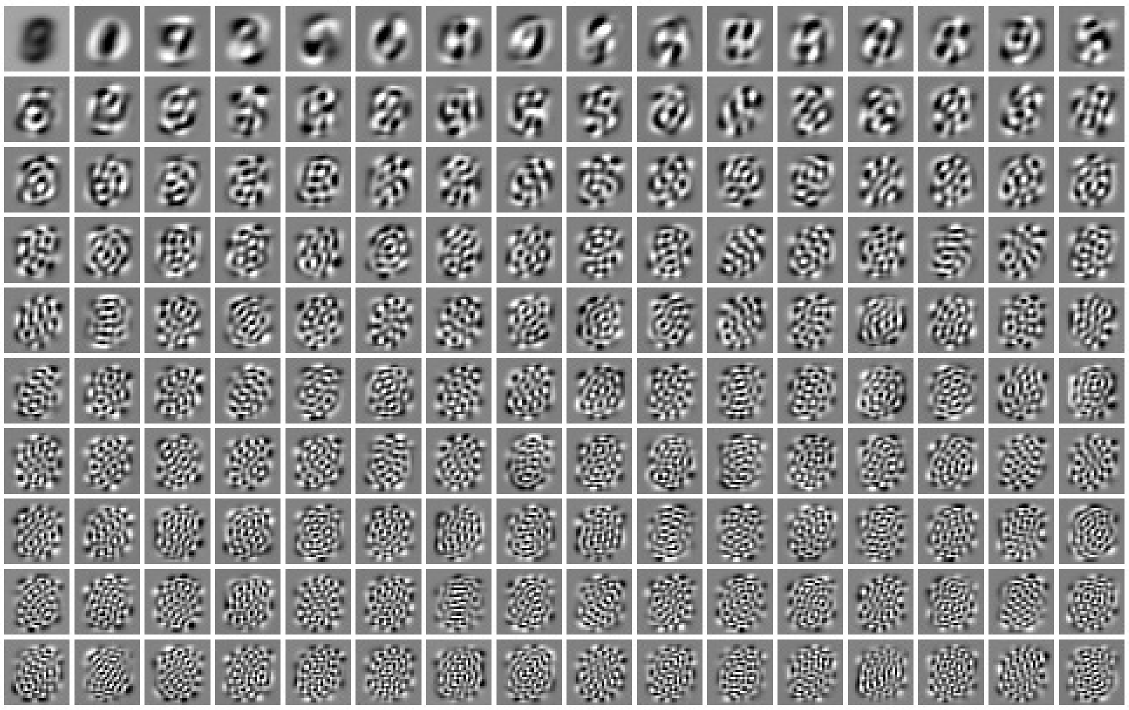
As mentioned before, classification results can be improved by incorporation of prior knowledge, for example by using the jittered learning set which is not permutation-invariant. However, it was discovered that most of the information of the position of the individual pixels is already encoded in the pixel intensities [61]. In the experiments described therein, a random permutation $\tau \in S_{784}$ was chosen and for every sample $x \in \mathbb{R}^{784}$ a pixel-wise permuted sample $y := P_\tau x \in \mathbb{R}^{784}$ was computed. Here, P_τ is the permutation matrix associated with τ. After this pre-processing, the image information is no longer human-readable, see Figure 3.6b.

The permuted samples were then input to a manifold learning algorithm to estimate the original pixel arrangement. For this, a distance matrix using a distance feature had to be computed for the manifold embedding. In [61] the absolute value of the correlation coefficient of the pixel intensities over the entire data set was used as distance feature. This was motivated by noting that it is very likely that two neighboring pixels are either almost always on or off at the same time, or their state is inverse when both pixels are located at an edge in the image. When the manifold learning algorithm is used to compute a two-dimensional embedding, the pixels are arranged in two-dimensional space according to the distance matrix. It should be stressed that the permutation applied to the pixels is not input to the learning algorithm, and that the distance matrix is the only input. The resulting pixel placement can then be pigmented using the intensity values of the pixels to yield a reconstruction, as shown in Figure 3.6. Although the reconstruction is not perfect, a human observer is able to recognize the original sample.

For this work, the experiment was repeated and its results confirmed using an implementation of the Isomap algorithm [62]. The results show that pixel neighborhood information can be restored from pixel intensities. However, learning algorithms can make better use of this information when it is given more explicitly, for example when a jittered learning set is used or a classification architecture that is inherently designed for two-dimensional inputs like Convolutional Neural Networks because then pixel topology information does not have to be discovered first.

3.3 Discussion

This chapter introduced the MNIST database of handwritten digits, which is a benchmark data set for learning algorithms, in particular for classification tasks. Two methods for creating virtual samples to augment the original learning set were described, jittering the samples and using elastic distortions. As will be demonstrated in subsequent chapters, using these extended learning sets results in improved generalization performance. The chapter was concluded by a preliminary analysis of the MNIST database, mainly by application of standard unsupervised learning algorithms to discover the structure of the data. This showed that the intraclass variance of the samples is rather high, which renders the classification problem all but trivial. Moreover, it was shown that 468 principal components are sufficient to represent 99.9% of the information present in the data set, so that 1000 filters correspond to a two times overcomplete representation. The analysis also has shown that Independent Component Analysis and Non-Negative Matrix Factorization fail to produce useful representations. However, when using Non-Negative Matrix Factorization with Sparseness Constraints where sparseness degrees are easily controllable, the representations can be tuned so that more of the structure of the data set becomes apparent. Further, a minimum sparseness degree of 0.75 for connectivity was derived by evaluation of the sample sparseness. In conclusion, it was shown that pixel neighborhood information is already present in the pixel intensities. However, making this information explicitly available from the start simplifies the learning task, as it is non-trivial to discover pixel topology from the raw data.


(a) The first 160 principal components of the entire data set.

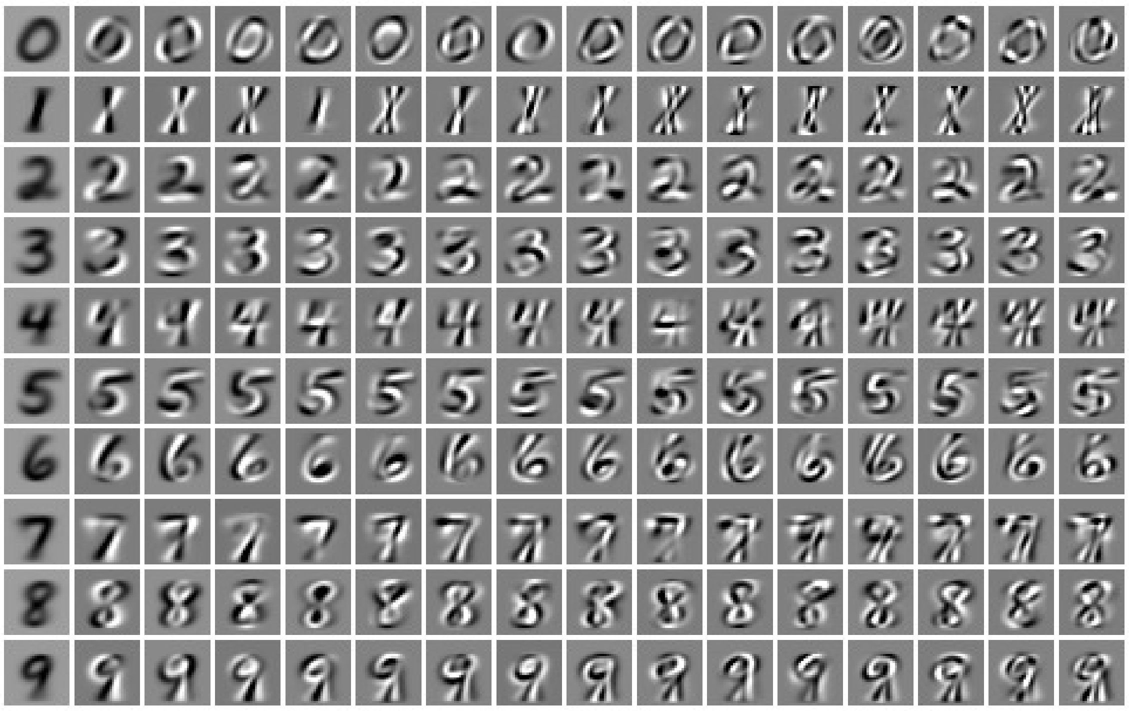

(b) The first 16 principal components for the ten distinct classes, where each row refers to a different class.

Figure 3.7: Results of Principal Component Analysis applied to the MNIST database of handwritten digits. The resulting principal components are shown, in decreasing order of relevance for reproduction.

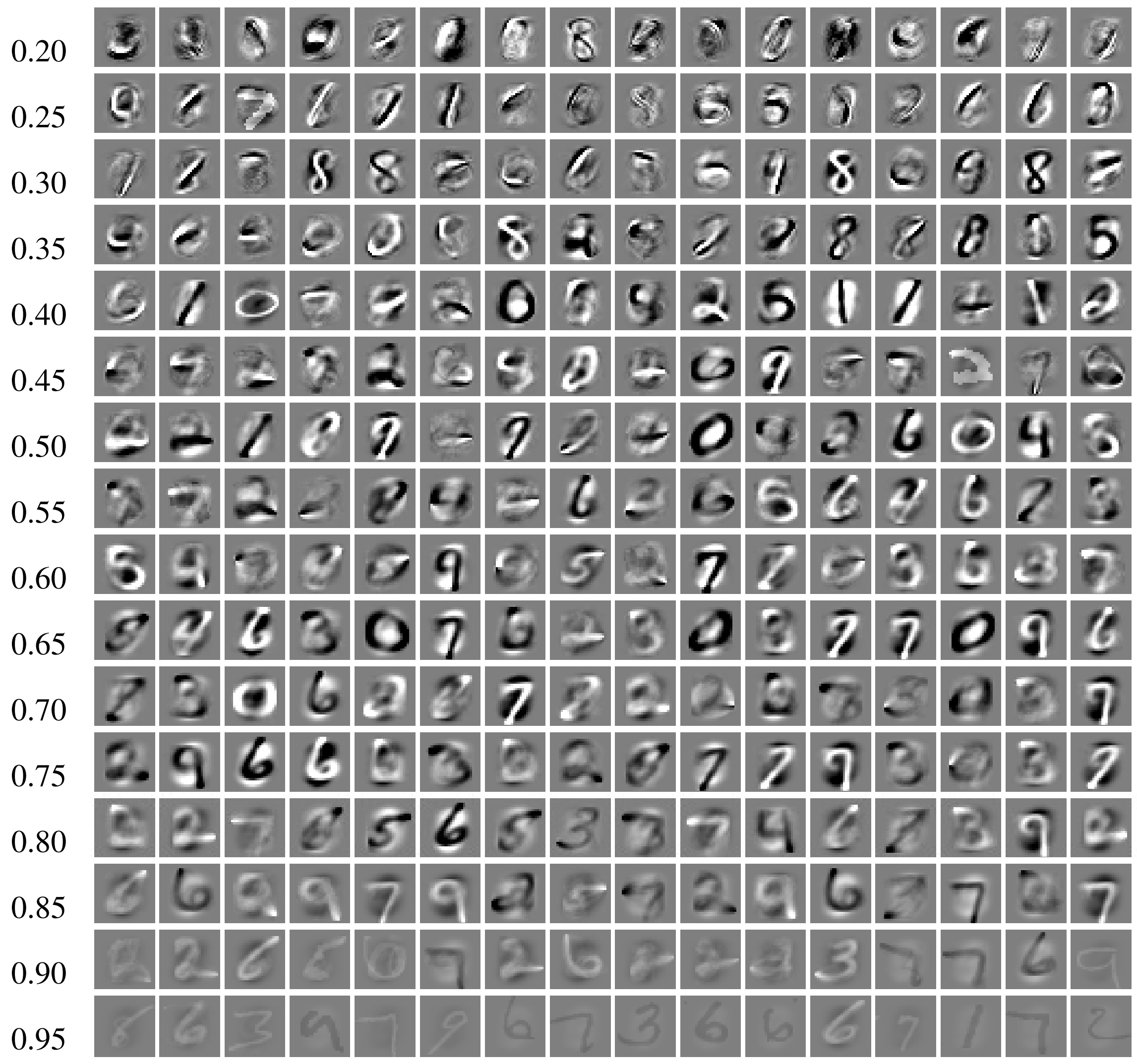


Figure 3.8: A subset of the bases from the mixing matrix resulting from Independent Component Analysis applied to MNIST. The bases were sorted according to the sparseness of the induced representation. For example, all bases from the first row can be used to infer signals with a sparseness in $[0.20,\ 0.25) \subseteq \mathbb{R}$ on the entire database. For low sparseness the bases feature localized strokes overlaid with random clouds, and the digit "eight" can be observed numerous times, whereas for high sparseness individual digits can be recognized.

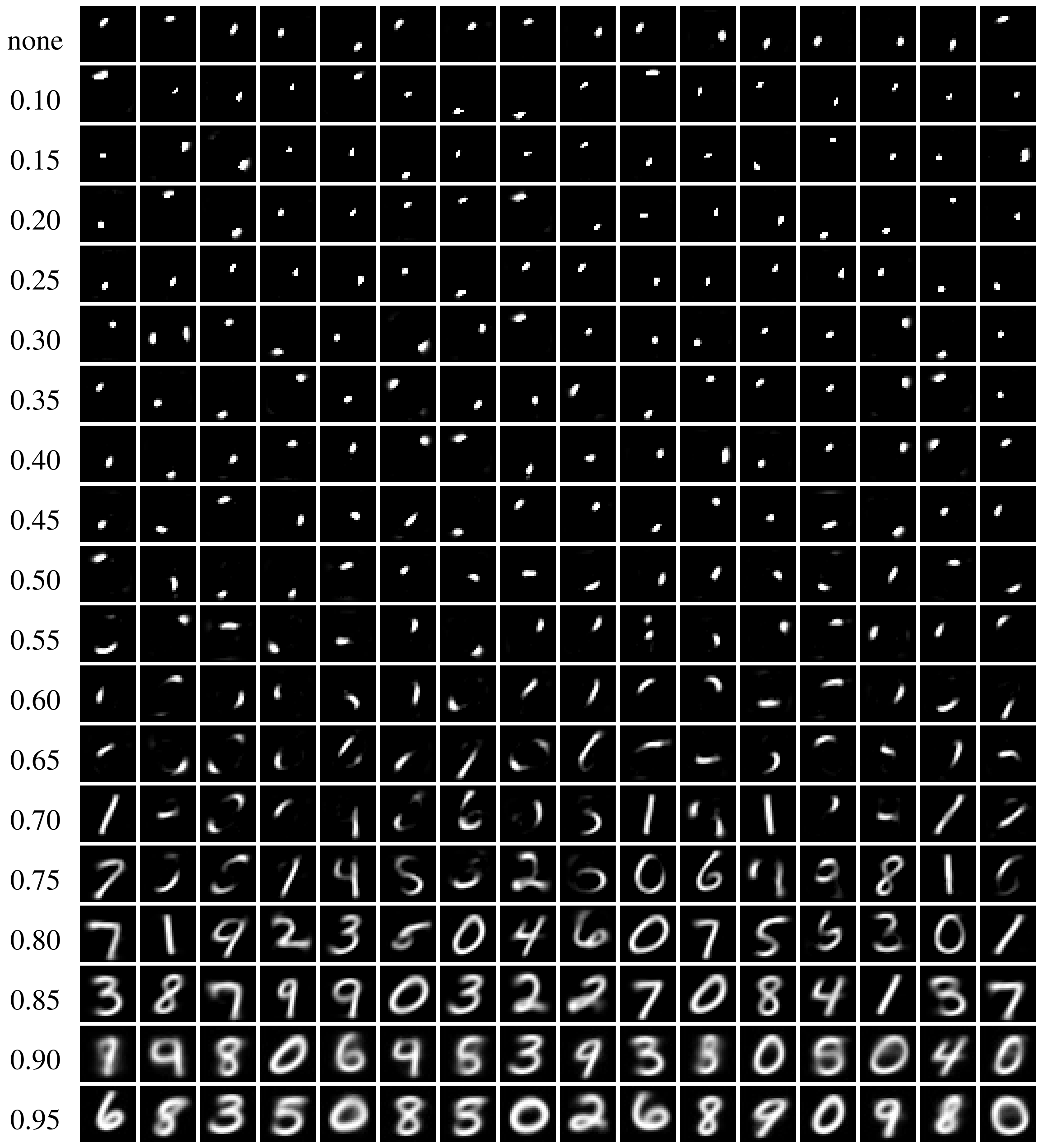


Figure 3.9: Resulting bases of Non-Negative Matrix Factorization (top row) and Non-Negative Matrix Factorization with Sparseness Constraints with σ_H set as depicted beneath the corresponding rows. When no or only a low sparseness of activity is enforced, then the bases resemble blobs without particular structure. When activity is sparser, individual strokes of the digits become visible, and for very sparse activity the bases look like samples from the data set.

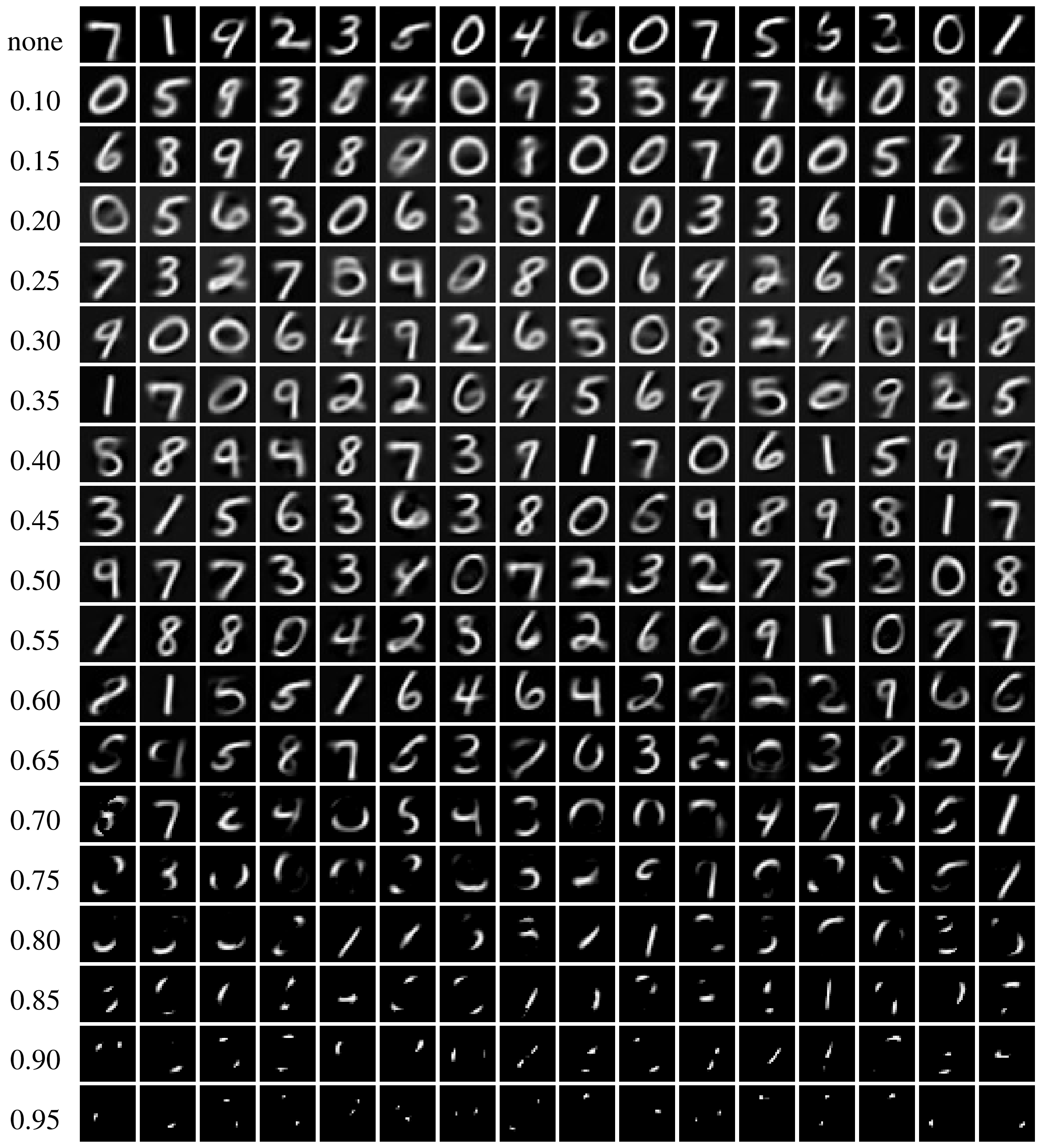


Figure 3.10: Resulting bases of Non-Negative Matrix Factorization with Sparseness Constraints with $\sigma_H := 0.80$ and σ_W set as given beneath the rows. Entire digits can be recognized for $\sigma_W \leq 0.60$, and begin to unravel for higher values of σ_W. For $\sigma_W = 0.75$ individual strokes are visible, while for $\sigma_W = 0.95$ only local blobs result.

4 On Hoyer's Sparseness-Enforcing Projection Operator

This chapter addresses Hoyer's normalized sparseness measure [26],

$$\sigma\colon \mathbb{R}^n \setminus \{0\} \to [0,\ 1], \qquad x \mapsto \frac{\sqrt{n} - \|x\|_1/\|x\|_2}{\sqrt{n} - 1},$$

where n denotes the dimensionality of the input vector, and in particular the problem of, given an arbitrary vector, finding the closest vector in the L_2 sense that achieves a certain degree of sparseness with respect to σ. Clearly, σ is constant on the following sets for the *target L_1 norm* $\lambda_1 > 0$ and the *target L_2 norm* $\lambda_2 > 0$:

$$S^{(\lambda_1,\lambda_2)} := \{\, s \in \mathbb{R}^n \mid \|s\|_1 = \lambda_1 \text{ and } \|s\|_2 = \lambda_2 \,\}, \text{ and } S_{\geq 0}^{(\lambda_1,\lambda_2)} := S^{(\lambda_1,\lambda_2)} \cap \mathbb{R}^n_{\geq 0}.$$

Here, the second set is for purpose of non-negative solutions. For existence of non-trivial solutions, $\lambda_2 \leq \lambda_1 \leq \sqrt{n}\lambda_2$ must hold. The *sparseness-enforcing projection operator*, denoted π in this work, is essentially a Euclidean projection onto parameterizations of these sets. For elimination of the scaling invariance between the L_1 and the L_2 norm, a parameter is introduced whether the L_2 norm of the input vector shall be retained, or if the output vector should be normalized. In formal terms, the sparseness projection is the function

$$\begin{gathered}\pi\colon \mathbb{R}^n \times [0,\ 1] \times \{\text{non-negative, arbitrary}\} \times \{\text{unit\_norm, keep\_norm}\} \to \mathbb{R}^n,\\ (x,\ \sigma^*,\ \omega,\ \eta) \mapsto \arg\min_{s \in S} \|x - s\|_2,\end{gathered}$$

where S is given by Table 4.1 based on the values of ω and η. Since $S^{(\lambda_1,\lambda_2)}$ and $S_{\geq 0}^{(\lambda_1,\lambda_2)}$ are scale-invariant, that is set membership is stable upon multiplication with a scalar, the problems of retaining the argument's norm or yielding a normalized projection, or finding the best approximation to the argument regardless of the scaling are closely related.

Hoyer has proposed an algorithm to compute π in [26], which is an alternating projection onto a hyperplane representing the L_1 norm constraint, a hypersphere representing the L_2 norm constraint and the non-negative orthant $\mathbb{R}^n_{\geq 0}$. In the special case when exactly one negative entry emerges that is zeroed out in the projection onto $\mathbb{R}^n_{\geq 0}$, Hoyer's algorithm was proven to be correct by [63]. There is however no theoretical justification whether alternating projections work in the general case, when more than one entry is set to zero at once. Because of the

Table 4.1: Decision table for the set S from the definition of π, accounting for the parameters ω and η. Here, x is the input vector and $\ell := \sqrt{n} - \sigma^* \cdot (\sqrt{n} - 1)$ is a constant for computing the appropriate L_1 norm such that a target sparseness of σ^* is achieved.

ω	η	S
non-negative	unit_norm	$S_{\geq 0}^{(\ell,1)}$
arbitrary	unit_norm	$S^{(\ell,1)}$
non-negative	keep_norm	$S_{\geq 0}^{(\ell \cdot \|x\|_2, \|x\|_2)}$
arbitrary	keep_norm	$S^{(\ell \cdot \|x\|_2, \|x\|_2)}$

non-convexity of $S^{(\lambda_1,\lambda_2)}$ and $S_{\geq 0}^{(\lambda_1,\lambda_2)}$, general alternating projections methods [64, 65] are not guaranteed to find a correct solution.

The remainder of this chapter studies the algorithmic computation of π. First, a formal definition of projections onto sets is given, and projections onto sets with certain symmetries are investigated. Then, an algorithm for π is deduced, where the main idea is that the solution with respect to π is not altered by projections onto simple sets, as hyperplanes, hyperspheres and simplexes. At the end of this chapter, a simple algorithm for π is given and its correctness is shown. Additionally, a more efficient implementation is given, using arguments from the analysis of projection properties. It is ultimately shown that the projection cast as function is differentiable almost everywhere and can hence be used as smooth neural transfer function suitable for gradient-based learning. The chapter is concluded with a comparison with alternative methods for computation of π and a discussion. This chapter contains material previously published in [66].

4.1 Projections onto Symmetric Sets

The following basic statement will be used thoroughly and follows from $\langle x,\ x\rangle = \|x\|_2^2$ for all $x \in \mathbb{R}^n$, keeping in mind that the scalar product is a symmetric bilinear form [27]:

Proposition 1. Let $a, b \in \mathbb{R}^n$. Then $\|a \pm b\|_2^2 = \|a\|_2^2 + \|b\|_2^2 \pm 2\langle a,\ b\rangle$. With another $p \in \mathbb{R}^n$,

$$\|a-b\|_2^2 = \|(a-p)+(p-b)\|_2^2 = \|a-p\|_2^2 + \|p-b\|_2^2 + 2\langle a-p,\ p-b\rangle .$$

On a real finite-dimensional vector space, all considered norms are equivalent in the topological sense [27]:

Proposition 2. The following inequalities hold for all $x \in \mathbb{R}^n$:

$$\|x\|_\infty \leq \|x\|_2 \leq \|x\|_1 \leq \sqrt{n}\,\|x\|_2 \leq n\,\|x\|_\infty .$$

The definition of a projection onto a set is a fundamental concept, for example see [67]:

Definition 3. Let $x \in \mathbb{R}^n$ and $\varnothing \neq M \subseteq \mathbb{R}^n$. Then every point in

$$\operatorname{proj}_M(x) := \{\, y \in M \mid \|y - x\|_2 \leq \|z - x\|_2 \text{ for all } z \in M \,\}$$

is called *Euclidean projection* of x onto M. If there is exactly one point y in $\operatorname{proj}_M(x)$, then $y = \operatorname{proj}_M(x)$ is written for abbreviation.

In this work only Euclidean projections onto subsets of the finite-dimensional Euclidean space $\mathbb{R}^n$ are considered. Therefore $\operatorname{proj}_M(x)$ is nonempty for all $x \in \mathbb{R}^n$ if and only if M is closed, and $\operatorname{proj}_M(x)$ consists of exactly one point for all $x \in \mathbb{R}^n$ if and only if M is closed and convex [67]. It is clear that $S^{(\lambda_1,\lambda_2)}$ and $S_{\geq 0}^{(\lambda_1,\lambda_2)}$ are closed but non-convex. Hence projections onto these sets always exist, but may not always be unique. The following remark collects these properties for later reference:

Remark 4. If $x \in M$, then $\operatorname{proj}_M(x) = x$ because of all norms being positive definite. When $p \in \operatorname{proj}_M(x)$ and $q \in M$ with $\|p - x\|_2 = \|q - x\|_2$, then $q \in \operatorname{proj}_M(x)$. If M is compact, then $\operatorname{proj}_M(x)$ is nonempty for all $x \in \mathbb{R}^n$ as a consequence of the Weierstraß extreme value theorem. If M is convex, then $\operatorname{proj}_M(x)$ consists of exactly one point because of $y \mapsto \|y - x\|_2$ being strictly convex.

As an example for these concepts, it is first shown that the result of the sparseness projection only depends on the ratio of the target L_1 norm λ_1 to the target L_2 norm λ_2, up to scaling:

Remark 5. Let $\lambda_1, \lambda_2 > 0$ and $\tilde{\lambda}_1, \tilde{\lambda}_2 > 0$ be pairs of target norms such that $\lambda_1/\lambda_2 = \tilde{\lambda}_1/\tilde{\lambda}_2$. Then $\operatorname{proj}_{S^{(\lambda_1,\lambda_2)}}(x) = \lambda_2/\tilde{\lambda}_2 \cdot \operatorname{proj}_{S^{(\tilde{\lambda}_1,\tilde{\lambda}_2)}}(x)$ for all $x \in \mathbb{R}^n$.

Proof. It is sufficient to show only one inclusion. Let $x \in \mathbb{R}^n$ be arbitrary, $p \in \operatorname{proj}_{S^{(\lambda_1,\lambda_2)}}(x)$ and $\tilde{r} \in S^{(\tilde{\lambda}_1,\tilde{\lambda}_2)}$. Define $\tilde{p} := \tilde{\lambda}_1/\lambda_1 \cdot p = \tilde{\lambda}_2/\lambda_2 \cdot p \in \mathbb{R}^n$, then $\|\tilde{p}\|_1 = \left|\tilde{\lambda}_1/\lambda_1\right| \cdot \|p\|_1 = \tilde{\lambda}_1$ and analogously $\|\tilde{p}\|_2 = \tilde{\lambda}_2$, hence $\tilde{p} \in S^{(\tilde{\lambda}_1,\tilde{\lambda}_2)}$. For the claim to hold it has now to be shown that $\|\tilde{p} - x\|_2 \leq \|\tilde{r} - x\|_2$. Write $r := \lambda_2/\tilde{\lambda}_2 \cdot \tilde{r} \in \mathbb{R}^n$, which in fact lies in $S^{(\lambda_1,\lambda_2)}$. Therefore $\|p - x\|_2 \leq \|r - x\|_2$ by definition of p, Proposition 1 implies

$$\begin{aligned}\|\tilde{r} - x\|_2^2 - \|\tilde{p} - x\|_2^2 &= \|\tilde{r}\|_2^2 + \|x\|_2^2 - 2\langle \tilde{r},\, x\rangle - \|\tilde{p}\|_2^2 - \|x\|_2^2 + 2\langle \tilde{p},\, x\rangle = 2\langle \tilde{p} - \tilde{r},\, x\rangle \\ &= \tilde{\lambda}_2/\lambda_2 \cdot 2\langle p - r,\, x\rangle = \tilde{\lambda}_2/\lambda_2 \cdot \left(\|r - x\|_2^2 - \|p - x\|_2^2\right) \geq 0,\end{aligned}$$

and the claim follows. □

The argument from the proof of Remark 5 can be directly generalized to any scale-invariant set, and hence also holds for $S_{\geq 0}^{(\lambda_1,\lambda_2)}$. Remark 5 can for example be applied to the problem of finding the projection of a point x onto the set $\{ s \in \mathbb{R}^n \setminus \{0\} \mid \sigma(s) = \sigma^* \}$ of all points featuring a target degree of sparseness σ^* regardless of the concrete scaling. Since all solutions of the sparseness projection where the scaling is ignored lie on a line due to Remark 5, it is enough to project x onto $S^{(\lambda_1,\lambda_2)}$ for an arbitrary pair (λ_1, λ_2) of target norms that lead to a sparseness of σ^*, and then rescale the resulting point p such as to minimize $\|x - \alpha p\|_2$ under variation of $\alpha \in \mathbb{R}$, which yields $\alpha = \langle x,\, p\rangle / \|p\|_2^2$. In this work, however, the magnitude of the result is restricted to the use cases given in Table 4.1 to guarantee stability.

Before Hoyer's sparseness-enforcing projection operator is addressed in greater detail, consider some general results on projections onto sets with certain symmetries. A great variety of sparseness measures possesses symmetries, for the sign of individual entries of their argument is irrelevant, as is the concrete ordering of the argument. This is reflected by the criteria A1 and A2 of [18]. Consider the following definition:

Definition 6. Let $\emptyset \neq M \subseteq \mathbb{R}^n$. M is called *permutation-invariant* if and only if $P_\tau x \in M$ for all $x \in M$ and all $\tau \in S_n$. M is called *reflection-invariant* if and only if $b \circ x \in M$ for all $x \in M$ and all $b \in \{\pm 1\}^n$.

Here, P_τ is the permutation matrix associated with the permutation τ and the operator $\circ$ denotes the Hadamard product, see Appendix A. Because every finite permutation can be expressed as composition of finitely many transpositions, an equivalent definition would be to only demand that τ be a single transposition. In other words, a subset M of the Euclidean space is called permutation-invariant if set membership is invariant to arbitrary permutation of individual coordinates.

M is called reflection-invariant if single entries can be negated without violating set membership of a point. This is equivalent to $x - 2\sum_{i \in I} x_i e_i \in M$ for all $x \in M$ and all index sets $I \subseteq \{1, \dots, n\}$, which is a condition that is technically easier to handle. With $|I|$ finite follows that this is equivalent to demand I to consist of exactly one element. The following observation states that these symmetries are closed under common set operators:

Remark 7. Let $A, B \subseteq \mathbb{R}^n$ be nonempty sets. When A and B are permutation-invariant or reflection-invariant, then so are $A \cup B$, $A \cap B$ and $A^C := \mathbb{R}^n \setminus A$.

Proof. The proof for $A \cup B$ and $A \cap B$ is trivial for both types of invariances.

Now let A be permutation-invariant, let $\tau \in S_n$ and $x \in \mathbb{R}^n \setminus A$. Would $P_\tau x \in A$ hold, then $P_{\tau^{-1}} P_\tau x = x \in A$ would hold, hence $P_\tau x \in \mathbb{R}^n \setminus A$.

Next let A be reflection-invariant, let $b \in \{\pm 1\}^n$ and let $x \in A^C$. If $b \circ x \in A$, then $b \circ b \circ x = x \in A$, so $b \circ x \in A^C$, hence A^C is reflection-invariant also. □

Therefore $S^{(\lambda_1,\lambda_2)}$ is both permutation-invariant and reflection-invariant. The same applies for example to the set $\{\, x \in \mathbb{R}^n \mid \|x\|_0 = \kappa \,\}$ of all vectors with exactly κ non-vanishing entries, and the set $\left\{\, x \in \mathbb{R}^n \mid \sum_{i=1}^n \log\left(1 + x_i^2\right) = \kappa \,\right\}$ of all vectors meeting the same constant sparseness degree κ with respect to the slog function.

Now consider the following general properties of functions mapping to power sets:

Definition 8. Let $\varnothing \neq M \subseteq \mathbb{R}^n$, let $\wp(M)$ be its power set and let $f\colon \mathbb{R}^n \to \wp(M)$ be a function. f is called *order-preserving* if and only if $x_i > x_j$ implies $p_i \geq p_j$ for all $x \in \mathbb{R}^n$, for all $p \in f(x)$ and for all $i, j \in \{1, \ldots, n\}$. f is called *absolutely order-preserving* if and only if from $|x_i| > |x_j|$ follows $|p_i| \geq |p_j|$ for all $x \in \mathbb{R}^n$, for all $p \in f(x)$ and for all $i, j \in \{1, \ldots, n\}$. f is called *orthant-preserving* if and only if $\operatorname{sgn}(x_i) = \operatorname{sgn}(p_i)$ or $x_i = 0$ or $p_i = 0$ for all $x \in M$, $p \in f(x)$ and $i \in \{1, \ldots, n\}$.

Hence, a function f is order-preserving if the relative order of entries of its arguments does not change upon function evaluation. Thus if the entries of x are sorted in ascending or descending order, then so are the entries of every vector in $f(x)$. Orthant-preservation denotes the fact that x and every vector from $f(x)$ are located within the same orthant. The link between set symmetries and projection properties is established by the following results:

Lemma 9. Let $x, p \in \mathbb{R}^n$ and define $d\colon S_n \to \mathbb{R}$, $\tau \mapsto \|p - x\|_2^2 - \|P_\tau p - x\|_2^2$.

(a) If $\tau = \tau^{-1}$, then $d(\tau) = \sum_{\substack{i=1 \\ i \neq \tau(i)}}^{n} \left(p_{\tau(i)} - p_i\right)\left(x_i - x_{\tau(i)}\right)$.

(b) If $\tau = (i_1, j_1) \circ \cdots \circ (i_d, j_d)$ is a composition of disjoint transpositions, that is $i_k, j_k \notin \{\, i_l \mid l \neq k \,\} \cup \{\, j_l \mid l \neq k \,\}$, then $d(\tau) = 2\sum_{k=1}^{d} \left(p_{j_k} - p_{i_k}\right)\left(x_{i_k} - x_{j_k}\right)$.

(c) If $\tau = (i, j)$ is a single transposition, then $d(\tau) = 2(p_j - p_i)(x_i - x_j)$.

Proof. (a) Let $\tau \in S_n$ be an involution. Then

$$\begin{aligned} d(\tau) &= \|p\|_2^2 + \|x\|_2^2 - 2\langle p,\, x\rangle - \|P_\tau p\|_2^2 - \|x\|_2^2 + 2\langle P_\tau p,\, x\rangle = 2\langle P_\tau p - p,\, x\rangle \\ &= \langle P_\tau p - p,\, x\rangle + \langle p - P_{\tau^{-1}} p,\, P_{\tau^{-1}} x\rangle \overset{\tau^2 = 1}{=} \langle P_\tau p - p,\, x\rangle + \langle p - P_\tau p,\, P_\tau x\rangle \\ &= \langle P_\tau p - p,\, x - P_\tau x\rangle = \sum_{\substack{i=1 \\ i \neq \tau(i)}}^{n} \left(p_{\tau(i)} - p_i\right)\left(x_i - x_{\tau(i)}\right). \end{aligned}$$

(b) Clearly, $\tau = \tau^{-1}$. Using (a) yields

$$d(\tau) = \sum_{k=1}^{d} \left[\left(p_{j_k} - p_{i_k}\right)\left(x_{i_k} - x_{j_k}\right) + \left(p_{i_k} - p_{j_k}\right)\left(x_{j_k} - x_{i_k}\right)\right] = 2\sum_{k=1}^{d} \left(p_{j_k} - p_{i_k}\right)\left(x_{i_k} - x_{j_k}\right).$$

(c) Follows directly from (b) with $d = 1$. □

Lemma 10. Let $\varnothing \neq M \subseteq \mathbb{R}^n$ and $p\colon \mathbb{R}^n \to \wp(M)$, $x \mapsto \operatorname{proj}_M(x)$. Then the following holds:

(a) When M is permutation-invariant, then p is order-preserving.

(b) When M is reflection-invariant, then p is orthant-preserving.

Proof. (a) Let $x \in \mathbb{R}^n$ and $p \in \operatorname{proj}_M(x)$. Let $i, j \in \{1, \dots, n\}$ with $x_i > x_j$. Assume $p_i < p_j$. Let $\tau := (i, j)$ and $q := P_\tau p$, then $q \in M$ because of M being permutation-invariant. With Lemma 9(c), $d(\tau) > 0$ and thus $\lVert p - x\rVert_2 > \lVert q - x\rVert_2$, which contradicts the minimality of p. Therefore $p_i \geq p_j$ must hold.

(b) Let $x \in \mathbb{R}^n$, $p \in \operatorname{proj}_M(x)$ and $I := \{\, i \in \{1, \dots, n\} \mid \operatorname{sgn}(x_i) \neq \operatorname{sgn}(p_i) \,\}$. The claim holds trivially if $I = \varnothing$. Assume $I \neq \varnothing$ and define $q := p - 2\sum_{i \in I} p_i e_i$. Because of M reflection-invariant follows $q \in M$. With

$$\lVert q\rVert_2^2 = \lVert p\rVert_2^2 + \left\lVert 2\sum\nolimits_{i \in I} p_i e_i\right\rVert_2^2 - 4\left\langle p,\ \sum\nolimits_{i \in I} p_i e_i\right\rangle = \lVert p\rVert_2^2 + 4\sum\nolimits_{i \in I} p_i^2 - 4\sum\nolimits_{i \in I} p_i^2 = \lVert p\rVert_2^2$$

and $\langle q,\ x\rangle = \langle p,\ x\rangle - 2\sum_{i \in I} p_i x_i$ follows

$$d := \lVert p - x\rVert_2^2 - \lVert q - x\rVert_2^2 = \lVert p\rVert_2^2 + \lVert x\rVert_2^2 - 2\,\langle p,\ x\rangle - \lVert q\rVert_2^2 - \lVert x\rVert_2^2 + 2\,\langle q,\ x\rangle = -4\sum\nolimits_{i \in I} p_i x_i.$$

By the definition of I follows that $p_i x_i \in \{\, -1, 0 \,\}$. Hence when there is an $i \in I$ with $p_i \neq 0$ and $x_i \neq 0$, then $d > 0$, but $\lVert p - x\rVert_2^2 > \lVert q - x\rVert_2^2$ contradicts the minimality of p. Therefore $I = \{\, i \in \{1, \dots, n\} \mid p_i = 0 \text{ or } x_i = 0 \,\}$, and the claim follows. □

Lemma 11. Let $\varnothing \neq M \subseteq \mathbb{R}^n$ be permutation-invariant and $x \in \mathbb{R}^n$. If $p = \operatorname{proj}_M(x)$ is unique, then $p_i = p_j$ holds for all $i, j \in \{1, \dots, n\}$ with $x_i = x_j$.

Proof. Let $x \in \mathbb{R}^n$, $p = \operatorname{proj}_M(x)$ and $i, j \in \{1, \dots, n\}$ with $x_i = x_j$. Assume $p_i \neq p_j$ and let $\tau := (i, j)$ and $q := P_\tau p \neq p$. With the permutation-invariance of M follows $q \in M$, and $\lVert q - x\rVert_2 = \lVert p - x\rVert_2$ with $x_i = x_j$. Hence $q \in \operatorname{proj}_M(x)$, so $q = p$ with the uniqueness of the projection, which contradicts $q \neq p$. Therefore, $p_i = p_j$. □

Hence, unique projections of sorted vectors onto permutation-invariant sets yield sorted vectors with Lemma 10 and Lemma 11. Note that this does not hold when the projection is not unique. For example consider the projection of 0 onto the unit sphere $\{\, x \in \mathbb{R}^n \mid \lVert x\rVert_2 = 1 \,\}$ yielding the whole hypersphere, which also contains vectors that are not sorted. However, from every such projection a sorted vector can be constructed solely using permutations.

The next result was observed by [26] without proof in the special case of the sparseness-enforcing projection operator, and by [68] in the context of an L_1 ball, where a hint to a possible proof was given. The result shows how solutions to a projection onto reflection-invariant sets can be turned into non-negative solutions and vice-versa. Therefore, it suffices to consider non-negative solutions when studying such projections, for example projections onto $S^{(\lambda_1,\lambda_2)}$ can be computed easily from projections onto $S_{\geq 0}^{(\lambda_1,\lambda_2)}$.

Lemma 12. Let $\varnothing \neq A \subseteq \mathbb{R}^n$ be reflection-invariant, $B := A \cap \mathbb{R}^n_{\geq 0}$ and $p, x \in \mathbb{R}^n$. Then:

(a) If $p \in \mathrm{proj}_A(x)$, then $|p| \in \mathrm{proj}_B(|x|)$.

(b) If $p \in \mathrm{proj}_B(|x|)$, then $s \circ p \in \mathrm{proj}_A(x)$ where $s \in \mathbb{R}^n$ is given by $s_i := 1$ if $x_i \geq 0$ and $s_i := -1$ otherwise for all $i \in \{1, \dots, n\}$.

Proof. First note that if $q \in \mathrm{proj}_A(x)$, then $\mathrm{sgn}(x_i) = \mathrm{sgn}(q_i)$ or $x_i = 0$ or $q_i = 0$ with Lemma 10. Hence for all $i \in \{1, \dots, n\}$ follows $(|q_i| - |x_i|)^2 = (q_i \cdot \mathrm{sgn}(q_i) - x_i \cdot \mathrm{sgn}(x_i))^2 = (q_i - x_i)^2$, and therefore $\|q - x\|_2^2 = \||q| - |x|\|_2^2$. Furthermore, $|q| \in A$ because of A reflection-invariant and $|q| \in \mathbb{R}^n_{\geq 0}$, so $|q| \in B$.

(a) Let $p \in \mathrm{proj}_A(x)$ and $q \in B$, then $|p| \in B$ and it has to be shown that $\||p| - |x|\|_2 \leq \|q - |x|\|_2$. Define $I := \{\, i \in \{1, \dots, n\} \mid x_i < 0 \,\}$ and $\tilde{q} := q - 2\sum_{i \in I} q_i e_i$. Clearly $\tilde{q} \in A$, so in conjunction with the remark at the beginning of the proof follows $\||p| - |x|\|_2^2 = \|p - x\|_2^2 \leq \|\tilde{q} - x\|_2^2$. For $i \notin I$ is $x_i \geq 0$ and $\tilde{q}_i = q_i$, hence $\tilde{q}_i - x_i = q_i - |x_i|$. For $i \in I$ follows $x_i < 0$ and $\tilde{q}_i = -q_i$, hence $\tilde{q}_i - x_i = -(q_i - |x_i|)$. This yields $(\tilde{q}_i - x_i)^2 = (q_i - |x_i|)^2$ for all $i \in \{1, \dots, n\}$, thus $\|\tilde{q} - x\|_2^2 = \|q - |x|\|_2^2$, and the claim follows.

(b) Let $p \in \mathrm{proj}_B(|x|)$. If $i \in \{1, \dots, n\}$ with $x_i \geq 0$, then clearly $s_i p_i - x_i = p_i - |x_i|$. For $i \in \{1, \dots, n\}$ with $x_i < 0$ follows $s_i p_i - x_i = -(p_i - |x_i|)$. Therefore, $\|s \circ p - x\|_2^2 = \|p - |x|\|_2^2$. Let $q \in \mathrm{proj}_A(x)$, then the remark at the beginning of the proof yields $\|q - x\|_2^2 = \||q| - |x|\|_2^2$ and $|q| \in B$. $p \in \mathrm{proj}_B(|x|)$ yields $\|p - |x|\|_2^2 \leq \||q| - |x|\|_2^2$, and the claim follows. $\square$

This result immediately yields a condition for projections to be absolutely order-preserving:

Lemma 13. Let $\varnothing \neq M \subseteq \mathbb{R}^n$ be both permutation-invariant and reflection-invariant. Then $p\colon \mathbb{R}^n \to \wp(M)$, $x \mapsto \mathrm{proj}_M(x)$, is absolutely order-preserving.

Proof. Let $x \in \mathbb{R}^n$, $p \in \mathrm{proj}_M(x)$, and $i, j \in \{1, \dots, n\}$ with $|x_i| > |x_j|$. Define $L := M \cap \mathbb{R}^n_{\geq 0}$, which is permutation-invariant with Remark 7. Lemma 12 implies that $|p| \in \mathrm{proj}_L(|x|)$, and with Lemma 10 follows $|p_i| \geq |p_j|$. $\square$

The considered symmetry concepts can be expanded to functions as well:

Definition 14. Let $\varnothing \neq M \subseteq \mathbb{R}^n$ and $f\colon M \to \mathbb{R}$. f is called *permutation-invariant* if and only if $f(P_\tau x) = f(x)$ for all $x \in M$ and all $\tau \in S_n$. f is called *reflection-invariant* if and only if $f(b \circ x) = f(x)$ for all $x \in M$ and all $b \in \{\,\pm 1\,\}^n$.

The following theorem summarizes the results from this section applied to preimages of constants under symmetric functions:

Theorem 15. Let $\varnothing \neq M \subseteq \mathbb{R}^n$ be a set and $f\colon M \to \mathbb{R}$ be a function. Let $\kappa \in f(M)$ be constant and consider $g\colon \mathbb{R}^n \to M$, $x \mapsto \operatorname{proj}_{M \cap f^{-1}(\{\kappa\})}(x)$.

(a) When M and f are permutation-invariant, then g is order-preserving.

(b) When M and f are reflection-invariant, then g is orthant-preserving.

(c) When M and f are permutation-invariant and reflection-invariant, then g is absolutely order-preserving.

Proof. (a) With Lemma 10 and Remark 7 it is sufficient to show that $f^{-1}(\{\kappa\})$ is permutation-invariant. Let $\tau \in S_n$ and $x \in f^{-1}(\{\kappa\})$. Then $f(P_\tau x) = f(x) = \kappa$, hence $P_\tau x \in f^{-1}(\{\kappa\})$.

(b) Analogous to (a).

(c) From (a) and (b) follows that $f^{-1}(\{\kappa\})$ is both permutation-invariant and reflection-invariant, and so is $M \cap f^{-1}(\{\kappa\})$ with Remark 7, thus the claim follows with Lemma 13. $\square$

4.2 Projection onto an L_0 Pseudo-Norm Constraint

As an example for the previously discussed concepts, consider the L_0 pseudo-norm introduced in Section 2.2.1:

$$\|\cdot\|_0 \colon \mathbb{R}^n \to \{0,\dots,n\}, \qquad x \mapsto \|x\|_0 := |\{\, i \in \{1,\dots,n\} \mid x_i \neq 0 \,\}| .$$

Let $\kappa \in \mathbb{N}$ be a constant and consider the set $Z := \{\, x \in \mathbb{R}^n \mid \|x\|_0 = \kappa \,\}$ of all vectors with exactly κ non-vanishing entries. Because $\|\cdot\|_0$ and Z are both permutation-invariant and reflection-invariant, the statements from Theorem 15 hold.

In fact, the projection onto Z consists simply of zeroing out all entries but the κ that are greatest in absolute value [69]. For realizing this, consider an arbitrary $p \in Z$ and let $I \subseteq \{1,\dots,n\}$ with $|I| = \kappa$ be the index subset of nonzero entries. Consider the expression $\|p - x\|_2^2 = \sum_{i \in I}(p_i - x_i)^2 + \sum_{i \notin I} x_i^2$. The latter term can be minimized by choosing I as the index set with the greatest absolute entries, and the first term is minimized by choosing $p_i = x_i$ for all $i \in I$. Similarly, the projection onto $Z \cap \mathbb{R}^n_{\geq 0}$ can be carried out analogously, but choosing I as the index set with the greatest entries instead of the greatest *absolute* entries. With application of Lemma 12, the general case where non-negativity is not required can be recovered again.

These arguments can directly be transferred to Hoyer's sparseness measure σ and the sets $S^{(\lambda_1,\lambda_2)}$ and $S_{\geq 0}^{(\lambda_1,\lambda_2)}$, on which σ attains a constant value. In the remainder of this chapter it is enough to consider projections onto $S_{\geq 0}^{(\lambda_1,\lambda_2)}$, as solutions for projections onto $S^{(\lambda_1,\lambda_2)}$ can be computed using Lemma 12. As will be shown later on, the property of order preservation can be exploited to both simplify and increase the efficiency of the proposed algorithm for π.

4.3 Sparseness-Enforcing Projection Operator

Now that the basic concepts have been introduced, consider the special case of the sparseness-enforcing projection operator π. As explained in the previous section, it is enough to handle projections onto the target set $S_{\geq 0}^{(\lambda_1,\lambda_2)}$. The basic idea for finding an algorithm for π is here carrying out intermediate projections onto sets that do not tamper the solution set, similar to the methods of [26, 63, 70]. If the target set was convex as an intersection of convex sets, alternating projections would be guaranteed to find a solution [64, 65]. Because of its non-convexity, the general approach must be shown to work still in this special situation. Extending these ideas and providing non-trivial generalizations, the remainder of this section presents a rigorous proof of correctness of an improved algorithm for π. Some arguments have been taken over from [63], affecting Lemma 16, Lemma 19, and the special case of Corollary 35 when there is exactly one negative entry.

First consider some definitions that are fixed throughout this chapter. Let $n \in \mathbb{N}$, $n \geq 2$, be the problem dimensionality. Let $e := J_{n\times 1} \in \mathbb{R}^n$ be the vector where each entry is equal to one. Let $e_1, \dots, e_n \in \mathbb{R}^n$ be the canonical basis vectors of $\mathbb{R}^n$. For all other vectors but e, subscripts denote their respective entries, for example $e_i^T x = x_i$ for $x \in \mathbb{R}^n$ and $i \in \{1, \dots, n\}$.

Let $\lambda_1, \lambda_2 > 0$ be the target norms with $\lambda_2 \leq \lambda_1 \leq \sqrt{n}\lambda_2$ to avoid the existence of trivial solutions only. Let $H := \{\, a \in \mathbb{R}^n \mid e^T a = \lambda_1 \,\}$, which is the target hyperplane with respect to the L_1 norm constraint in the non-negative orthant. Further, let $K := \{\, q \in \mathbb{R}^n \mid \|q\|_2 = \lambda_2 \,\}$ denote the target hypersphere of all points satisfying the L_2 norm constraint. This yields:

$$S_{\geq 0}^{(\lambda_1,\lambda_2)} = \mathbb{R}^n_{\geq 0} \cap H \cap K =: D.$$

Let $L := H \cap K$ denote the intersection of the L_1 norm target hyperplane and the L_2 norm target hypersphere. L essentially possesses the structure of a hypercircle, that is, all points in L lie in H and have constant squared distance $\rho := \lambda_2^2 - \lambda_1^2/n$ from the barycenter $m := \lambda_1/n \cdot e \in \mathbb{R}^n$, as will be shown later. The intersection of the non-negative orthant with the L_1 norm hyperplane, $C := \mathbb{R}^n_{\geq 0} \cap H$, is a scaled canonical simplex with barycenter m.

Using these definitions the intermediate goal is now to deduce and prove correctness of an algorithm for carrying out projections onto D. As will be shown, this projection can be computed by alternating projections onto the geometric structures just defined. The key is to show that the set of solutions is not tampered by these intermediate projections. A diagram summarizing these intermediate sets is depicted in Figure 4.1.

4.3.1 L_1 Norm Constraint – Target Hyperplane

First, the projection onto the target hyperplane H is considered, see Figure 4.2a for an illustration. Lemma 16 is an elaborated version of a result from [63]. Using its statements, it can be assumed that $x \in H$ without tampering the set of Euclidean projections of x onto D.

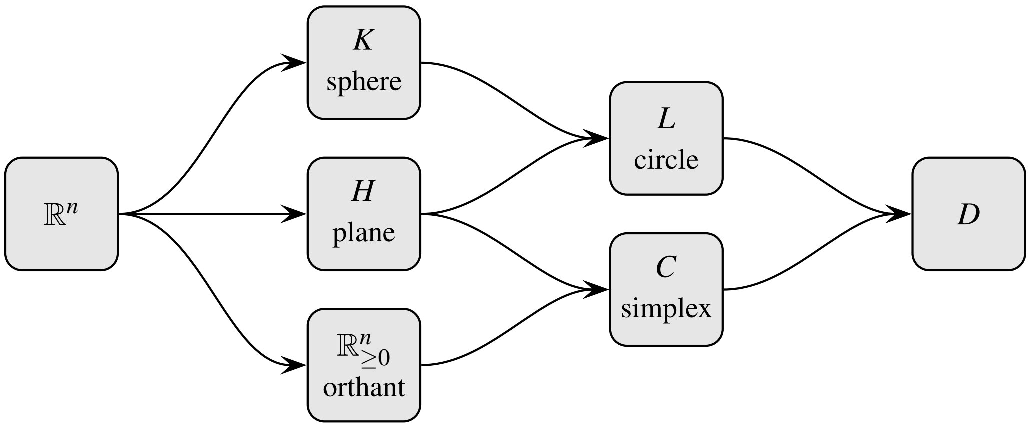


Figure 4.1: Overview of involved sets, where $A \to B$ means $A \supseteq B$. H and K are the target hyperplane and hypersphere, respectively. Further, $L := H \cap K$ is a hypercircle lying in H, $C := \mathbb{R}^n_{\geq 0} \cap H$ is a simplex, and $D = L \cap C$ is the set of feasible solutions.

Lemma 16. Let $x \in \mathbb{R}^n$. Then:

(a) $\operatorname{proj}_H(x) = x + {}^1\!/_n \cdot \left(\lambda_1 - e^T x\right) e$.

(b) Let $r := \operatorname{proj}_H(x)$. Then $\operatorname{proj}_D(x) = \operatorname{proj}_D(r)$.

Proof. (a) Because of H being convex, a projection must exist and is unique according to Remark 4. Now consider the Lagrangian $L\colon \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$, $(y, \mu) \mapsto \|x - y\|_2^2 + \mu\left(e^T y - \lambda_1\right)$. It is $\left({}^{\partial L}\!/_{\partial y}\right)^T = 2y - 2x + \mu e \overset{!}{=} 0 \in \mathbb{R}^n$ and ${}^{\partial L}\!/_{\partial \mu} = e^T y - \lambda_1 \overset{!}{=} 0 \in \mathbb{R}$, hence $y = x - {}^{\mu}\!/_2 \cdot e$ and thus $\lambda_1 \overset{!}{=} e^T y = e^T x - {}^{\mu}\!/_2 \cdot e^T e = e^T x - {}^{\mu n}\!/_2$, so $\mu = {}^2\!/_n \cdot \left(e^T x - \lambda_1\right)$. Substitution into the Lagrangian's derivative for y yields the claim.

(b) With (a), it is $r - x = {}^1\!/_n \cdot \left(\lambda_1 - e^T x\right) e$. Hence $\langle h,\ r - x\rangle = {}^{\lambda_1}\!/_n \cdot \left(\lambda_1 - e^T x\right)$ for all $h \in H$, which is independent of the concrete entries of h. Thus $\langle a - b,\ r - x\rangle = 0$ for every $a, b \in H$.

Now let $p \in \operatorname{proj}_D(x)$, that is $\|p - x\|_2 \leq \|q - x\|_2$ for all $q \in D$. Let $q \in D$ be arbitrary. With $D \subseteq H$ follows $\langle q - p,\ r - x\rangle = 0$, and thus Proposition 1 yields $\|q - r\|_2^2 - \|p - r\|_2^2 = \|q - x\|_2^2 - \|p - x\|_2^2 + 2\langle q - p,\ x - r\rangle = \|q - x\|_2^2 - \|p - x\|_2^2 \geq 0$, hence $\|p - r\|_2^2 \leq \|q - r\|_2^2$, so $p \in \operatorname{proj}_D(r)$.

For the converse, let $p \in \operatorname{proj}_D(r)$. Analogous to above, it follows $\|q - x\|_2^2 - \|p - x\|_2^2 = \|q - r\|_2^2 - \|p - r\|_2^2 \geq 0$, hence $p \in \operatorname{proj}_D(x)$. □

Remark 17. With Lemma 16 it is $\operatorname{proj}_H(0) = m$, and indeed $e^T m = {}^{\lambda_1}\!/_n \cdot e^T e = \lambda_1$, thus $m \in H$. Moreover, $\|m\|_2^2 = {}^{\lambda_1^2}\!/_{n^2} \cdot e^T e = {}^{\lambda_1^2}\!/_n$. For all $h \in H$, it is $\langle m,\ h\rangle = {}^{\lambda_1}\!/_n \langle e,\ h\rangle = {}^{\lambda_1^2}\!/_n$, and thus $\|h - m\|_2^2 = \|h\|_2^2 - {}^{\lambda_1^2}\!/_n$.

4.3.2 L_1 and L_2 Norm Constraints – Target Hypersphere

After the projection onto the target hyperplane has been carried out, consider now the joint constraint of the target hyperplane with the target hypersphere. First note that the intersection of the hyperplane with the hypersphere, $L := H \cap K$, yields a hypersphere in the plane, where the intrinsic dimension is reduced by one.

Lemma 18 provides insight into the geometric properties of this hypercircle: The mid-point of L is m, and L collapses to exactly one point if and only if $\lambda_2 = \lambda_1/\sqrt{n}$. The latter is equivalent to Hoyer's sparseness measure σ attaining zero. The position of L in H is depicted in Figure 4.2a and Figure 4.2b.

Lemma 18. Let $L := H \cap K$ and $\rho := \lambda_2^2 - \lambda_1^2/n$.

(a) $L = \tilde{L} := \{\, q \in H \mid \|q - m\|_2^2 = \rho \,\}$.

(b) $L \neq \varnothing$ if and only if $\lambda_2 \geq \lambda_1/\sqrt{n}$.

(c) For all $a, b \in L$ it is $\|a - b\|_2^2 = 2\left(\lambda_2^2 - \langle a,\, b\rangle\right)$, hence $\langle a,\, b\rangle = \lambda_2^2 - 1/2\,\|a - b\|_2^2$.

(d) For all $a, b \in L$ it is $\|a - b\|_2^2 \leq 4\rho$ with equality if and only if $m = \frac{a+b}{2}$.

Proof. (a) For $L \subseteq \tilde{L}$, let $q \in L$. Clearly $q \in H$. With Remark 17, $\|q - m\|_2^2 = \|q\|_2^2 - \lambda_1^2/n \overset{q \in L}{=} \lambda_2^2 - \lambda_1^2/n = \rho$. For $\tilde{L} \subseteq L$, let $q \in \tilde{L}$. Then $q \in H$, and with Remark 17 it is $\|q\|_2^2 = \|q - m\|_2^2 + \lambda_1^2/n = \rho + \lambda_1^2/n = \lambda_2^2$, hence $q \in K$, hence $q \in L$.

(b) L is nonempty if and only if $\rho \geq 0$ with (a). This number can be factorized into $\rho = (\lambda_2 + \lambda_1/\sqrt{n})\,(\lambda_2 - \lambda_1/\sqrt{n})$. Hence, with $\lambda_1, \lambda_2 > 0$ it is $\rho \geq 0$ if and only if $\lambda_2 - \lambda_1/\sqrt{n} \geq 0$.

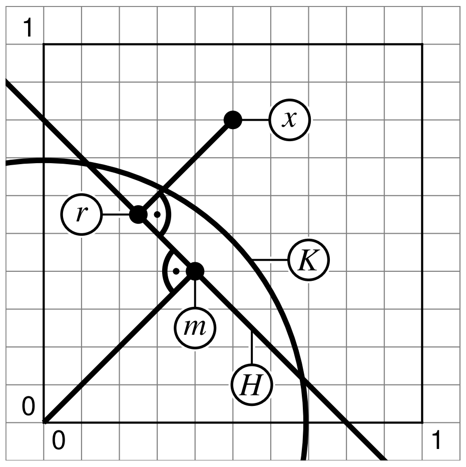
(c) The claim follows directly with Proposition 1 and $\|a\|_2 = \|b\|_2 = \lambda_2$.

(d) With the triangle inequality, $\|a - b\|_2 \leq \|a - m\|_2 + \|m - b\|_2 \overset{\text{(a)}}{=} 2\sqrt{\rho}$. With the parallelogram law [67] and $a, b \in K$ follows $\|a + b\|_2^2 + \|a - b\|_2^2 = 2\,\|a\|_2^2 + 2\,\|b\|_2^2 = 4\lambda_2^2$.

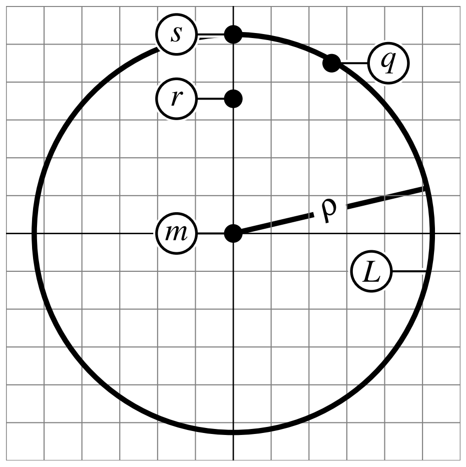
Thus, if $m = \frac{a+b}{2}$, then $\|a - b\|_2^2 = 4\lambda_2^2 - \|2m\|_2^2 = 4\rho$ with Remark 17.

Now let $\|a - b\|_2^2 = 4\rho$. Then $\|a + b\|_2^2 = 4\lambda_2^2 - 4\rho$. Define $q := a + 2\,(m - a)$. With Remark 17, it is $\|q - b\|_2^2 = \|2m - (a + b)\|_2^2 = \|2m\|_2^2 - 4\left(\langle m,\, a\rangle + \langle m,\, b\rangle\right) + \|a + b\|_2^2 = 0$. Thus $q = b$ because of $\|\cdot\|_2$ being positive definite, hence $a + b = 2m$. □

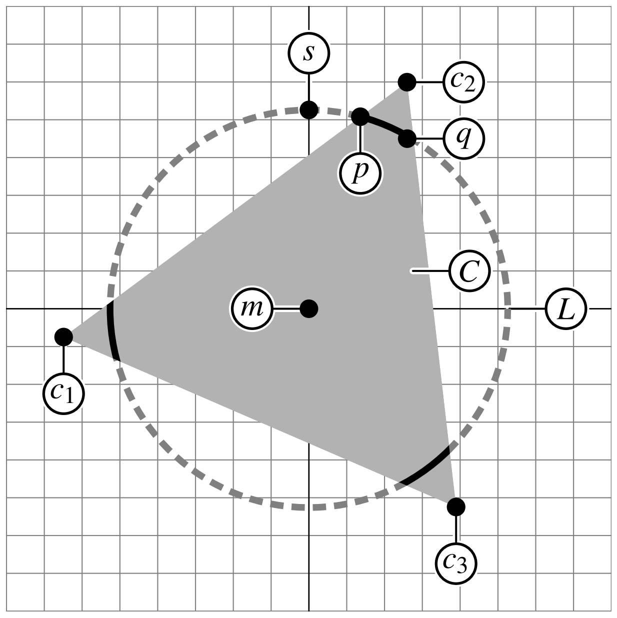
Now that a characterization of the geometry of L is obtained, projections onto it are considered. Lemma 19 states where the projection of an arbitrary point from H lies in L, and that the solution set with respect to π is not altered by this projection. For a sketch of the points involved, see Figure 4.2b. The arguments have been taken over from [63], where the minor flaw of the ignorance of the statements of Lemma 18 has been corrected and the solution to the quadratic equation has been solved explicitly, simplifying the original proof of [63].



(a) Situation of Lemma 16, projected onto the depth plane.



(b) Situation of Lemma 18 and Lemma 19, projected onto H.



(c) Illustration of Corollary 27, projected onto H. Non-negative regions on L are marked black, negative regions are dashed in gray. The corner points of the simplex are $c_i = \lambda_1 e_i$ for $i \in \{1,2,3\}$.

Figure 4.2: Illustration of the first steps of Algorithm 4.2 for $\lambda_1 = 0.8$ and $\lambda_2 = 0.7$, and a point $x = (0.5, 0.8, 0.3)^T$. Adapted from [66].

Lemma 19. Let $r \in H$ with $r \neq m$. Let $s := m + \delta(r-m)$ where $\delta := \sqrt{\rho}/\|r-m\|_2$. Then:

(a) $\delta > 0$, $s \in L$ and for all $q \in L$ holds $\|q-r\|_2^2 - \|s-r\|_2^2 = {}^1\!/\!_\delta \|q-s\|_2^2$.

(b) $s = \operatorname{proj}_L(r)$.

(c) $\operatorname{proj}_D(r) = \operatorname{proj}_D(s)$.

Proof. (a) First note that $\delta > 0$ because of $r \neq m$. Clearly $s = (1-\delta)m + \delta r$. It is $s \in H$ because of $e^T s = \lambda_1$, and $s \in L$ because of $\|s-m\|_2^2 = \rho$ and Lemma 18.

Let $q \in L$ be arbitrary. Because of $q, s \in K$ it is $\|q-r\|_2^2 - \|s-r\|_2^2 = \|q\|_2^2 + \|r\|_2^2 - 2\langle q,\, r\rangle - \|s\|_2^2 - \|r\|_2^2 + 2\langle s,\, r\rangle = 2\langle s-q,\, r\rangle$. With Remark 17, it is $\langle m,\, r\rangle = \lambda_1^2/n$ and $\|r\|_2^2 = \|r-m\|_2^2 + \lambda_1^2/n$. Hence, $\langle s,\, r\rangle = (1-\delta)\langle m,\, r\rangle + \delta\langle r,\, r\rangle = \lambda_1^2/n + \delta\|r-m\|_2^2$. On the other hand, from $s - m = \delta(r-m)$ and hence $r = (1 - 1/\delta)m + 1/\delta \cdot s$, and using Lemma 18(c) it follows that $\langle q,\, r\rangle = (1-1/\delta)\lambda_1^2/n + 1/\delta \cdot \left(\lambda_2^2 - 1/2 \cdot \|q-s\|_2^2\right)$. Therefore with $\delta\|r-m\|_2^2 = \rho/\delta$ it is

$$\langle s-q,\, r\rangle = \delta\|r-m\|_2^2 + 1/\delta \cdot \left(\lambda_1^2/n - \lambda_2^2 + 1/2 \cdot \|q-s\|_2^2\right) = \tfrac{1}{2\delta}\|q-s\|_2^2,$$

and the claim follows directly by substitution.

(b) Let $q \in L$, then with (a) it follows $\|q-r\|_2^2 - \|s-r\|_2^2 = 1/\delta\|q-s\|_2^2 \geq 0$ with equality if and only if $q = s$ because of the positive definiteness of $\|\cdot\|_2$. Thus s is the unique projection of r onto L.

(c) With (a) follows $\|q-s\|_2^2 - \|p-s\|_2^2 = \delta\big(\|q-r\|_2^2 - \|p-r\|_2^2\big)$ for all $p, q \in D$ because of $D \subseteq L$. For $\operatorname{proj}_D(r) \subseteq \operatorname{proj}_D(s)$, let $p \in \operatorname{proj}_D(r)$ and $q \in D$. By definition $\|p-r\|_2^2 \leq \|q-r\|_2^2$, and thus $\|q-s\|_2^2 - \|p-s\|_2^2 \geq 0$ with $\delta > 0$, hence $p \in \operatorname{proj}_D(s)$. For the converse, let $p \in \operatorname{proj}_D(s)$ and $q \in D$. Similarly, $\|q-r\|_2^2 - \|p-r\|_2^2 = 1/\delta \cdot \left(\|q-s\|_2^2 - \|p-s\|_2^2\right) \geq 0$, thus $p \in \operatorname{proj}_D(r)$. □

Remark 20. Analogous to Lemma 16 it can thus be assumed that $x \in L$ without changing the set of Euclidean projections of x onto D. The only exception $r = m$ forms a null set. However, in practice this case can occur when the input vector x is poorly chosen, for example if all elements are equal. In this case, $\operatorname{proj}_L(r) = L$, hence any point from L can be chosen for further processing.

One possibility would be to choose $s := \alpha\sum_{i=1}^{n-1} e_i + \beta e_n$ for $\alpha, \beta \in \mathbb{R}$, that is forcing the last entry to be unequal to the other ones. For satisfying $e^T s = \lambda_1$ and $\|s\|_2 = \lambda_2$, that is $s \in L$, set $\alpha := \lambda_1/n + \sqrt{\rho}/\sqrt{n(n-1)}$ and $\beta := \lambda_1 - \alpha(n-1) = \lambda_1/n - \sqrt{\rho(n-1)}/\sqrt{n}$. This is an example for a sorted solution where projections are not unique, see Lemma 11.

A geometric interpretation of how s is computed in Lemma 19 is as follows: A ray is cast, starting in m and going through r. The intersection of this ray with L then yields s. The value of δ is chosen such that a point with squared distance ρ from m is achieved.

Combining these properties of H and L, it can now be shown that projections onto D are invariant to affine-linear transformations with positive scaling:

Corollary 21. Let $\alpha > 0$, $\beta \in \mathbb{R}$ and $x \in \mathbb{R}^n$. Then $\operatorname{proj}_D(\alpha x + \beta e) = \operatorname{proj}_D(x)$.

Proof. Let $\alpha > 0$, $\beta \in \mathbb{R}$ and $x \in \mathbb{R}^n$. With Lemma 16 and Lemma 19 it is sufficient to show

$$\operatorname{proj}_L(\operatorname{proj}_H(\alpha x + \beta e)) \overset{!}{=} \operatorname{proj}_L(\operatorname{proj}_H(x)).$$

Let $\tilde{x} := \alpha x + \beta e$, $\tilde{r} := \operatorname{proj}_H(\tilde{x})$ and $\tilde{s} := \operatorname{proj}_L(\tilde{r})$. Lemma 16 yields

$$\tilde{r} = (\alpha x + \beta e) + {}^1\!/_n \left(\lambda_1 - \alpha e^T x - \beta e^T e\right) e = \alpha x + {}^1\!/_n \left(\lambda_1 - \alpha e^T x\right) e,$$

where $e^T e = n$ was used. Hence $\tilde{r}$ is independent of β. Lemma 19 yields $\tilde{s} = m + \tilde{\delta}(\tilde{r} - m)$, where $\tilde{\delta} := \sqrt{\rho}/\|\tilde{r}-m\|_2$. With

$$\begin{aligned}
\|\tilde{r}\|_2^2 &= \|\alpha x\|_2^2 + \left\|{}^1\!/_n \left(\lambda_1 - \alpha e^T x\right) e\right\|_2^2 + 2\left\langle \alpha x,\ {}^1\!/_n \left(\lambda_1 - \alpha e^T x\right) e\right\rangle \\
&= \alpha^2 \|x\|_2^2 + {}^1\!/_{n^2} \left(\lambda_1 - \alpha e^T x\right)^2 \|e\|_2^2 + {}^2\!/_n \left(\lambda_1 - \alpha e^T x\right) \alpha e^T x \\
&= \alpha^2 \|x\|_2^2 + {}^1\!/_n \left(\lambda_1 - \alpha e^T x\right) \left(\lambda_1 - \alpha e^T x + 2\alpha e^T x\right) \\
&= \alpha^2 \|x\|_2^2 + {}^1\!/_n \left(\lambda_1^2 - \alpha^2 (e^T x)^2\right),
\end{aligned}$$

and with Remark 17 follows that $\|\tilde{r} - m\|_2^2 = \|\tilde{r}\|_2^2 - {}^{\lambda_1^2}\!/_n = \alpha^2\left(\|x\|_2^2 - {}^1\!/_n \cdot (e^T x)^2\right)$. Next, let $r := \operatorname{proj}_H(x)$ and $s := \operatorname{proj}_L(r)$, then Lemma 16 and Lemma 19 imply $r = x + {}^1\!/_n \cdot \left(\lambda_1 - e^T x\right) e$ and $s = m + \delta(r - m)$ where $\delta := \sqrt{\rho}/\|r-m\|_2$. Likewise $\|r - m\|_2^2 = \|x\|_2^2 - {}^1\!/_n \cdot (e^T x)^2$, and hence ${}^\delta\!/_{\tilde{\delta}} = {}^{\|\tilde{r}-m\|_2}\!/_{\|r-m\|_2} = \alpha$, where $\alpha > 0$ must hold. This yields

$$\tilde{s} = m + \tilde{\delta}(\tilde{r} - m) = m + {}^\delta\!/_\alpha \left(\alpha x + {}^{\lambda_1}\!/_n \cdot e - {}^\alpha\!/_n \cdot e^T x e - {}^{\lambda_1}\!/_n \cdot e\right) = m + \delta(x - {}^1\!/_n \cdot e^T x e) = s,$$

and the claim follows. □

Remark 22. Let $x \in \mathbb{R}^n$. When $s := \operatorname{proj}_L(\operatorname{proj}_H(x)) \in \mathbb{R}^n_{\geq 0}$, then already $s \in D$ and hence $s \in \operatorname{proj}_D(x)$. Therefore situations in which $s \notin \mathbb{R}^n_{\geq 0}$ will be investigated in the remainder of this chapter. It is evident from Figure 4.2c that the non-negativity constraint leads to the geometric structure known as simplex.

4.3.3 L_1 Norm Constraint And Non-Negativity – Target Simplex

Next consider the joint constraint of the target hyperplane with non-negativity. This yields a simplex, which is a geometric structure with certain regularities. The following definition is likewise to definitions from [71, 70]:

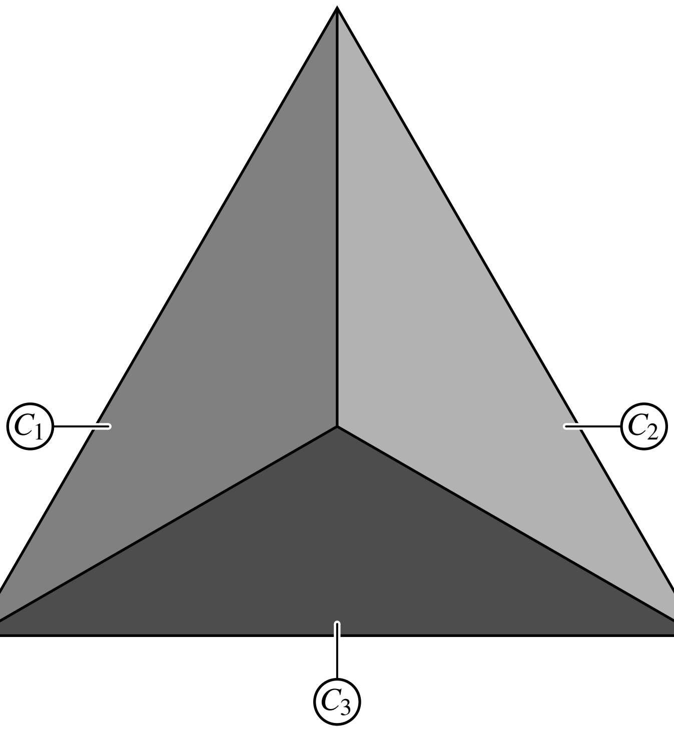


Figure 4.3: Illustration of Lemma 24: Tetrahedron or 4-simplex where three faces C_1, C_2 and C_3 are shown. For each face, one entry in the vector α is zero. Each face is thus isomorphic to a 3-simplex or equilateral triangle, embedded in a higher-dimensional space by padding zeros. By considering the boundary of the boundary sets, lines and points are gained, as 2- and 1-simplexes, respectively.

Definition 23. For $n \in \mathbb{N}$, $n \geq 1$, the set $\triangle^n := \{\alpha \in \mathbb{R}^n_{\geq 0} \mid e^T\alpha = 1\}$ is called *canonical n-simplex*. For an index set $I \subseteq \{1, \dots, n\}$, $\triangle^n_I := \{\alpha \in \triangle^n \mid \alpha_i = 0 \text{ for all } i \notin I\}$ is called *face* of $\triangle^n$ with respect to I.

The first observation is that trivial isomorphisms between canonical simplexes and its scaled versions exist, and that the boundary of simplexes in a topological sense is itself a simplex, albeit of lower dimensionality.

Lemma 24. Let $C := \mathbb{R}^n_{\geq 0} \cap H$.

(a) $C = \{\lambda_1\alpha \mid \alpha \in \triangle^n\}$, and thus $C \cong \triangle^n$.

(b) Let $I \subseteq \{1, \dots, n\}$ and $d := |I|$. Then $C_I := \{c \in C \mid c_i = 0 \text{ for all } i \notin I\} \cong \triangle^n_I \cong \triangle^d$.

(c) Let $c = \lambda_1\alpha \in C$ with $\alpha \in \triangle^n$. Then $c \in \partial C$ in the metric space $(H, \|\cdot\|_2)$ if and only if there is a $j \in \{1, \dots, n\}$ with $\alpha_j = 0$.

Proof. (a) and (b) are trivial.

(c) H is clearly a subspace of $\mathbb{R}^n$. Hence H and $\|\cdot\|_2$ restricted to H forms a metric space where topological properties can be investigated.

For "$\Rightarrow$", let $\alpha_i > 0$ for all $i \in \{1,\dots,n\}$. Let $\varepsilon := \lambda_1/2 \cdot \min_{i\in\{1,\dots,n\}} \alpha_i > 0$ and let $d \in H$ with $\|d-c\|_2 < \varepsilon$. For the claim to hold it has to be shown that $d \in C$ as well. Let $\beta := 1/\lambda_1 \cdot (d-c) + \alpha$. Then $\lambda_1\beta = d$ and $e^T\beta = 1$ because of $c,d \in H$. For $\beta \in \triangle^n$ it therefore suffices to show that $\beta_j \geq 0$ for all j. Let $j \in \{1,\dots,n\}$ be arbitrary and consider $\gamma := d_j - c_j$. It is $|\gamma| \leq \max_{i\in\{1,\dots,n\}} |d_i - c_i| = \|d-c\|_\infty \leq \|d-c\|_2 < \varepsilon$. If $\gamma \geq 0$, then $\beta_j = \gamma/\lambda_1 + \alpha_j \geq \alpha_j > 0$. If $\gamma < 0$, then $-\gamma = |\gamma| < \varepsilon$, hence $\beta_j = -|\gamma|/\lambda_1 + \alpha_j > -\varepsilon/\lambda_1 + \alpha_j = \alpha_j - 1/2 \cdot \min_{i\in\{1,\dots,n\}} \alpha_i \geq \alpha_j/2 > 0$. Thus $d \in C$.

For "$\Leftarrow$", let $j \in \{1,\dots,n\}$ with $\alpha_j = 0$. It has to be shown that in every neighborhood of c there is one element from C, and one not contained in C. The first condition is trivially satisfied. For the second condition, let $\varepsilon > 0$ be arbitrary but constant. Let $\delta := \varepsilon/2 \cdot (1 - 1/n)^{-1/2} > 0$. Let $d := \operatorname{proj}_H\left(c - \delta e_j\right) = c - \delta e_j + \delta/n \cdot e$. Clearly $d \in H$ and

$$\|d-c\|_2^2 = \left\|\delta/n \cdot e - \delta e_j\right\|_2^2 = \sum_{\substack{i=1\\ i\neq j}}^{n} (\delta/n)^2 + (\delta/n - \delta)^2 = \delta^2\,(1 - 1/n) = (\varepsilon/2)^2,$$

hence $\|d-c\|_2 = \varepsilon/2 < \varepsilon$. Assume $d \in C$, then there is a $\beta \in \triangle^n$ with $d = \lambda_1\beta$. Consider row j of d: $d_j = \lambda_1\beta_j = c_j - \delta + \delta/n = \lambda_1\alpha_j + \delta\,(1/n - 1) = \delta\,(1/n - 1) < 0$ for $n \geq 2$, thus $\beta_j < 0$, which is impossible. Hence c must have been on the boundary of C. □

Lemma 24 shows that the boundary set of a simplex is a face of the simplex, which is a simplex of lower dimensionality itself. Figure 4.3 illustrates this idea for $n = 4$. This is especially important, as it will be shown later that the projection onto a simplex from the outside must lie on the simplex boundary. Hence with Lemma 24 some entries of the projected point must become zero. Thus the problem dimension is reduced except for padding zeros, while the geometric properties remain the same.

Remark 25. Lemma 24 shows that C is convex because every point from C is a convex combination of $\lambda_1 e_1,\dots,\lambda_1 e_n$. With Remark 4 follows that unique projections onto C exist.

4.3.4 Inradius and Circumradius of the Scaled Canonical Simplex

Considering Figure 4.2c it is clear that the existence of points in $L \setminus C$ is strongly connected with the inradius and the circumradius of C. The question in which cases no projection onto C must take place is answered by Lemma 26 and its corollary. Figure 4.4 provides an overview for $n = 3$ for some arguments of the proof.

Lemma 26. The squared inradius of C is $\rho_{\text{in}} := \lambda_1^2/n(n-1)$, and the squared circumradius of C is $\rho_{\text{out}} := \lambda_1^2(n-1)/n$. The center of the associated spheres is m.

Proof. Because of C being closed and convex, it suffices to consider the distance between inner points and boundary points. Hence the insphere radius of a point p can be computed as being the minimum distance to any of the boundary points, $\rho_{\text{in}}(p) = \min_{c \in \partial C} \|p-c\|_2^2$. Analogously, the circumsphere radius is the maximum distance between p and all of the boundary points, $\rho_{\text{out}}(p) = \max_{c \in \partial C} \|p-c\|_2^2$. According to Lemma 24(c), boundary points are exactly that points where at least one entry vanishes. It will be shown first that $\rho_{\text{in}}(m) = \lambda_1^2/n(n-1)$ and $\rho_{\text{out}}(m) = \lambda_1^2(n-1)/n$. Then it will be shown that m is the center of the optimal spheres, that is $\arg\max_{p \in C} \rho_{\text{in}}(p) = m$ and $\arg\min_{p \in C} \rho_{\text{out}}(p) = m$.

Let $c \in \partial C$ and $I := \{\, i \in \{1,\dots,n\} \mid c_i \neq 0 \,\}$. With Lemma 24(c) follows $I \neq \{1,\dots,n\}$, and clearly $I = \varnothing$ would contradict $e^T c = \lambda_1$. Consider the Lagrangian $L(c,\mu) := \|m-c\|_2^2 + \mu\left(e_I^T c - \lambda_1\right)$ where $e_I := \sum_{i \in I} e_i \in \mathbb{R}^n$. With Remark 17 follows $\|m-c\|_2^2 = \|c\|_2^2 - \lambda_1^2/n$, and thus $(\partial L/\partial c)^T = 2c + \mu e_I \overset{!}{=} 0 \in \mathbb{R}^n$ and $\partial L/\partial \mu = e^T c - \lambda_1 \overset{!}{=} 0 \in \mathbb{R}$. Therefore, $0 = e^T (\partial L/\partial c)^T = 2e^T c + \mu e^T e_I = 2\lambda_1 + \mu \cdot |I|$, hence $\mu = -2\lambda_1/|I|$, and so $c = \lambda_1/|I| \cdot e_I$ in an optimum of $\|m-c\|_2^2$ for $c \in \partial C$. Thus in an optimum follows $\|m-c\|_2^2 = \|c\|_2^2 - \lambda_1^2/n = \lambda_1^2\left(1/|I| - 1/n\right)$.

For the squared inradius and squared circumradius follows

$$\rho_{\text{in}}(m) = \min\nolimits_{\varnothing \neq I \subsetneq \{1,\dots,n\}} \lambda_1^2\left(1/|I| - 1/n\right) = \lambda_1^2\left(1/(n-1) - 1/n\right) = \lambda_1^2/n(n-1), \text{ and}$$
$$\rho_{\text{out}}(m) = \max\nolimits_{\varnothing \neq I \subsetneq \{1,\dots,n\}} \lambda_1^2\left(1/|I| - 1/n\right) = \lambda_1^2\left(1/1 - 1/n\right) = \lambda_1^2(n-1)/n.$$

The optimum for the insphere is achieved for border points $\lambda_1/(n-1) \cdot e_I$ for $|I| = n-1$, corresponding to points d_1, d_2 and d_3 in Figure 4.4. The optimum for the circumsphere is achieved for corner points $\lambda_1 e_i$ where $I = \{\, i \,\}$, corresponding to points c_1, c_2 and c_3 in Figure 4.4.

It remains to be shown that no other point is center of a larger insphere or a smaller circumsphere. Let $p \in C \setminus \partial C$ and $i^* := \arg\min_{i \in \{1,\dots,n\}} p_i$ be the index of a minimum entry. Because $p \in H$, $\lambda_1 = p_{i^*} + \sum_{i \in \{1,\dots,n\} \setminus \{i^*\}} p_i \geq p_{i^*} + (n-1)p_{i^*} = np_{i^*}$, so $p_{i^*} \leq \lambda_1/n$. A point $d \in \partial C$ can be constructed as follows such that $\|p-d\|_2^2 \leq \rho_{\text{in}}(m)$, showing that m is the optimal center of an insphere of C. It is $p_{i^*} > 0$ because of $p \notin \partial C$ and Lemma 24(c). Let $d := p - p_{i^*} e_{i^*} + 1/(n-1) \cdot p_{i^*}(e - e_{i^*})$. Clearly $e^T d = \lambda_1$, $d_{i^*} = 0$ and $d_i \geq 0$ for all $i \in \{1,\dots,n\}$, so $d \in \partial C$. Hence,

$$\begin{aligned}\rho_{\text{in}}(p) &= \min_{c \in \partial C} \|p-c\|_2^2 \leq \|p-d\|_2^2 = \|p_{i^*} e_{i^*} - 1/(n-1) \cdot p_{i^*}(e - e_{i^*})\|_2^2 \\ &= p_{i^*}^2 + (n-1) \cdot p_{i^*}^2/(n-1)^2 = p_{i^*}^2 \cdot \tfrac{n}{n-1} \leq \tfrac{\lambda_1^2}{n^2} \cdot \tfrac{n}{n-1} = \rho_{\text{in}}(m),\end{aligned}$$

showing m is optimal with respect to the maximum insphere radius.

For showing the analogous statement for an optimal circumsphere of C, first note that Proposition 2 and $p \in C$ imply $\|p\|_2^2 \geq \|p\|_1^2/n = \lambda_1^2/n$. Now let $d := \lambda_1 e_{i^*} \in \partial C$ be the corner point

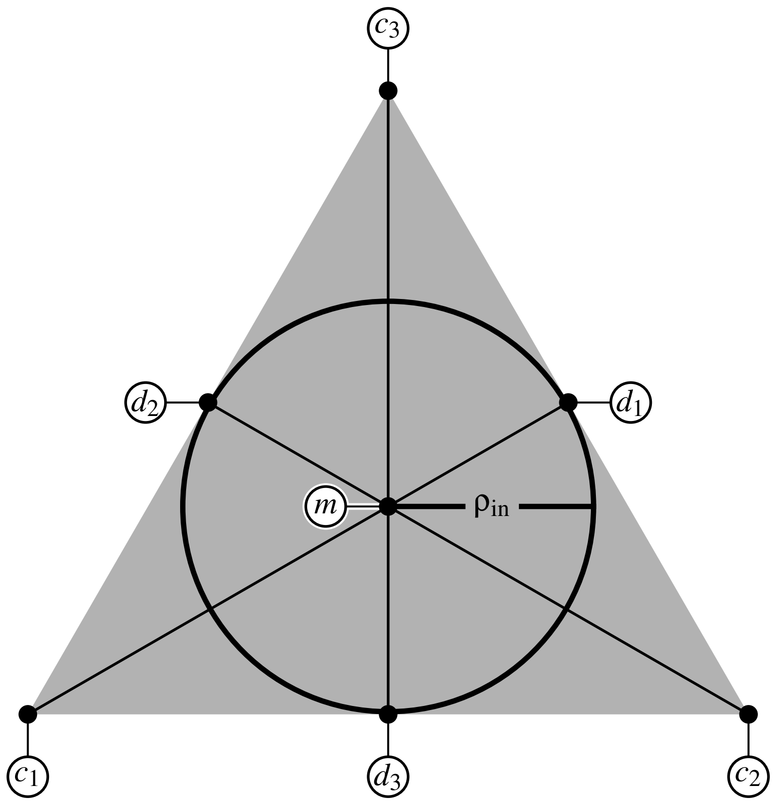


Figure 4.4: According to Lemma 26, m is the center of the largest insphere and the smallest circumsphere (latter one not shown in graphics) of C, which have squared radius ρ_{in} and ρ_{out}, respectively. For the insphere, the corresponding points on ∂C are d_1, d_2 and d_3, and lie on the middle of the simplex boundary lines. For the circumsphere, the corresponding points are the corner points c_1, c_2 and c_3.

associated with the minimum entry of p. Then using $p_{i^*} \leq {}^{\lambda_1}/_{n}$ as shown above yields

$$\begin{aligned}\rho_{\text{out}}(p) &= \max_{c \in \partial C} \|p - c\|_2^2 \geq \|p - d\|_2^2 = \|p\|_2^2 + \|d\|_2^2 - 2\lambda_1 p_{i^*} \\ &\geq {}^{\lambda_1^2}/_{n} + \lambda_1^2 - {}^{2\lambda_1^2}/_{n} = {}^{\lambda_1^2 (n-1)}/_{n} = \rho_{\text{out}}(m),\end{aligned}$$

so m is an optimal center of the circumsphere of C. $\square$

Note that the construction of $d \in \partial C$ in the arguments for showing that m is an optimal center of an insphere is essentially the same as will be used in the forthcoming Lemma 32.

Corollary 27. The maximum sparseness degree such that $\text{proj}_D(x) = \text{proj}_L(\text{proj}_H(x))$ for all $x \in \mathbb{R}^n$ is bounded from above and converges decreasingly to zero. In the case of $n = 2$, $\text{proj}_D(x) = \text{proj}_L(\text{proj}_H(x))$ always holds.

Table 4.2: Maximum sparseness $\sigma_{\max}$ in dependence of the problem dimensionality n in which cases no simplex projection has to be carried out.

n	2	3	4	5	6	7	8	9	10	25	50
$\sigma_{\max}$	1.00	0.43	0.27	0.19	0.15	0.12	0.10	0.09	0.08	0.03	0.01

Proof. Clearly, $\operatorname{proj}_L(\operatorname{proj}_H(x)) \subseteq \mathbb{R}^n_{\geq 0}$ for all $x \in \mathbb{R}^n$ if and only if $\rho \leq \rho_{\text{in}}$ using Lemma 18, Lemma 26 and Remark 22. This is the case, when $\lambda_2 \leq \lambda_1/\sqrt{n-1}$, hence when

$$\sigma \leq \sigma_{\max} := \frac{\sqrt{n} - \sqrt{n-1}}{\sqrt{n} - 1} \to 0 \ (n \to \infty).$$

Table 4.2 gives $\sigma_{\max}$ for some values of n. When $n = 2$ and $\lambda_2 \leq \lambda_1$, which is fulfilled by requirement on λ_1 and λ_2, then $\rho = \lambda_2^2 - \lambda_1^2/n \leq \lambda_1^2/2 = \rho_{\text{in}}$, and the claim follows with Lemma 18. Thus, for $n = 2$ a non-negativity projection has not to be carried out regardless of the sparseness degree. For $n \geq 25$, that projection can only be omitted for very low sparseness degrees. □

4.3.5 Projections onto Simplexes

With Corollary 27 follows that for almost all sparseness degrees there are points $s \in L \setminus C$ that need to be projected onto D. First consider the projection onto canonical simplexes and a necessary condition for a solution [68, 71]:

Lemma 28. Let $x \in \mathbb{R}^n$, $x \notin \triangle^n$. Let $r := \operatorname{proj}_{\triangle^n}(x)$.

(a) There exists $\hat{t} \in \mathbb{R}$ with $r = \max(x - \hat{t} \cdot e,\ 0)$, where the maximum is taken element-wise.

(b) Let $\tau \in S_n$ such that $y := P_\tau x$ is sorted in descending order. Let $t_j := 1/j \cdot \left(\sum_{i=1}^{j} y_i - 1\right) \in \mathbb{R}$ for $j \in \{1, \dots, n\}$. There is exactly one $j^* \in \{1, \dots, n\}$ such that $\hat{t} = t_{j^*}$, characterized by either $y_{j^*} \geq \hat{t} \geq y_{j^*+1}$ for $j^* \in \{1, \dots, n-1\}$ or $y_n \geq \hat{t}$ if $j^* = n$.

(c) Algorithm 4.1 computes $\hat{t}$ such that (a) holds.

Proof. The point r exists and is unique with Remark 25. The remainder is shown in [71]. □

The run-time of Algorithm 4.1 is clearly quasilinear since the argument needs to be sorted [23]. An alternative was proposed by [72], who have shown that the number $\hat{t}$ from Lemma 28(a) can also be found as the unique zero of the monotonic function $t \mapsto \|\max(|x| - t \cdot e,\ 0)\|_1 - 1$. The zero of this auxiliary function can be found in linear time using the Bisection method [72]. In this work an explicit, closed-form expression for $\hat{t}$ is preferable as this facilitates analysis of the properties of the projected point.

Algorithm 4.1: Computation of $\hat{t}$ for $\operatorname{proj}_{\triangle^n}$ [71].

```
Input: $x \in \mathbb{R}^n$.
Output: $\hat{t} \in \mathbb{R}$ such that $\operatorname{proj}_{\triangle^n}(x) = \max(x - \hat{t} \cdot e,\ 0)$.
// Sort the input vector.
1 Let $\tau$ be a permutation of $\{1, \dots, n\}$ such that $x_{\tau(1)} \geq \dots \geq x_{\tau(n)}$;
2 Let $y \in \mathbb{R}^n$ with $y_i := x_{\tau(i)}$ for $i \in \{1, \dots, n\}$;
// Try all possibilities for $\hat{t}$ and return the only feasible one.
3 $s := 0$;
4 for $i := 1$ to $n - 1$ do
5     $s := s + y_i$;
6     $t := \frac{s-1}{i}$;
7     if $t \geq y_{i+1}$ then return $t$;
8 end
9 return $\frac{s + y_n - 1}{n}$;
```

The next result states that a projection from $L \setminus C$ onto D must lie on the boundary of C in the metric space $(H, \|\cdot\|_2)$. Lemma 24 then gives a characterization of the projected point, namely that the number of vanishing entries must increase. The idea of the proof of Lemma 29 is illustrated in Figure 4.5.

Lemma 29. Let $s \in L \setminus C$. Then $\operatorname{proj}_D(s) \subseteq \partial C$ in $(H, \|\cdot\|_2)$.

Proof. Let $p \in \operatorname{proj}_D(s)$ and assume $p \notin \partial C$. With the triangle inequality, $\|p - s\|_2^2 \leq 4\rho$. Assume equality, then $m = \frac{p+s}{2}$ with Lemma 18(d). Assume there is a $\gamma \in \mathbb{R}$ such that $p = \gamma e$. Then $\lambda_2^2 = \|p\|_2^2 = n\gamma^2$ and $\lambda_1 = \|p\|_1 = \gamma n$, hence $\lambda_2^2 = \lambda_1^2/n$, hence $\rho = 0$, which contradicts the existence of $s \in L \setminus C$ according to Lemma 18. Thus there must be two indices $j \neq k$ with $p_j \neq p_k$. Let $\tau := (j, k) \in S_n$ and $q := P_\tau p$. D is permutation-invariant as intersection of permutation-invariant sets, hence $q \in D$. Because of $p \neq q$, it is $\frac{q+s}{2} \neq m$, hence with Lemma 18(d) one obtains $\|q - s\|_2^2 < 4\rho = \|p - s\|_2^2$, contradicting the minimality of p. Hence it can be assumed that $\|p - s\|_2^2 < 4\rho$.

Still assuming $p \notin \partial C$, there is an $\varepsilon > 0$ such that from $q \in H$ and $\|q - p\|_2 < \varepsilon$ follows $q \in C$. Define $\gamma := \varepsilon^2 / 2\|s-p\|_2^2$. Because of $s \notin C$, $0 < \varepsilon < \|s - p\|_2$ and hence $0 < \gamma < 1/2$. Define $r := p + \gamma(s - p)$ and $\delta := \sqrt{\rho}/\|r-m\|_2$. Clearly $r \in H$. With Lemma 18(c) follows:

$$\|r\|_2^2 = \|p\|_2^2 + 2\gamma\langle p,\ s - p\rangle + \gamma^2 \|s - p\|_2^2 = \lambda_2^2 - \gamma(1 - \gamma)\|p - s\|_2^2.$$

Using Remark 17, $\|r - m\|_2^2 = \rho - \gamma(1 - \gamma)\|p - s\|_2^2$. It is $\gamma(1 - \gamma) \in (0,\ 1/4)$ since $\gamma \in (0,\ 1/2)$, thus $\|r - m\|_2^2 < \rho$, hence $\delta > 1$. Define $q := m + \delta(r - m)$. It is $q \in H$. Because of Remark 17,

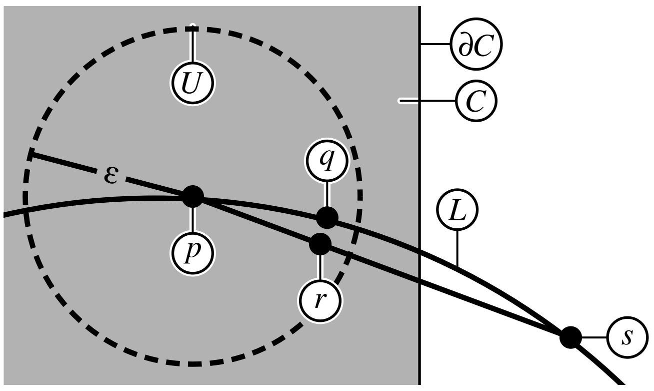


Figure 4.5: Sketch of the proof of Lemma 29: The projection p of s onto D must lie on ∂C. If p belongs to the interior of C, a neighborhood $U := \{ x \in H \mid \|x-h\|_2 < \varepsilon \} \subseteq C$ exists. Within U, a point q closer to s than p can be constructed, violating the minimality of the projection.

$\|q\|_2^2 = \|m\|_2^2 + \delta^2 \|r-m\|_2^2 = \lambda_2^2$, thus $q \in K$. Using Remark 17 and Lemma 18(c):

$$\begin{aligned}\langle p,\, q\rangle &= (1-\delta)\langle p,\, m\rangle + \delta\langle p,\, r\rangle = (1-\delta)\cdot {\lambda_1^2}/{n} + \delta\langle p,\, p+\gamma(s-p)\rangle \\ &= {\lambda_1^2}/{n} - \delta\cdot{\lambda_1^2}/{n} + \delta\lambda_2^2 + \delta\gamma\big(\langle p,\, s\rangle - \lambda_2^2\big) = {\lambda_1^2}/{n} + \delta\rho - {1}/{2}\cdot\delta\gamma\|p-s\|_2^2.\end{aligned}$$

Therefore using $1-\delta<0$:

$$\|p-q\|_2^2 = 2\lambda_2^2 - 2\langle p,\, q\rangle = 2\rho(1-\delta) + \delta\gamma\|p-s\|_2^2 < \delta\gamma\|p-s\|_2^2.$$

One obtains $\varepsilon \leq \sqrt{\rho_{\text{in}}}$ with Lemma 26. Using Corollary 27 it must be $\rho_{\text{in}} \leq \rho$, because otherwise $s \in L \setminus C$ would not exist. Hence $\varepsilon \leq \sqrt{\rho}$. With $\gamma \in (0,\, {1}/{2})$ it is $\gamma(1-\gamma) \leq \gamma$, hence

$$\delta^2 = \frac{\rho}{\rho - \gamma(1-\gamma)\|p-s\|_2^2} \leq \frac{\rho}{\rho-\gamma\|p-s\|_2^2} = \frac{\rho}{\rho-\frac{\varepsilon^2}{2}} \leq \frac{\rho}{\rho-\frac{\rho}{2}} = 2.$$

Therefore $\|p-q\|_2^2 < {\varepsilon^2}/{\sqrt{2}}$, which implies $\|p-q\|_2 < \varepsilon$, so $q \in C$, so $q \in D$.

It is $\langle m,\, s-p\rangle = 0$ with Remark 17. With Lemma 18(c) follows $\langle p,\, s-p\rangle = -{1}/{2}\,\|p-s\|_2^2$. Therefore,

$$\begin{aligned}\langle q,\, s-p\rangle &= (1-\delta)\langle m,\, s-p\rangle + \delta\langle r,\, s-p\rangle = \delta\langle p+\gamma(s-p),\, s-p\rangle \\ &= \delta\big(\langle p,\, s-p\rangle + \gamma\|s-p\|_2^2\big) = \|p-s\|_2^2\,\delta(\gamma-{1}/{2}),\end{aligned}$$

thus $\langle q-p,\, s-p\rangle = \|p-s\|_2^2\,[\delta(\gamma-{1}/{2})+{1}/{2}]$. Application of Proposition 1 yields:

$$\begin{aligned}\|q-s\|_2^2 &= \|q-p\|_2^2 + 2\langle q-p,\, p-s\rangle + \|p-s\|_2^2 \\ &= 2\rho(1-\delta) + \delta\gamma\|p-s\|_2^2 + \|p-s\|_2^2\,[\delta(1-2\gamma)-1] + \|p-s\|_2^2 \\ &= 2\rho(1-\delta) + \|p-s\|_2^2\,\delta(1-\gamma).\end{aligned}$$

With $\|p-s\|_2^2 < 4\rho$ and $1-\delta < 0$ it is

$$\|q-s\|_2^2 < {}^1\!/_2\,\|p-s\|_2^2\,(1-\delta) + \|p-s\|_2^2\,\delta\,(1-\gamma) = \|p-s\|_2^2\,[{}^1\!/_2 + \delta\,({}^1\!/_2-\gamma)],$$

and

$$\delta^2 = \frac{\rho}{\rho-\gamma(1-\gamma)\,\|p-s\|_2^2} < \frac{\rho}{\rho-4\rho\gamma(1-\gamma)} = \frac{1}{(1-2\gamma)^2}.$$

Hence $\delta\,({}^1\!/_2-\gamma) < {}^1\!/_2$, so $\|q-s\|_2^2 < \|p-s\|_2^2$, which is a contradiction of p being the projection of s onto D. Thus p must have been in ∂C in the first place. □

Remark 30. In a similar fashion, it can be shown that $\mathrm{proj}_C(s) \subseteq \partial C$ in $(H, \|\cdot\|_2)$. For this, the point q from the proof can be constructed to lie on the line between s and the hypothetical projection p, instead of having to lie on L.

It is important that the topological boundary of C is always considered *relative* to the subspace H of $\mathbb{R}^n$. Otherwise, the existence of vanishing entries in the projection due to Lemma 24(c) would not hold. As a simple example, consider the origin $0 \in \mathbb{R}^n$ where all entries are equal to zero. Clearly, the projection of 0 onto C is the mid-point m, see also Remark 17, but $m \notin \partial C$ in $(H, \|\cdot\|_2)$. Because $0 \notin H$ this is no counterexample to Lemma 29 and Remark 30. However, $m \in \partial C$ in $(\mathbb{R}^n, \|\cdot\|_2)$ because $C = \partial C$ in $\mathbb{R}^n$, since there are no interior points of C in $\mathbb{R}^n$. This simple argument demonstrates the importance of relative topology. Due to Lemma 16 it can always be assumed that the working point lies on H, and the statements of Lemma 29 and Remark 30 hold in the context discussed here.

4.3.6 Projections onto Faces of Simplexes

First consider some definitions about the structure of faces of simplexes:

Definition 31. Let $I \subseteq \{1,\ldots,n\}$ and $d := |I|$. Like in Lemma 24, let the face of C with respect to index set I be denoted by $C_I := \{c \in C \mid c_i = 0 \text{ for all } i \notin I\}$. Let $m_I := {}^{\lambda_1}\!/_d \cdot \sum_{i \in I} e_i \in C_I$ be the mid-point of C_I, and let $\rho_I := \lambda_2^2 - {}^{\lambda_1^2}\!/_d$.

As will be shown later, ρ_I is the distance between m_I and all points $a \in L$ with $a_i = 0$ for all $i \notin I$. m_I is the barycenter of C_I, and for all $c \in C_I$ follows $\langle m - m_I,\ m_I - c\rangle = 0$.

A fundamental part in the sparseness-enforcing projection operator is the generation of vanishing entries. With the results from Lemma 28 it can be conjectured that the simplex projection generates zero entries. For proving that the solution with respect to π has zeros at the same coordinates, first consider projections from within a simplex onto one of its faces, that is with Lemma 24 a face of a simplex where certain entries must vanish. Lemma 32 provides the explicit construction of such a projection, and gives a statement on the norm of the projection when additional requirements hold:

Lemma 32. Let $q \in C$ and $I \subseteq \{1,\dots,n\}$. Then there exists an $s \in C_I$ with $\|q-v\|_2^2 = \|q-s\|_2^2 + \|s-v\|_2^2$ for all $v \in C_I$. If with $J := \{1,\dots,n\} \setminus I$ additionally $\max_{j \in J} q_j \le \min_{i \in I} q_i$ holds, then $\|s\|_2 \ge \|q\|_2$ with equality if and only if $q_j = 0$ for all $j \in J$.

More precisely, let $h := |J|$ and let $J = \{j_1,\dots,j_h\}$ such that $q_{j_1} \le \dots \le q_{j_h}$. Define $s^{(0)} := q$, and for $k \in \{1,\dots,h\}$ let

$$s^{(k)} := s^{(k-1)} - s^{(k-1)}_{j_k} e_{j_k} + \tfrac{1}{n-k} s^{(k-1)}_{j_k} \left(e - \textstyle\sum_{i=1}^{k} e_{j_i}\right),$$

and $s := s^{(h)}$. Then the following holds:

(a) $s^{(k)} \in C_{\{j_1,\dots,j_k\}^C}$ for all $k \in \{1,\dots,h\}$.

(b) $\left\langle s^{(0)} - s^{(k)},\ s^{(k)} - s^{(k+1)} \right\rangle = 0$ for all $k \in \{0,\dots,h-1\}$.

(c) $\left\|s^{(k)} - q\right\|_2^2 = \sum_{i=1}^{k} \left\|s^{(i)} - s^{(i-1)}\right\|_2^2$ for all $k \in \{0,\dots,h\}$.

(d) $s^{(k-1)}_{j_k} = q_{j_k} + \frac{1}{n-k+1} \sum_{i=1}^{k-1} q_{j_i}$ for all $k \in \{1,\dots,h\}$.

(e) $\left\langle s^{(0)} - s^{(k)},\ s^{(k)} - v \right\rangle = 0$ for all $k \in \{0,\dots,h\}$ and for all $v \in C_I$.

(f) $\|q-v\|_2^2 = \left\|q - s^{(k)}\right\|_2^2 + \left\|s^{(k)} - v\right\|_2^2$ for all $k \in \{0,\dots,h\}$ and for all $v \in C_I$.

(g) $s = \operatorname{proj}_{C_I}(q)$.

If $\max_{j \in J} q_j \le \min_{i \in I} q_i$, then the following holds as well:

(h) $s^{(k)}_{j_1} \le \dots \le s^{(k)}_{j_h} \le \min_{i \in I} s^{(k)}_i$ for all $k \in \{0,\dots,h\}$.

(i) $s^{(k-1)}_{j_k} \le \frac{\lambda_1}{n-k+1}$ for all $k \in \{1,\dots,h\}$.

(j) $\left\|s^{(k-1)}\right\|_2 \le \left\|s^{(k)}\right\|_2$ for all $k \in \{1,\dots,h\}$, and hence $\|s\|_2 \ge \|q\|_2$.

(k) $\|s\|_2 = \|q\|_2$ if and only if $q_j = 0$ for all $j \in J$.

Proof. First note that $s^{(k)}$ is constructed from $s^{(k-1)}$ by setting entry j_k to zero, and adjusting all remaining entries, but the ones previously set to zero, such that the L_1 norm is preserved. This generates a series of points coming closer to C_I, see (a), where the final point is from C_I. As all relevant dot products vanish, see (b) and (e), this is a recursive process of orthogonal projections. Hence the distance between points can be computed using the Pythagorean theorem, see (c) and (f). In (g) it is then shown that s is the unique projection of q onto C_I.

If the entry j_k in $s^{(k-1)}$ does not vanish, then the L_2 norm of the newly constructed point is greater than that of the original point, see (j) and (k). The entries with indices from J must be sufficiently small for this non-decreasing norm property to hold, see (h) and (i). The magnitude of these entries, however, is strongly connected with the magnitudes of respective entries from

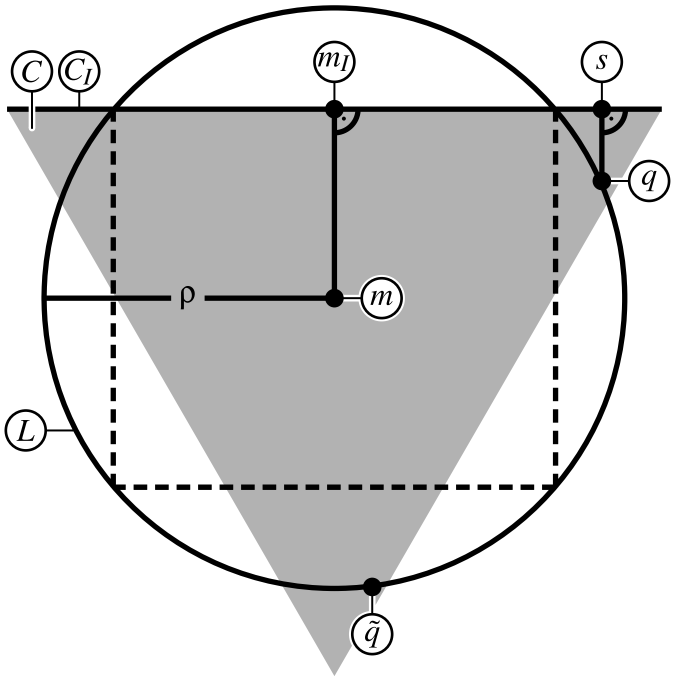


Figure 4.6: Lemma 32: Projection of a point q from within a simplex C onto one of its faces C_I yields point s. If q is sufficiently close to C_I, than $\|s\|_2 \geq \|q\|_2$. Otherwise this does not hold, see for example point $\tilde{q}$ which is located outside the dashed square with edge length $2\,\|m_I - m\|_2$. Adapted from [66].

the original point q, that is the rank is preserved from one point to its successor. Figure 4.6 gives an example for $n = 3$ in which cases the non-decreasing norm property holds.

For $k \in \{1, \dots, h\}$ let $a_k := \frac{1}{n-k}\left(e - \sum_{i=1}^{k} e_{j_i}\right) - e_{j_k} \in \mathbb{R}^n$. Then $s^{(k)} = s^{(k-1)} + s_{j_k}^{(k-1)} a_k$, and with induction follows $s^{(k)} = s^{(0)} + \sum_{i=1}^{k} s_{j_i}^{(i-1)} a_i$ for $k \in \{1, \dots, h\}$. This abbreviation will be used for computing norms and scalar products of the points $s^{(1)}, \dots, s^{(h)}$.

(a) First note that $e^T a_k = \frac{1}{n-k}(n-k) - 1 = 0$ and that $a_k \in \mathbb{R}^n_{\geq 0}$ for all $k \in \{1, \dots, h\}$. It is now shown by induction that $s^{(k)}$ lies on the claimed face of C. For $k = 1$, it is $s_{j_1}^{(1)} = 0$ and $s_i^{(1)} = q_i + \frac{1}{n-1} q_{j_1}$ for $i \neq j_1$. Thus $s_i^{(1)} \geq 0$ for all $i \in \{1, \dots, n\}$ because of $q_i \geq 0$ for all $i \in \{1, \dots, n\}$. Furthermore $e^T s^{(1)} = e^T q + q_{j_1} e^T a_1 = e^T q = \lambda_1$, hence $s^{(1)} \in C_{\{1,\dots,n\} \setminus \{j_1\}}$.

For $k - 1 \to k$, assume $s_i^{(k-1)} = 0$ for all $i \in \{j_1, \dots, j_{k-1}\}$, $s_i^{(k-1)} \geq 0$ for all $i \in \{1, \dots, n\}$ and $e^T s^{(k-1)} = \lambda_1$. Clearly, for $i \in \{j_1, \dots, j_{k-1}\}$ it is $s_i^{(k)} = s_i^{(k-1)} = 0$. Furthermore, $s_{j_k}^{(k)} = s_{j_k}^{(k-1)} - s_{j_k}^{(k-1)} = 0$. With $s^{(k-1)} \in \mathbb{R}^n_{\geq 0}$ and $a_k \in \mathbb{R}^n_{\geq 0}$ follows $s^{(k)} \in \mathbb{R}^n_{\geq 0}$. At last, $e^T s^{(k)} = e^T s^{(k-1)} + s_{j_k}^{(k-1)} e^T a_k = e^T s^{(k-1)} = \lambda_1$. Hence $s^{(k)} \in C_{\{1,\dots,n\} \setminus \{j_1,\dots,j_k\}}$.

(b) First note that for $i \in \{1, \dots, k\}$ it is

$$\begin{aligned}
&\left\langle e - \sum\nolimits_{\mu=1}^{i} e_{j_\mu},\ e - \sum\nolimits_{\nu=1}^{k+1} e_{j_\nu} \right\rangle \\
=&\langle e,\ e\rangle - \left\langle e,\ \sum\nolimits_{\nu=1}^{k+1} e_{j_\nu} \right\rangle - \left\langle \sum\nolimits_{\mu=1}^{i} e_{j_\mu},\ e \right\rangle + \left\langle \sum\nolimits_{\mu=1}^{i} e_{j_\mu},\ \sum\nolimits_{\nu=1}^{k+1} e_{j_\nu} \right\rangle \\
=&n - (k+1) - i + i = n - k - 1,
\end{aligned}$$

and therefore

$$\begin{aligned}
\langle a_i,\ a_{k+1}\rangle &= \tfrac{1}{(n-i)(n-k-1)} \left\langle e - \sum\nolimits_{\mu=1}^{i} e_{j_\mu},\ e - \sum\nolimits_{\nu=1}^{k+1} e_{j_\nu} \right\rangle + \left\langle e_{j_i},\ e_{j_{k+1}} \right\rangle \\
&\quad - \tfrac{1}{n-i} \left\langle e - \sum\nolimits_{\mu=1}^{i} e_{j_\mu},\ e_{j_{k+1}} \right\rangle - \tfrac{1}{n-k-1} \left\langle e_{j_i},\ e - \sum\nolimits_{\nu=1}^{k+1} e_{j_\nu} \right\rangle \\
&= \tfrac{n-k-1}{(n-i)(n-k-1)} + 0 - \tfrac{1}{n-i} - \tfrac{0}{n-k-1} = 0.
\end{aligned}$$

Thus

$$\left\langle s^{(0)} - s^{(k)},\ s^{(k)} - s^{(k+1)} \right\rangle = \left\langle \sum\nolimits_{i=1}^{k} s_{j_i}^{(i-1)} a_i,\ s_{j_{k+1}}^{(k)} a_{k+1} \right\rangle = \sum\nolimits_{i=1}^{k} s_{j_i}^{(i-1)} s_{j_{k+1}}^{(k)} \langle a_i,\ a_{k+1}\rangle = 0.$$

(c) Proof by induction. For $k \in \{0,\ 1\}$ the claim trivially holds. For $k-1 \to k$ and using Proposition 1,

$$\begin{aligned}
\left\| s^{(k)} - q \right\|_2^2 &\overset{\text{P. 1}}{=} \left\| s^{(k)} - s^{(k-1)} \right\|_2^2 + \left\| s^{(k-1)} - q \right\|_2^2 + \underbrace{2\left\langle s^{(k)} - s^{(k-1)},\ s^{(k-1)} - q \right\rangle}_{=0 \,\Leftarrow\, \text{(b)}} \\
&\overset{\text{I.H.}}{=} \left\| s^{(k)} - s^{(k-1)} \right\|_2^2 + \sum\nolimits_{i=1}^{k-1} \left\| s^{(i)} - s^{(i-1)} \right\|_2^2 = \sum\nolimits_{i=1}^{k} \left\| s^{(i)} - s^{(i-1)} \right\|_2^2.
\end{aligned}$$

(d) For $i \in \{1, \dots, k-1\}$ follows

$$\left\langle e_{j_k},\ a_i \right\rangle = \tfrac{1}{n-i} \left(\left\langle e_{j_k},\ e \right\rangle - \left\langle e_{j_k},\ \sum\nolimits_{\mu=1}^{i} e_{j_\mu} \right\rangle \right) - \left\langle e_{j_k},\ e_{j_i} \right\rangle = \tfrac{1}{n-i}(1-0) - 0 = \tfrac{1}{n-i},$$

hence

$$\begin{aligned}
s_{j_k}^{(k-1)} &= \left\langle e_{j_k},\ s^{(k-1)} \right\rangle = \left\langle e_{j_k},\ s^{(0)} \right\rangle + \sum\nolimits_{i=1}^{k-1} s_{j_i}^{(i-1)} \left\langle e_{j_k},\ a_i \right\rangle = q_{j_k} + \sum\nolimits_{i=1}^{k-1} \tfrac{1}{n-i} s_{j_i}^{(i-1)} \\
&= q_{j_k} + \sum\nolimits_{i=1}^{k-1} \tfrac{1}{n-i} q_{j_i} + \underbrace{\sum\nolimits_{i=1}^{k-1} \sum\nolimits_{\mu=1}^{i-1}}_{=\sum_{1 \leq \mu < i \leq k-1}} \tfrac{1}{n-i} \tfrac{1}{n-i+1} q_{j_\mu} \\
&= q_{j_k} + \sum\nolimits_{i=1}^{k-1} \left[\tfrac{1}{n-i} q_{j_i} + \sum\nolimits_{\mu=i+1}^{k-1} \tfrac{1}{n-\mu} \tfrac{1}{n-\mu+1} q_{j_i} \right] \\
&= q_{j_k} + \sum\nolimits_{i=1}^{k-1} q_{j_i} \left[\tfrac{1}{n-i} + \sum\nolimits_{\mu=i+1}^{k-1} \tfrac{1}{n-\mu} \tfrac{1}{n-\mu+1} \right],
\end{aligned}$$

and for the claim to hold it is sufficient to show

$$\tfrac{1}{n-i} + \sum\nolimits_{\mu=i+1}^{k-1} \tfrac{1}{n-\mu} \tfrac{1}{n-\mu+1} \overset{!}{=} \tfrac{1}{n-k+1}$$

for all $i \in \{1,\dots,k-1\}$. This is shown by reverse induction. For $i = k-1$, the identity trivially holds. For $i+1 \to i$ it is

$$\frac{1}{n-i} + \sum_{\mu=i+1}^{k-1} \frac{1}{n-\mu}\frac{1}{n-\mu+1} = \frac{1}{n-i} - \frac{1}{n-(i+1)} + \underbrace{\frac{1}{n-(i+1)} + \sum_{\mu=i+2}^{k-1} \frac{1}{n-\mu}\frac{1}{n-\mu+1}}_{=\frac{1}{n-k+1} \Leftarrow \text{I.H.}} + \frac{1}{n-(i+1)}\frac{1}{n-(i+1)+1}$$
$$= \frac{1}{n-i} - \frac{1}{n-i-1} + \frac{1}{n-k+1} + \frac{1}{n-i-1}\frac{1}{n-i} = \frac{1}{n-k+1} + \underbrace{\frac{(n-i-1)-(n-i)+1}{(n-i-1)(n-i)}}_{=0},$$

and the claim follows.

(e) Proof by induction. Let $v \in C_I$, that is $e^T v = \lambda_1$ and $v_j = 0$ for all $j \in J$. For $k = 0$, $s^{(0)} - s^{(k)} = 0$ and the claim follows. For $k-1 \to k$, first note that $\langle a_k,\, q\rangle = \frac{1}{n-k}\left(\lambda_1 - \sum_{i=1}^{k} q_{j_i}\right) - q_{j_k}$, $\left\langle a_k,\, s^{(k-1)}\right\rangle = \left\langle a_k,\, s^{(0)}\right\rangle + \sum_{i=1}^{k-1} s_{j_i}^{(i-1)} \langle a_k,\, a_i\rangle = \langle a_k,\, q\rangle$ because $\langle a_k,\, a_i\rangle = 0$ for all $i \in \{1,\dots,k-1\}$ as shown in (b), thus $\left\langle a_k,\, s^{(0)} - 2s^{(k-1)}\right\rangle = -\langle a_k,\, q\rangle$. Moreover it is $\langle a_k,\, v\rangle = \frac{\lambda_1}{n-k}$ because $\left\langle e_{j_i},\, v\right\rangle = v_{j_i} = 0$ for all $i \in \{1,\dots,h\}$. One further obtains that $\langle a_k,\, a_k\rangle = \|a_k\|_2^2 = \frac{1}{(n-k)^2}\left\|e - \sum_{i=1}^{k} e_{j_i}\right\|_2^2 + \left\|e_{j_k}\right\|_2^2 = 1 + \frac{1}{n-k}$. Therefore,

$$\begin{aligned}
\left\langle s^{(0)} - s^{(k)},\, s^{(k)} - v\right\rangle &= \left\langle s^{(0)} - s^{(k-1)} - s_{j_k}^{(k-1)} a_k,\, s^{(k-1)} + s_{j_k}^{(k-1)} a_k - v\right\rangle \\
&= \underbrace{\left\langle s^{(0)} - s^{(k-1)},\, s^{(k-1)} - v\right\rangle}_{=0 \Leftarrow \text{I.H.}} + s_{j_k}^{(k-1)}\left\langle a_k,\, s^{(0)} - 2s^{(k-1)} - s_{j_k}^{(k-1)} a_k + v\right\rangle \\
&= s_{j_k}^{(k-1)}\left(-\langle a_k,\, q\rangle - s_{j_k}^{(k-1)}\langle a_k,\, a_k\rangle + \langle a_k,\, v\rangle\right) \\
&= s_{j_k}^{(k-1)}\left(-\frac{\lambda_1}{n-k} + \frac{1}{n-k}\sum_{i=1}^{k} q_{j_i} + q_{j_k} - s_{j_k}^{(k-1)}\left(1 + \frac{1}{n-k}\right) + \frac{\lambda_1}{n-k}\right) \\
&= s_{j_k}^{(k-1)}\left(q_{j_k}\left(1 + \frac{1}{n-k}\right) + \frac{1}{n-k}\sum_{i=1}^{k-1} q_{j_i} - s_{j_k}^{(k-1)}\left(1 + \frac{1}{n-k}\right)\right) \overset{\text{(d)}}{=} 0,
\end{aligned}$$

where the last equality was yielded from (d) and $\left(1 + \frac{1}{n-k}\right) \cdot \frac{1}{n-k+1} = \frac{1}{n-k}$.

(f) Let $k \in \{0,\dots,h\}$ and $v \in C_I$. Then

$$\|q - v\|_2^2 \overset{\text{P. 1}}{=} \left\|q - s^{(k)}\right\|_2^2 + \left\|s^{(k)} - v\right\|_2^2 + \underbrace{2\left\langle q - s^{(k)},\, s^{(k)} - v\right\rangle}_{=0 \Leftarrow \text{(e)}},$$

thus the identity holds.

(g) C_I is convex with Lemma 24(b). Hence a unique Euclidean projection exists with Remark 4. $s \in C_I$ with (a). $\|q - s\|_2^2 \leq \|q - v\|_2^2$ for all $v \in C_I$ with (f). Therefore $s = \operatorname{proj}_{C_I}(q)$.

(h) In the remainder of the proof assume $\max_{j \in J} q_j \leq \min_{i \in I} q_i$. It is first shown by induction that $s_{j_1}^{(k)} \leq \dots \leq s_{j_h}^{(k)}$ for all $k \in \{0,\dots,h\}$. For $k = 0$ this is fulfilled as requirement on q. For

$k-1 \to k$, let $\mu, \nu \in \{j_1, \dots, j_h\}$ with $\mu < \nu$. Then with χ being the indicator function and with

$$A := \chi_{\{j_k\}}(\mu) - \chi_{\{j_k\}}(\nu) + \tfrac{1}{n-k}\left(\chi_{\{j_1,\dots,j_k\}^C}(\nu) - \chi_{\{j_1,\dots,j_k\}^C}(\mu)\right)$$

it is $s_\nu^{(k)} - s_\mu^{(k)} = s_\nu^{(k-1)} - s_\mu^{(k-1)} + s_{j_k}^{(k-1)} A$. Clearly, when $A \geq 0$ then the claim follows with the induction hypothesis and with $s_{j_k}^{(k-1)} \geq 0$ due to (a).

First consider the case of $\mu \in \{j_1, \dots, j_{k-1}\}$. If $\nu \in \{j_1, \dots, j_{k-1}\}$ also, then $A = 0$. If $\nu = j_k$, then $A = -1$, and hence $s_\nu^{(k)} - s_\mu^{(k)} = s_\nu^{(k-1)} - s_\mu^{(k-1)} - s_\nu^{(k-1)} = -s_\mu^{(k-1)}$ which however vanishes with (a). If $\nu \in \{j_{k+1}, \dots, j_h\}$, then $A = \frac{1}{n-k} \geq 0$. If $\mu = j_k$, then $\nu \in \{j_{k+1}, \dots, j_h\}$, and then $A = 1 + \frac{1}{n-k} \geq 0$. If $\mu \in \{j_{k+1}, \dots, j_h\}$, then $\nu \in \{j_{\mu+1}, \dots, j_h\}$, thus $A = 0$.

Next, it is shown that $\max_{j \in J} s_j^{(k)} \leq \min_{i \in I} s_i^{(k)}$ for all $k \in \{0, \dots, h\}$. For $k = 0$ this is the requirement on q. For $k-1 \to k$, let $i \in I$ and $j \in J$. It is then

$$\begin{aligned} s_i^{(k)} - s_j^{(k)} = {} & s_i^{(k-1)} - s_{j_k}^{(k-1)} \underbrace{\chi_{\{j_k\}}(i)}_{=0} + \tfrac{1}{n-k} s_{j_k}^{(k-1)} \underbrace{\chi_{\{j_1,\dots,j_k\}^C}(i)}_{=1} \\ & - s_j^{(k-1)} + s_{j_k}^{(k-1)} \chi_{\{j_k\}}(j) - \tfrac{1}{n-k} s_{j_k}^{(k-1)} \underbrace{\chi_{\{j_1,\dots,j_k\}^C}(j)}_{=0} \\ = {} & \underbrace{s_i^{(k-1)} - s_j^{(k-1)}}_{\geq 0 \Leftarrow \text{I.H.}} + \underbrace{s_{j_k}^{(k-1)} \left(\tfrac{1}{n-k} + \chi_{\{j_k\}}(j)\right)}_{\geq 0} \geq 0. \end{aligned}$$

(i) Using (a) and (h),

$$\begin{aligned} \lambda_1 \overset{\text{(a)}}{=} \sum_{i=1}^n s_i^{(k-1)} &= \sum_{i \in I} \underbrace{s_i^{(k-1)}}_{\geq s_{j_k}^{(k-1)}} + \sum_{i=1}^{k-1} \underbrace{s_{j_i}^{(k-1)}}_{=0} + s_{j_k}^{(k-1)} + \sum_{i=k+1}^{h} \underbrace{s_{j_i}^{(k-1)}}_{\geq s_{j_k}^{(k-1)}} \\ &\geq \left((n-h) + 1 + (h-k)\right) s_{j_k}^{(k-1)} = (n-k+1)\, s_{j_k}^{(k-1)}, \end{aligned}$$

and the claim follows because $n-k+1 > 0$.

(j) In (e) it was shown that $\|a_k\|_2^2 = 1 + \frac{1}{n-k}$. Furthermore,

$$\begin{aligned} \left\langle s^{(k-1)},\, a_k \right\rangle &= \tfrac{1}{n-k} \left\langle s^{(k-1)},\, e - \textstyle\sum_{i=1}^k e_{j_i} \right\rangle - \left\langle s^{(k-1)},\, e_{j_k} \right\rangle \\ &= \tfrac{1}{n-k} \left(\lambda_1 - \textstyle\sum_{i=1}^{k-1} s_{j_i}^{(k-1)} - s_{j_k}^{(k-1)}\right) - s_{j_k}^{(k-1)}, \end{aligned}$$

where $s_{j_i}^{(k-1)} = 0$ for $i \in \{1, \dots, k-1\}$ using (a). With

$$\|s^{(k)}\|_2^2 = \|s^{(k-1)}\|_2^2 + \left(s_{j_k}^{(k-1)}\right)^2 \|a_k\|_2^2 + 2 s_{j_k}^{(k-1)} \left\langle s^{(k-1)},\, a_k \right\rangle$$

follows

$$\begin{aligned}\|s^{(k)}\|_2^2 - \|s^{(k-1)}\|_2^2 &= s_{j_k}^{(k-1)}\left[s_{j_k}^{(k-1)}\left(1+\tfrac{1}{n-k}\right)+\tfrac{2}{n-k}\left(\lambda_1 - s_{j_k}^{(k-1)}\right)-2s_{j_k}^{(k-1)}\right]\\ &= s_{j_k}^{(k-1)}\left[\tfrac{2\lambda_1}{n-k} - s_{j_k}^{(k-1)}\left(1+\tfrac{1}{n-k}\right)\right],\end{aligned}$$

which is non-negative when $s_{j_k}^{(k-1)} \leq \left(1+\frac{1}{n-k}\right)^{-1} \cdot \frac{2\lambda_1}{n-k} = \frac{2\lambda_1}{n-k+1}$. With (i) this is always fulfilled. Therefore,

$$\|s\|_2 - \|q\|_2 = \|s^{(h)}\|_2 - \|s^{(0)}\|_2 = \sum_{k=1}^{h}\left(\|s^{(k)}\|_2 - \|s^{(k-1)}\|_2\right) \geq 0.$$

(k) When $q_j = 0$ for all $j \in J$, then $s = q$ and the claim follows. When there is a $j_k \in \{j_1, \dots, j_k\}$ with $q_{j_k} \neq 0$, let k be minimal such that either $k = 1$ or $q_{j_{k-1}} = 0$, hence $s_{j_k}^{(k-1)} = q_{j_k}$. With (i) follows $0 < s_{j_k}^{(k-1)} \leq \frac{\lambda_1}{n-k+1} < \frac{2\lambda_1}{n-k+1}$, and hence $\|s^{(k)}\|_2^2 - \|s^{(k-1)}\|_2^2 > 0$, thus $\|s\|_2 > \|q\|_2$. □

The application of Lemma 32 shows that the projection of a point from a face onto D must reside on the same face, given the original point is located within a sphere with squared radius ρ_I around m_I. As will be shown in Lemma 34, this is automatically fulfilled for projections from L onto C. A sketch for the second part of the proof of Corollary 33 is given in Figure 4.7.

Corollary 33. Let $I \subseteq \{1, \dots, n\}$, let $v \in C_I$ with $\|v\|_2 < \lambda_2$, and let $q \in \text{proj}_D(v)$. Then $q \in C_I$.

Proof. Let $J := \{1, \dots, n\} \setminus I$, and let $q \in \text{proj}_D(v)$. Assume there is at least one $j \in J$ with $q_j \neq 0$. For showing $\max_{j \in J} q_j \leq \min_{i \in I} q_i$, assume there are $i \in I$ and $j \in J$ with $q_j > q_i$. Then $v_i > 0$ and $v_j = 0$ because of $v \in C_I$. D is permutation-invariant using Remark 7 as intersection of permutation-invariant sets. Hence, analogous to Lemma 9(c), let $\tau := (i, j)$ be the transposition swapping i and j, and consider:

$$d(\tau) = \|q - v\|_2^2 - \|P_\tau q - v\|_2^2 = 2\underbrace{(q_j - q_i)}_{>0}\underbrace{(v_i - v_j)}_{=v_i>0} > 0.$$

Hence $\|P_\tau q - v\|_2 < \|q - v\|_2$ and $P_\tau q \in D$, which violates the minimality of q. Therefore, $\max_{j \in J} q_j \leq \min_{i \in I} q_i$ must hold.

Thus using Lemma 32 there is an $s \in C_I$ with $\|q - v\|_2^2 = \|q - s\|_2^2 + \|s - v\|_2^2$, and $\|s\|_2 > \lambda_2$ with Lemma 32(k). Consider $f\colon [0,\ 1] \to \mathbb{R}$, $\beta \mapsto \|v + \beta(s - v)\|_2$. Clearly, $f(0) = \|v\|_2 < \lambda_2$ and $f(1) = \|s\|_2 > \lambda_2$, hence with the intermediate value theorem there exists a $\beta^* \in (0,\ 1)$ with $f(\beta^*) = \lambda_2$. Let $t := v + \beta^*(s - v) \in \mathbb{R}^n$. Because of $v, s \in C_I$, C_I convex and $0 < \beta^* < 1$, $t \in C_I$ as well. By construction $\|t\|_2 = \lambda_2$, hence $t \in \tilde{D} := \{a \in D \mid a_i = 0 \text{ for all } i \notin I\}$. Clearly, $\|v - t\|_2 + \|t - s\|_2 = |\beta^*| \cdot \|s - v\|_2 + |1 - \beta^*| \cdot \|v - s\|_2 = \|s - v\|_2$. Let $p \in \text{proj}_{\tilde{D}}(v)$, then $\|v - p\|_2 \leq \|v - t\|_2$. Therefore,

$$\|q - v\|_2^2 = \|q - s\|_2^2 + \|s - v\|_2^2 = \|q - s\|_2^2 + (\|v - t\|_2 + \|t - s\|_2)^2 > \|v - t\|_2^2 \geq \|v - p\|_2^2.$$

Because of $\tilde{D} \subseteq D$, $p \in D$ also, hence $\|q - v\|_2 \leq \|p - v\|_2$, which contradicts $\|q - v\|_2 > \|v - p\|_2$. Hence, $q_j = 0$ for all $j \in J$, and thus $q \in C_I$. □

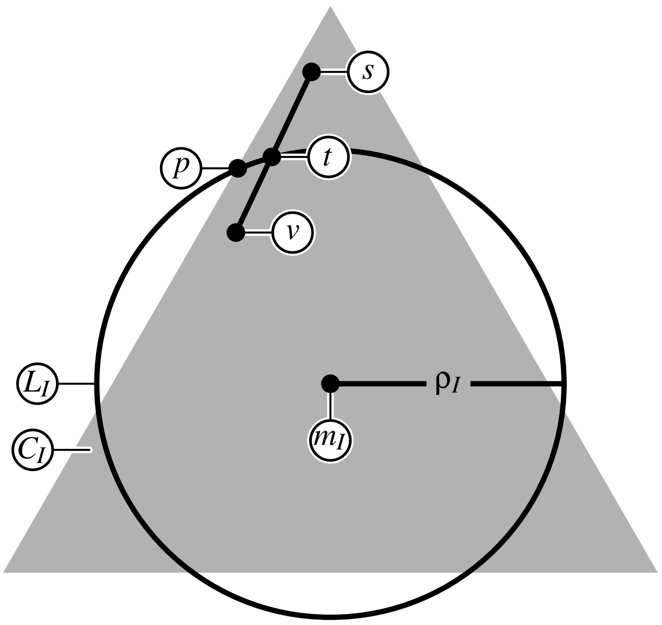


Figure 4.7: Sketch of the proof of Corollary 33: The projection of v onto D must be located on face C_I. Assume there is a projection q which is not on C_I, then it must be sufficiently close to C_I for application of Lemma 32. The projection s of q onto C_I is located outside $L_I := \{\, a \in L \mid a_i = 0 \text{ for all } i \notin I \,\}$. Hence the intersection t of the line between v and s and L_I is in C_I due to its convexity, and it is farther than the projection p of v onto $\tilde{D} := \{\, a \in D \mid a_i = 0 \text{ for all } i \notin I \,\}$. Adapted from [66].

Because C_I is isomorphic to a simplex, but with lower dimensionality than C, an algorithm with tail-recursion can be yielded to compute the projection onto D, see the next section.

4.3.7 Self-Similarity of the Feasible Set and Recursion Step

The next Lemma summarizes previous results and analyzes projections from L onto C in greater detail. It shows that the solution set with respect to the projection onto D is not tampered, and that all solutions have zeros at the same positions as the projection onto C. Figure 4.8 provides orientation on the simplex projection as discussed in Lemma 34.

Lemma 34. Let $s \in L \setminus C$ and $r \in \operatorname{proj}_C(s)$. Let $I := \{\, i \in \{1, \dots, n\} \mid r_i \neq 0 \,\}$ and $d := |I|$. Then the following holds:

(a) It is $r = \lambda_1 \operatorname{proj}_{\triangle^n}({}^{s}/_{\lambda_1})$, and there is $\hat{t} \in \mathbb{R}_{\geq 0}$ such that $r = \lambda_1 \max({}^{s}/_{\lambda_1} - \hat{t} \cdot e,\ 0)$, where the maximum is taken element-wise.

(b) $I \neq \varnothing$, $I \neq \{1, \dots, n\}$ and $\lambda_1 = \sum_{i \in I} h_i + \sum_{i \notin I} h_i$ for all $h \in H$.

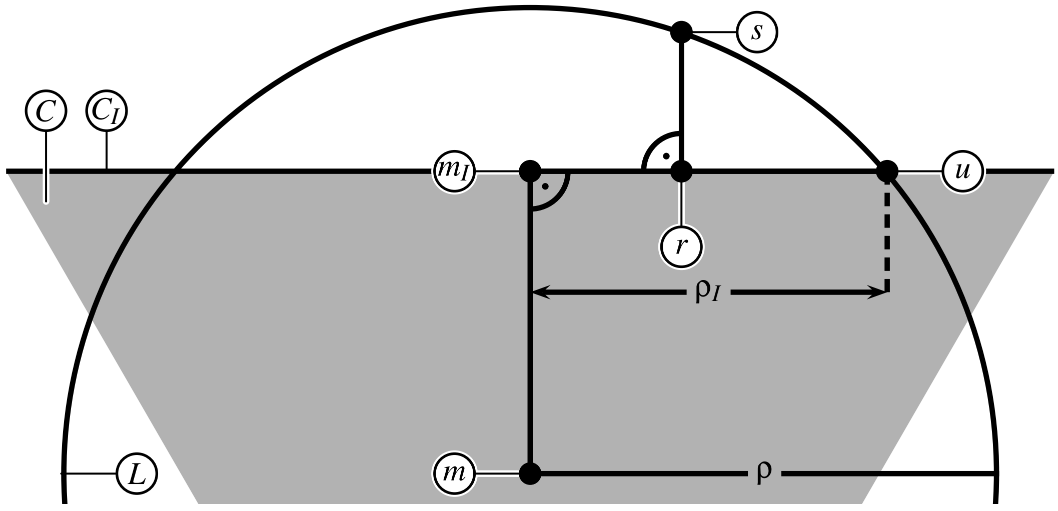


Figure 4.8: Situation of Lemma 34 and Lemma 36: s is projected onto simplex C yielding r, which resides on one of its faces. From there point u can be constructed by projecting within C_I onto the target hypersphere. Adapted from [66].

(c) When $i \in I$, then $s_i > \lambda_1 \hat{t}$ and $s_i - r_i = \lambda_1 \hat{t}$. When $i \notin I$, then $s_i \leq \lambda_1 \hat{t}$ and $s_i - r_i = s_i$.

(d) $\langle r,\ s - r \rangle = \lambda_1^2 \hat{t}$.

(e) $m_I \in C_I$ and $\langle m_I,\ s - r \rangle = \lambda_1^2 \hat{t}$, thus $\langle m_I - r,\ s - r \rangle = 0$.

(f) $\operatorname{proj}_D(r) \subseteq C_I$, that is from $q \in \operatorname{proj}_D(r)$ follows $q_i = 0$ for all $i \notin I$.

(g) Let $q \in \operatorname{proj}_D(r)$. Then $\langle q - r,\ s - r \rangle = 0$, and thus $\|q - s\|_2^2 = \|q - r\|_2^2 + \|r - s\|_2^2$.

(h) $\operatorname{proj}_D(r) = \operatorname{proj}_D(s) \subseteq C_I$.

Proof. (a) Let $\tilde{s} := {}^{s}/_{\lambda_1}$. Then $\tilde{s} \notin \triangle^n$ with Lemma 24(a). Let $\tilde{r} := \operatorname{proj}_{\triangle^n}(\tilde{s})$, then with Lemma 28(a) there exists $\hat{t} \in \mathbb{R}$ such that $\tilde{r} = \max(\tilde{s} - \hat{t} \cdot e,\ 0)$. With Lemma 24(a), $r = \lambda_1 \tilde{r}$, and the claim follows.

For showing $\hat{t} \geq 0$, first consider $J := \{\, j \in \{1, \ldots, n\} \mid s_j \geq 0 \,\}$. Because of $s \in L \setminus C$, it is $e^T s = \lambda_1$ and there is an index i with $s_i < 0$, hence $J \neq \{1, \ldots, n\}$. Therefore, $\lambda_1 = \sum_{j \in J} s_j + \sum_{j \notin J} s_j < \sum_{j \in J} s_j$. Now assume $\hat{t} < 0$, then ${}^{s_j}/_{\lambda_1} - \hat{t} > 0$ for all $j \in J$, and hence $r_j = s_j - \lambda_1 \hat{t}$ for all $j \in J$. This yields

$$\lambda_1 \overset{r \in H}{=} \sum_{j=1}^{n} r_j \overset{r_j \geq 0}{\geq} \sum_{j \in J} r_j = \sum_{j \in J} \left(s_j - \lambda_1 \hat{t} \right) = \sum_{j \in J} s_j - \underbrace{|J| \cdot \lambda_1 \hat{t}}_{<0} > \sum_{j \in J} s_j,$$

which contradicts $\sum_{j \in J} s_j > \lambda_1$. Hence $\hat{t} \geq 0$.

(b) Assume $I = \varnothing$, then $r = 0$ which contradicts $r \in C$. With Remark 30, $r \in \partial C$, and with Lemma 24(c) there is a $j \in \{1,\dots,n\}$ with $r_j = 0$, hence $I \neq \{1,\dots,n\}$. The summation formula is trivial, because $\{1,\dots,n\} = I \cup (\{1,\dots,n\} \setminus I)$.

(c) Using (a): When $i \in I$, then $0 < r_i = \lambda_1 \left({}^{s_i}/_{\lambda_1} - \hat{t}\right) = s_i - \lambda_1 \hat{t}$ and the claim follows. When $i \notin I$, then $r_i = 0$, hence ${}^{s_i}/_{\lambda_1} - \hat{t} \le 0$ and the claim follows.

(d) Because $r, s \in H$ it is $e^T r = e^T s = \lambda_1$, hence

$$0 = e^T(s-r) = \sum\nolimits_{i \in I}(s_i - r_i) + \sum\nolimits_{i \notin I}(s_i - r_i) \overset{\text{(c)}}{=} \sum\nolimits_{i \in I} \lambda_1 \hat{t} + \sum\nolimits_{i \notin I} s_i \overset{\text{(b)}}{=} d\lambda_1 \hat{t} + \lambda_1 - \sum\nolimits_{i \in I} s_i,$$

thus

$$\langle r,\ s-r \rangle = \sum\nolimits_{i \in I} r_i(s_i - r_i) \overset{\text{(c)}}{=} \sum\nolimits_{i \in I}(s_i - \lambda_1 \hat{t})\lambda_1 \hat{t} = \lambda_1 \hat{t}\left(\sum\nolimits_{i \in I} s_i - d\lambda_1 \hat{t}\right) = \lambda_1^2 \hat{t}.$$

(e) Clearly, $m_I \in C_I$. With (c) follows $\langle m_I,\ s-r \rangle = \sum_{i \in I} {}^{\lambda_1}/_d \cdot (s_i - r_i) \overset{\text{(c)}}{=} {}^{\lambda_1}/_d \sum_{i \in I} \lambda_1 \hat{t} = \lambda_1^2 \hat{t}$, and the claim follows with (d).

(f) Clearly $r \in C_I$ and

$$\begin{aligned}\lambda_2^2 = \|s\|_2^2 &= \sum\nolimits_{i \in I} s_i^2 + \sum\nolimits_{i \notin I} s_i^2 \overset{\text{(c)}}{=} \sum\nolimits_{i \in I}(r_i + \lambda_1 \hat{t})^2 + \sum\nolimits_{i \notin I} s_i^2 \\ &> \sum\nolimits_{i \in I} r_i^2 + d\lambda_1^2 \hat{t}^2 + 2\lambda_1 \hat{t} \sum\nolimits_{i \in I} r_i \overset{r \in H}{=} \|r\|_2^2 + \lambda_1^2 \hat{t}(d\hat{t} + 2) \overset{\text{(a)}}{\ge} \|r\|_2^2,\end{aligned}$$

thus the claim holds using Corollary 33.

(g) It is $\langle q,\ s-r \rangle \overset{\text{(f)}}{=} \sum_{i \in I} q_i(s_i - r_i) \overset{\text{(c)}}{=} \lambda_1 \hat{t} \sum_{i \in I} q_i \overset{\text{(f)}}{=} \lambda_1^2 \hat{t}$, and the claims follow with (d) and Proposition 1.

(h) Let $p \in \operatorname{proj}_D(s)$ and $q \in \operatorname{proj}_D(r)$. Using (g) and Proposition 1, it is $\|q-s\|_2^2 - \|p-s\|_2^2 = \|q-r\|_2^2 - \|p-r\|_2^2 + 2\langle p-r,\ s-r \rangle$. Furthermore,

$$\begin{aligned}\langle p,\ s-r \rangle &= \sum\nolimits_{i \in I} p_i(s_i - r_i) + \sum\nolimits_{i \notin I} p_i(s_i - r_i) \overset{\text{(c)}}{=} \sum\nolimits_{i \in I} p_i \lambda_1 \hat{t} + \sum\nolimits_{i \notin I} p_i s_i \\ &\overset{\text{(b)}}{=} \lambda_1 \hat{t}\left(\lambda_1 - \sum\nolimits_{i \notin I} p_i\right) + \sum\nolimits_{i \notin I} p_i s_i,\end{aligned}$$

and hence

$$\langle p-r,\ s-r \rangle \overset{\text{(d)}}{=} \lambda_1 \hat{t}\left(\lambda_1 - \sum\nolimits_{i \notin I} p_i\right) + \sum\nolimits_{i \notin I} p_i s_i - \lambda_1^2 \hat{t} = \sum\nolimits_{i \notin I} \underbrace{p_i}_{\ge 0} \underbrace{(s_i - \lambda_1 \hat{t})}_{\le 0 \Leftarrow \text{(c)}} \le 0.$$

$q \in \operatorname{proj}_D(r)$ yields $\|q-r\|_2^2 \le \|p-r\|_2^2$, and hence $\|q-s\|_2^2 - \|p-s\|_2^2 \le 0$, thus $q \in \operatorname{proj}_D(s)$. On the other hand, $p \in \operatorname{proj}_D(s)$ yields $\|q-s\|_2^2 - \|p-s\|_2 \ge 0$, hence $\|q-r\|_2^2 - \|p-r\|_2^2 \ge -2\langle p-r,\ s-r \rangle \ge 0$, thus $p \in \operatorname{proj}_D(r)$. □

The following corollary states a similar result as in [63]. However, the proof here uses the notion of simplex projections instead of relying on pure analytical statements. The result here is stronger, as multiple entries of the vector can be set to zero simultaneously, while in [63] at most one entry can be zeroed out at a single iteration.

Corollary 35. Let $s \in L \setminus C$, $p \in \operatorname{proj}_D(s)$ and $i \in \{1, \dots, n\}$. When $s_i \leq 0$ then $p_i = 0$.

Proof. Let $i \in \{1, \dots, n\}$ with $s_i \leq 0$. Let $r \in \operatorname{proj}_C(s)$. With Lemma 34(a) follows $r_i = 0$ because $\hat{t} \geq 0$, and the claim follows with Lemma 34(h). □

The final part is now to meet the hypersphere constraint again. For this, the simplex projection r is projected onto the target hypersphere, simultaneously keeping already vanished entries at zero, yielding point u. Lemma 36 gives an explicit formulation of this projection and shows that the solution set with respect to π stays the same. Refer to Figure 4.8 for a sketch of the construction of u.

Lemma 36. Let $s \in L \setminus C$, $r := \lambda_1 \max\left({}^{s}/_{\lambda_1} - \hat{t} \cdot e,\ 0\right) = \operatorname{proj}_C(s)$ where $\hat{t} \in \mathbb{R}_{\geq 0}$ by application of Lemma 34, $I := \{\, i \in \{1, \dots, n\} \mid r_i \neq 0 \,\}$ and $d := |I|$. Further, let $u := m_I + \delta\,(r - m_I)$ where $\delta := \sqrt{\rho_I} / \lVert r - m_I \rVert_2$. Then the following holds:

(a) $u \in L$ and $u_i = 0$ for all $i \notin I$.

(b) $\operatorname{proj}_D(u) \subseteq C_I$.

(c) Let $L_I := \{\, a \in L \mid a_i = 0 \text{ for all } i \notin I \,\}$, then $u = \operatorname{proj}_{L_I}(r)$.

(d) $\operatorname{proj}_D(u) = \operatorname{proj}_D(r) = \operatorname{proj}_D(s) \subseteq C_I$.

Proof. (a) Clearly $u \in H$. With $m_I, r \in H$ and Remark 17, $\langle m,\ m_I \rangle = \langle m,\ r \rangle = \lambda_1^2/n$. Furthermore, $\langle m_I,\ m_I \rangle = \lambda_1^2/d$ and $\langle r,\ m_I \rangle = \sum_{i \in I} r_i \cdot \lambda_1/d = \lambda_1^2/d$, therefore $\langle r,\ m_I - m \rangle = \langle m_I,\ m_I - m \rangle$. With $u = (1 - \delta)\, m_I + \delta r$ it is $\langle u,\ m_I - m \rangle = \langle m_I,\ m_I - m \rangle = \lambda_1^2\,(1/d - 1/n)$. Therefore it is $\langle u - m_I,\ m_I - m \rangle = 0$. Furthermore, $\lVert m - m_I \rVert_2^2 = \lVert m \rVert_2^2 + \lVert m_I \rVert_2^2 - 2\,\langle m,\ m_I \rangle = \lambda_1^2\,(1/d - 1/n)$.

Thus with Proposition 1, $\lVert u - m \rVert_2^2 = \lVert u - m_I \rVert_2^2 + \lVert m_I - m \rVert_2^2 = \rho_I + \lambda_1^2\,(1/d - 1/n) = \rho$, and with Lemma 18 follows $u \in L$. For $i \notin I$, it is $u_i = (1 - \delta)\, e_i^T m_I + \delta r_i = 0$.

(b) If $u \in C$, then $u \in D$ because of $u \in L$ with (a), and hence $u = \operatorname{proj}_D(u)$. With $u_i = 0$ for all $i \notin I$ the claim follows.

If $u \notin C$, then let $q \in \operatorname{proj}_D(u)$. With Corollary 35 applied to u follows that $q_i = 0$ for all i with $u_i \leq 0$, especially for all $i \in I$. Hence the claim follows.

(c) Write $I = \{i_1, \dots, i_d\}$ and consider $\varphi\colon \mathbb{R}^n \to \mathbb{R}^d$, $(x_1,\ \dots,\ x_n)^T \mapsto (x_{i_1},\ \dots,\ x_{i_d})^T$. Furthermore, let $\tilde{H} := \{\, a \in \mathbb{R}^d \mid e^T a = \lambda_1 \,\}$, $\tilde{K} := \{\, q \in \mathbb{R}^d \mid \lVert q \rVert_2 = \lambda_2 \,\}$, $\tilde{L} := \tilde{H} \cap \tilde{K}$ and $\tilde{D} := \mathbb{R}^d_{\geq 0} \cap \tilde{H} \cap \tilde{K}$. Clearly, when $x_i = 0$ for all $i \notin I$, then $e^T x = e^T \varphi(x)$ and $\lVert x \rVert_2 = \lVert \varphi(x) \rVert_2$. Thus in this case membership of x in one of H, K, L or D implies membership of $\varphi(x)$ in $\tilde{H}$, $\tilde{K}$, $\tilde{L}$ or $\tilde{D}$, respectively.

Application of Lemma 19 to $\varphi(r)$ and $\varphi(u)$ states that $\varphi(u) = \text{proj}_{\tilde{L}}(\varphi(r))$. Let $q \in L_I$, then $\varphi(q) \in \tilde{L}$, hence $\|\varphi(u) - \varphi(r)\|_2 \leq \|\varphi(q) - \varphi(r)\|_2$. From $i \notin I$ follows $r_i = u_i = q_i = 0$, hence $\|u - r\|_2 = \|\varphi(u) - \varphi(r)\|_2$ and $\|q - r\|_2 = \|\varphi(q) - \varphi(r)\|_2$, and the claim follows.

(d) For the converse of φ, let $\psi\colon \mathbb{R}^d \to \mathbb{R}^n$, $\tilde{x} \mapsto x$ where $x_i = 0$ for all $i \notin I$ and $x_i = \tilde{x}_j$ when there is a $j \in \{1, \ldots, d\}$ with $i = i_j$. Analogous to above, membership of $\tilde{y}$ in one of $\tilde{H}$, $\tilde{K}$, $\tilde{L}$ or $\tilde{D}$ implies membership of $\psi(\tilde{y})$ in H, K, L or D, respectively. It is $x = \psi(\varphi(x))$ if and only if $x_i = 0$ for $i \notin I$.

With Lemma 34(f) and Lemma 34(h) it is sufficient to show $\text{proj}_D(u) = \text{proj}_D(r)$. Analogous to (c), from Lemma 19 follows also that $\text{proj}_{\tilde{D}}(\varphi(r)) = \text{proj}_{\tilde{D}}(\varphi(u))$. Let $p \in \text{proj}_D(u)$ and $q \in \text{proj}_D(r)$, then $p \in C_I$ with (b) and $q \in C_I$ with Lemma 34(f), and thus $\varphi(p), \varphi(q) \in \tilde{D}$. Assume $\varphi(p) \notin \text{proj}_{\tilde{D}}(\varphi(u))$, then there exists an $a \in \tilde{D}$ with $\|\psi(a) - u\|_2 = \|a - \varphi(u)\|_2 < \|\varphi(p) - \varphi(u)\|_2 = \|p - u\|_2$, violating the minimality of p. Hence $\varphi(p) \in \text{proj}_{\tilde{D}}(\varphi(u))$, and analogously $\varphi(q) \in \text{proj}_{\tilde{D}}(\varphi(r))$. Because $\text{proj}_{\tilde{D}}(\varphi(r)) = \text{proj}_{\tilde{D}}(\varphi(u))$ it is $\varphi(p) \in \text{proj}_{\tilde{D}}(\varphi(r))$ and $\varphi(q) \in \text{proj}_{\tilde{D}}(\varphi(u))$. Hence, $\|p - r\|_2 = \|\varphi(p) - \varphi(r)\|_2 = \|\varphi(q) - \varphi(r)\|_2 = \|q - r\|_2$, thus $p \in \text{proj}_D(r)$, and analogously $q \in \text{proj}_D(u)$. Therefore $\text{proj}_D(u) = \text{proj}_D(r)$. □

With Lemma 36 a point u is constructed. If $u \in C$, then u is the solution of π. Otherwise, Lemma 34 and Lemma 36 can be applied once more, gaining a new point u. Lemma 29 states that the amount of nonzero entries of u must decrease, hence this recursive process can be repeated for at most n iterations. If a point with only two non-vanishing entries results, it is guaranteed to be a solution with Corollary 27.

4.4 Algorithms

Theorem 37 uses the previous results to show that the proposed Algorithm 4.2 computes a correct solution, and that the algorithm always terminates in finite time.

Theorem 37. For every $x \in \mathbb{R}^n$, Algorithm 4.2 computes an element from $\text{proj}_D(x)$. If $r \neq m$ after Line 1 and $r \neq m_I$ after Line 4 in all iterations, then $\text{proj}_D(x)$ is a singleton.

Proof. For proving partial correctness, let $x \in \mathbb{R}^n$ be arbitrary. Lemma 16 yields $\text{proj}_D(x) = \text{proj}_D(r)$ after Line 1. With Lemma 19 follows $\text{proj}_D(x) = \text{proj}_D(s)$ after Line 2. In Line 3 there is a pre-test loop with loop-invariant $\text{proj}_D(x) = \text{proj}_D(s)$. At the beginning of the loop, $s \notin \mathbb{R}^n_{\geq 0}$ must hold, thus $s \in L \setminus C$. After Line 4, $\text{proj}_D(x) = \text{proj}_D(r)$ holds with Lemma 34. Then with Lemma 36, $\text{proj}_D(x) = \text{proj}_D(s)$ is ensured after Line 5, hence the loop-invariant holds. Thus, after the loop $\text{proj}_D(x) = \text{proj}_D(s)$ and $s \in D$ holds, so $\text{proj}_D(x) = s$. If $r = m$ in Line 2 or $r = m_I$ in Line 5, s can be chosen to be any point from L or L_I, respectively.

Algorithm 4.2: Computation of $\operatorname{proj}_D(x)$ for $x \in \mathbb{R}^n$.

Input: $x \in \mathbb{R}^n$ and $\lambda_1, \lambda_2 \in \mathbb{R}_{>0}$ with $\lambda_2 \leq \lambda_1 \leq \sqrt{n}\lambda_2$.
Output: $s \in \operatorname{proj}_D(x)$.

```
1 $r := \operatorname{proj}_H(x)$;
2 $s \in \operatorname{proj}_L(r)$;
3 while $s \notin \mathbb{R}^n_{\geq 0}$ do
4     $r := \operatorname{proj}_C(s)$;
5     $s \in \operatorname{proj}_{L_I}(r)$ where $I := \{\, i \in \{1, \dots, n\} \mid r_i \neq 0 \,\}$;
6 end
```

Refer to Remark 20 for details. In this case, the Euclidean projection is not unique, but a valid representative is found.

For proving total correctness it has to be shown that the loop in Line 3 terminates. Lemma 29 applied to C_I states that the number of nonzero entries in s is strictly less at the end of the loop then the number of nonzero entries upon entering the loop. Hence at most n iterations of the loop are carried out, and when $|I| = 2$ the solution is already in D with Corollary 27. Thus the algorithm terminates in finite time. □

This algorithm for the projection onto D can be improved by exploiting the general properties of projections onto symmetric sets. Algorithm 4.3 contains an elaborate description of the steps performed during the projection onto D of this optimized variant.

Theorem 38. Algorithm 4.2 is equivalent to Algorithm 4.3.

Proof. First note that Algorithm 4.3 consists of two procedures, `proj_H` and `proj_L`, carrying out the projections onto H and L in-place, a function `proj_C` returning the information on how to perform the projection onto C, and a main body. Upon entry of the main body, the input vector x is sorted, yielding y. The algorithm then works on the sorted vector y, and undoes the sorting permutation at the end. Because H, L and C are permutation-invariant, the projections onto the respective sets are guaranteed to remain sorted with Lemma 10 and Lemma 11.

Therefore, y has not to be sorted again for the simplex projection, as Algorithm 4.1 would require. Also note from Lemma 34 that in the simplex projection the smallest elements are set to zero, and the original Algorithm 4.2 continues working on the d non-vanishing entries. Because of the order-preservation, entries $d+1, \dots, n$ of y are zero, and all the information is concentrated in $y_1, \dots, y_d$. Therefore, Algorithm 4.3 can continue working on these first d entries only. As the nonzero elements are stored contiguously in memory, access to y can be realized as small unit-stride array. This is more efficient than working on a large, sparsely populated vector.

Algorithm 4.3: Computation of $\mathrm{proj}_D(x)$ for $x \in \mathbb{R}^n$, explicit variant.

```
Input: $x \in \mathbb{R}^n$ and $\lambda_1, \lambda_2 \in \mathbb{R}_{>0}$ with $\lambda_2 \leq \lambda_1 \leq \sqrt{n}\lambda_2$.
Output: $s \in \mathrm{proj}_D(x)$.
1  procedure proj_H($y \in \mathbb{R}^d$)
2      $s := \sum_{i=1}^{d} y_i$;  $(y_1, \ldots, y_d)^T := (y_1, \ldots, y_d)^T + 1/d \cdot (\lambda_1 - s)$;
3  end
4  procedure proj_L($y \in \mathbb{R}^d$)
5      $\rho := \lambda_2^2 - \lambda_1^2/d$;  $\varphi := \sum_{i=1}^{d} y_i^2 - \lambda_1^2/d$;
6      if $\varphi = 0$ then
7          $(y_1, \ldots, y_{d-1})^T := \lambda_1/d + \sqrt{\rho}/\sqrt{d(d-1)}$;
8          $y_d := \lambda_1/d - \sqrt{\rho(d-1)}/\sqrt{d}$;
9      else $(y_1, \ldots, y_d)^T := \lambda_1/d + \sqrt{\rho/\varphi} \cdot \left((y_1, \ldots, y_d)^T - \lambda_1/d\right)$;
10 end
11 function proj_C($y \in \mathbb{R}^d$) $\in \mathbb{R} \times \mathbb{N}$
12     $s := 0$;
13     for $i := 1$ to $d - 1$ do
14         $s := s + y_i$; $t := \frac{s-\lambda_1}{i}$;
15         if $t \geq y_{i+1}$ then return $(t, i)$;
16     end
17     $s := s + y_d$;  $t := \frac{s-\lambda_1}{d}$;  return $(t, d)$;
18 end
   // Sort the input vector
19 Let $\tau$ be a permutation of $\{1, \ldots, n\}$ such that $x_{\tau(1)} \geq \cdots \geq x_{\tau(n)}$;
20 Let $y \in \mathbb{R}^n$;  for $i := 1$ to $n$ do $y_i := x_{\tau(i)}$;
   // Project onto hyperplane and hypersphere
21 proj_H($y$);
22 proj_L($y$);
   // Project onto simplex and hypersphere until feasible solution is found
23 $d := n$;
24 while $(y_1, \ldots, y_d)^T \notin \mathbb{R}^d_{\geq 0}$ do
25     $(\hat{t}, d) :=$ proj_C$\left((y_1, \ldots, y_d)^T\right)$;
26     $(y_1, \ldots, y_d)^T := (y_1, \ldots, y_d)^T - \hat{t}$;
27     proj_L$\left((y_1, \ldots, y_d)^T\right)$;
28 end
   // Undo sorting and set all remaining entries to zero
29 $s \in \{0\}^n$;  for $i := 1$ to $d$ do $s_{\tau(i)} := y_i$;
```

Algorithm 4.4: Computation of $\pi(x, \sigma^*, \omega, \eta)$.

Input: $x \in \mathbb{R}^n$, $\sigma^* \in [0,\ 1]$, $\omega \in \{\,\text{non-negative, arbitrary}\,\}$, $\eta \in \{\,\text{unit\_norm, keep\_norm}\,\}$.
Output: $s \in \mathbb{R}^n$ with $\sigma(s) = \sigma^*$ and $s = \arg\min_{s \in S} \|x - s\|_2$ for S as determined by Table 4.1.

```
  // Remember signs if non-negativity is not required
1 if $\omega$ = arbitrary then $I := \{\, i \in \{1,\dots,n\} \mid x_i < 0 \,\}$; $x := |x|$;
  // Determine norm constraints for target sparseness
2 $\ell := \sqrt{n} - \sigma^* \cdot (\sqrt{n} - 1)$;
3 if $\eta$ = unit_norm then $\lambda_1 := \ell$; $\lambda_2 := 1$; else $\lambda_1 := \ell \cdot \|x\|_2$; $\lambda_2 := \|x\|_2$;
  // Carry out projection onto $D$
4 $s \in \text{proj}_D(x)$;
  // Restore signs if non-negativity is not required
5 if $\omega$ = arbitrary then for $i \in I$ do $s_i := -s_i$;
```

Procedure `proj_H` is a direct application of Lemma 16, and procedure `proj_L` is a direct application of Lemma 19 and Remark 20. Therefore, Lines 21 and 22 correspond to Lines 1 and 2 of Algorithm 4.2.

Function `proj_C` is an improved version of Algorithm 4.1. Sorting of the input vector y can be omitted. With Lemma 34(a), the scaling of the canonical simplex by λ_1 can be incorporated into Algorithm 4.1, yielding expressions $t := {}^{(s-\lambda_1)}/_{i}$ and $t := {}^{(s-\lambda_1)}/_{d}$, in Lines 14 and 17 respectively. The index i or d from the stopping condition can be used to yield the number of nonzero entries in the resulting vector. This follows again from the vector being sorted, and because of the result of the projection onto C is $\max(y - \hat{t} \cdot e,\ 0)$ with Lemma 28(a). Hence the loop starting at Line 24 corresponds to the loop starting at Line 3 in Algorithm 4.2.

At the end of the main body, the sorting permutation τ is inverted and the entries from the sorted result vector y are stored back into x. Because $y_{d+1},\ \dots,\ y_n = 0$, these entries can be ignored in the back-substitution by setting the entire vector x to zero before-hand. □

The wrapper for calling Algorithm 4.3 is given in Algorithm 4.4. It computes the target L_1 norm based on the target sparseness and the target L_2 norm, which is restricted to be unity or the L_2 norm of the input vector, see Table 4.1. With Lemma 12, Algorithm 4.4 is suitable for both non-negative and unrestricted solutions, where the latter is handled via Lines 1 and 5.

4.5 Analytical Properties

This section studies analytical properties of the sparseness-enforcing projection operator, especially in which cases it is continuous and differentiable. An application of Rademacher's theorem shows that projections onto closed and convex sets are differentiable almost everywhere [73]. But because the target set of the sparseness projection π is non-convex, a more detailed analysis has to be carried out to facilitate statements on the analytical nature of π.

As an introductory example, consider once more projections onto an L_0 pseudo-norm constraint as in Section 4.2. Let $Z := \{x \in \mathbb{R}^n \mid \|x\|_0 = \kappa\}$ where $\kappa \in \mathbb{N}$ is a constant. The projection onto Z consists simply of zeroing out the elements that are smallest in absolute value. Let $x \in \mathbb{R}^n$ be a point and let $\tau \in S_n$ be a permutation such that $\left|x_{\tau(1)}\right| \geq \cdots \geq \left|x_{\tau(n)}\right|$. Clearly, if $\left|x_{\tau(\kappa)}\right| \neq \left|x_{\tau(\kappa+1)}\right|$ then $\operatorname{proj}_Z(x) = y$ where $y_i = x_i$ for $i \in \{\tau(1), \ldots, \tau(\kappa)\}$ and $y_i = 0$ for $i \in \{\tau(\kappa+1), \ldots, \tau(n)\}$. Furthermore, when $\left|x_{\tau(\kappa)}\right| \neq \left|x_{\tau(\kappa+1)}\right|$ then there exists a neighborhood U of x such that $\operatorname{proj}_Z(s) = \sum_{i=1}^{\kappa} s_{\tau(i)} e_{\tau(i)}$ for all $s \in U$. Because of this closed-form expression, $s \mapsto \operatorname{proj}_Z(s)$ is differentiable in x with gradient ${}^{\partial \operatorname{proj}_Z(x)}/_{\partial x} = \operatorname{diag}\left(\sum_{i=1}^{\kappa} e_{\tau(i)}\right)$, that is the identity matrix where the entries on the diagonal belonging to small absolute values of x have been zeroed out. If the requirement on x is not fulfilled, then a small distortion of x is sufficient to find a point in which the projection onto Z is differentiable.

In contrast to the L_0 projection, differentiability of the sparseness projection π is non-trivial. The following observation is the starting point of this analysis:

Proposition 39. Let $x \in \mathbb{R}^n$ and $p \in \operatorname{proj}_D(x)$. Then there is a finite sequence of index sets $I_1, \ldots, I_h \subseteq \{1, \ldots, n\}$ with $I_j \supsetneq I_{j+1}$ for $j \in \{1, \ldots, h-1\}$ such that

$$p = \left[\bigcirc_1^{j=h}\left(\operatorname{proj}_{L_{I_j}} \circ \operatorname{proj}_C\right) \circ \operatorname{proj}_L \circ \operatorname{proj}_H\right](x),$$

where $\circ_1^{j=h}$ denotes iterated composition of functions, starting with $j = h$ and going down until $j = 1$, that is $p = \operatorname{proj}_{L_{I_h}}(\operatorname{proj}_C(\cdots \operatorname{proj}_{L_{I_1}}(\operatorname{proj}_C(\operatorname{proj}_L(\operatorname{proj}_H(x)))) \cdots))$.

Proof. Follows directly from Theorem 37. □

Hence for any point x there is a composition of functions such that $p = \operatorname{proj}_D(x)$ can be computed. It remains to be shown that small changes in x do not change the structure of this function, that is that the sequence $I_1, \ldots, I_h$ is not changed.

Lemma 40. Let $x \in \mathbb{R}^n \setminus C$ and $p := \operatorname{proj}_C(x)$. Then the following holds:

(a) There is a $\hat{t} \in \mathbb{R}$ such that $p = \max(x - \hat{t} \cdot e,\ 0)$.

(b) There is an $I \subseteq \{1, \ldots, n\}$ such that $\hat{t} = {}^1/_d \cdot \left(\sum_{i \in I} x_i - \lambda_1\right)$, where $d := |I|$.

For the remainder of the lemma, let $\hat{t}$ be an intermediate separator, that is $x_i \neq \hat{t}$ for all $i \in \{1, \ldots, n\}$. Under this requirement, the projection onto C can be written in closed form locally, and is differentiable.

Let I from (b) be fixed and write $u := \sum_{i \in I} e_i \in \mathbb{R}^n$ and $v := e - u \in \mathbb{R}^n$ as being the index vectors of I and I^C, respectively. Then the following holds:

(c) $p_i > 0$ if and only if $i \in I$.

(d) $p = x + {}^1\!/\!_d \cdot \left(\lambda_1 - u^T x\right) u - v \circ x$.

(e) There exists $\varepsilon > 0$ and a neighborhood $U := \{ s \in \mathbb{R}^n \mid \|x - s\|_2 < \varepsilon \}$ of x, such that $\operatorname{sgn}(\operatorname{proj}_C(s)) = \operatorname{sgn}(p)$ for all $s \in U$, that is the projection of s onto C has zeros at exactly the same entries as the projection of x onto C.

(f) $\operatorname{proj}_C(s) = s + {}^1\!/\!_d \cdot \left(\lambda_1 - u^T s\right) u - v \circ s$ for all $s \in U$.

(g) $\tilde{x} \mapsto \operatorname{proj}_C(\tilde{x})$ is differentiable in x.

Proof. (a) Follows directly like in Lemma 34(a).

(b) Let $\tau \in S_n$ such that $y := P_\tau x$ is sorted in descending order. With Lemma 28(b) and the analysis of function `proj_C` from Theorem 38 there is an index $j^* \in \{1, \dots, n\}$, such that $\hat{t} = t_{j^*} := {}^1\!/\!_{j^*} \cdot \left(\sum_{i=1}^{j^*} y_i - \lambda_1\right)$. Undoing the permutation τ such that $I := \{ \tau^{-1}(i) \mid i \in \{1, \dots, j^*\} \}$ yields the claim.

(c) From (b) and Lemma 28(b) follows $y_1 \geq \dots \geq y_{j^*} \geq \hat{t} \geq y_{j^*+1} \geq \dots \geq y_n$ if $I \neq \{1, \dots, n\}$ or $y_n \geq \hat{t}$ if $I = \{1, \dots, n\}$, respectively. Because $\hat{t} \neq x_i = y_{\tau^{-1}(i)}$ for all $i \in \{1, \dots, n\}$ by requirement, either $y_{j^*} \gneqq \hat{t} \gneqq y_{j^*+1}$ or $y_n \gneqq \hat{t}$ holds. Clearly $\{x_i \mid i \in I\} = \{y_i \mid i \in \{1, \dots, j^*\}\}$ by definition of I, hence $x_i \gneqq \hat{t}$ for all $i \in I$ and $\hat{t} \gneqq x_j$ for all $j \notin I$. The claim follows with $p_i = \max(x_i - \hat{t} \cdot e,\ 0)$.

(d) With (b) and by definition of u follows $\hat{t} = {}^1\!/\!_d \cdot \left(u^T x - \lambda_1\right)$. Let $i \in I$, then $p_i = x_i - \hat{t}$ with (a) and (c), and $u_i = 1$ and $v_i = 0$ by definition. Therefore $e_i^T \left(x + {}^1\!/\!_d \cdot \left(\lambda_1 - u^T x\right) u - v \circ x\right) = x_i - \hat{t} \cdot u_i - v_i \cdot x_i = x_i - \hat{t} = p_i$. Now let $j \notin I$, then $p_j = 0$ with (c). By definition $u_j = 0$ and $v_j = 1$, hence $e_j^T \left(x + {}^1\!/\!_d \cdot \left(\lambda_1 - u^T x\right) u - v \circ x\right) = x_j - \hat{t} \cdot u_j - v_j \cdot x_j = x_j - x_j = 0 = p_j$.

(e) Let $\delta := \min_{i \in \{1, \dots, n\}} |x_i - \hat{t}|$ and $\varepsilon := {}^\delta\!/\!_2$. It is $\delta, \varepsilon > 0$ because $\hat{t}$ is an intermediate separator by requirement. Let U be defined like in the claim, let $s \in U$ and define $\hat{u} := {}^1\!/\!_d \cdot (\sum_{i \in I} s_i - \lambda_1)$. Here I is still the index set from the projection of x onto C, see (b). With Lemma 28(b) it is sufficient to show that $s_i - \hat{u} > 0$ for $i \in I$ and $s_j - \hat{u} < 0$ for $j \notin I$, because then $q := \operatorname{proj}_C(s) = \max(s - \hat{u} \cdot e,\ 0)$ with the readily defined $\hat{u}$. When this holds, then trivially $q_i > 0$ for $i \in I$ and $q_j = 0$ for $j \notin I$, and hence $\operatorname{sgn}(q) = \operatorname{sgn}(p)$.

Because $s \in U$, Proposition 2 implies $|x_i - s_i| \leq \max_{j \in \{1, \dots, n\}} |x_j - s_j| = \|x - s\|_\infty \leq \|x - s\|_2 < \varepsilon$ and thus $-\varepsilon < x_i - s_i < \varepsilon$ for all $i \in \{1, \dots, n\}$. Furthermore, $\hat{t} - \hat{u} = {}^1\!/\!_d \cdot \sum_{i \in I} (x_i - s_i) \in (-\varepsilon,\ \varepsilon)$. With (c) follows that $x_i - \hat{t} > 0$ and thus $x_i - \hat{t} \geq \delta$ for $i \in I$, and conversely $x_j - \hat{t} < 0$ and thus $x_j - \hat{t} \leq -\delta$ for $j \notin I$. Let $i \in I$, then $s_i - \hat{u} = (s_i - x_i) + (x_i - \hat{t}) + (\hat{t} - \hat{u}) > -\varepsilon + \delta - \varepsilon = 0$. Conversely for $j \notin I$ follows $s_i - \hat{u} < \varepsilon - \delta + \varepsilon = 0$.

(f) In (e) it was shown that $\hat{u} := \hat{u}(s) := {}^1\!/\!_d \cdot (\sum_{i \in I} s_i - \lambda_1) = {}^1\!/\!_d \cdot \left(u^T s - \lambda_1\right)$ is suitable for satisfying Lemma 28(a) for all $s \in U$ for the projection onto C. The explicit, closed form for projection of all $s \in U$ onto C follows like in (d).

(g) The projection onto C can be written locally in closed form, see (f). In the same neighborhood, the index set of vanishing entries of the projected points does not change. Hence, the projection is differentiable as composition of differentiable functions. $\square$

The next results gathers the gradients of the individual projections. The chain rule then yields the gradient of $x \mapsto \mathrm{proj}_D(x)$ as multiplication of the individual gradients.

Lemma 41. The individual projections for $x \mapsto \mathrm{proj}_D(x)$ are differentiable almost everywhere. Their gradients are given as follows:

(a) $\frac{\partial \mathrm{proj}_H(x)}{\partial x} = E_n - {}^1\!/\!{}_n \cdot J_{n\times n}$.

(b) $\frac{\partial \mathrm{proj}_L(x)}{\partial x} = \frac{\sqrt{\rho}}{\|x-m\|_2}\left(E_n - {}^1\!/\!{}_{\|x-m\|_2^2} \cdot (x-m)(x-m)^T\right)$.

(c) $\frac{\partial \mathrm{proj}_C(x)}{\partial x} = E_n - {}^1\!/\!{}_d \cdot uu^T - \mathrm{diag}(v)$. Here, $I := \{\, i \in \{1,\ldots,n\} \mid e_i^T \mathrm{proj}_C(x) \neq 0 \,\}$, $d := |I|$, $u := \sum_{i\in I} e_i \in \mathbb{R}^n$ and $v := e - u \in \mathbb{R}^n$.

(d) $\frac{\partial \mathrm{proj}_{L_I}(x)}{\partial x} = \frac{\sqrt{\rho_I}}{\|x-m_I\|_2}\left(E_n - {}^1\!/\!{}_{\|x-m_I\|_2^2} \cdot (x-m_I)(x-m_I)^T\right)$.

Proof. (a) proj_H is clearly differentiable everywhere. Its gradient follows from Lemma 16, Lemma 66(c) and $ee^T = J_{n\times n}$.

(b) When $x \neq m$, then proj_L is differentiable as composition of differentiable functions. If $x = m$, then trivially in every neighborhood of x there is a point not equal m, hence proj_L is differentiable almost everywhere.

For the gradient computation, Lemma 19 yields $\mathrm{proj}_L(x) = m + \delta(x)\cdot(x-m)$ with $\delta(x) = \sqrt{\rho}/\|x-m\|_2$. Because ρ does not depend on x, the quotient rule and Lemma 66(a) yield $\partial\delta(x)/\partial x = -\sqrt{\rho}/\|x-m\|_2^3 \cdot (x-m)^T$. Hence, with application of the product rule follows

$$\frac{\partial \mathrm{proj}_L(x)}{\partial x} = \delta(x)\cdot\frac{\partial(x-m)}{\partial x} + (x-m)\cdot\frac{\partial\delta(x)}{\partial x} = \frac{\sqrt{\rho}}{\|x-m\|_2}E_n - \frac{\sqrt{\rho}}{\|x-m\|_2^3}(x-m)(x-m)^T.$$

(c) proj_C can be carried out via $\mathrm{proj}_C(x) = \max(x - \hat{t}\cdot e,\ 0)$, where $\hat{t} \in \mathbb{R}$ is an average value modulo an additive constant of a subset I of entries of x, see Lemma 40(b). First assume $\hat{t}$ is an intermediate separator, then there is a neighborhood U of x such that $\mathrm{proj}_C(s) = s + {}^1\!/\!{}_d \cdot \left(\lambda_1 - u^T s\right) u - v \circ s$ for all $s \in U$. Here, I, d, u and v are exactly as given in the claim with Lemma 40(f). Application of Lemma 66(c) and Lemma 66(e) yields the gradient.

Now consider the case when $\hat{t}$ is not an intermediate separator, that is when there is at least one collision $x_c = \hat{t}$ where $c \in \{1,\ldots,n\}$. For showing that proj_C is differentiable almost everywhere, it suffices to show that in every neighborhood of x a point exists, where proj_C is differentiable. Consider vector y which is x sorted in descending order using a permutation $\tau \in S_n$. By construction of $\hat{t}$ using Algorithm 4.1, there is an index $j^* \in \{1,\ldots,n\}$ with $y_{j^*} \gneqq$

$\hat{t} = y_{j^*+1} = x_c$. Since $\hat{t} = {}^1\!/_{j^*} \cdot \left(\sum_{i=1}^{j^*} y_i - 1\right)$, it does not depend on the actual value of x_c as long as it is less or equal to $\hat{t}$ and the relative ordering of the entries of y does not change.

For construction of a point where proj_C is differentiable in the neighborhood of x, let $\varepsilon > 0$. Let $J := \{\, j \in \{1,\dots,n\} \mid x_j = \hat{t}\,\}$ be the index set of all collisions with $\hat{t}$. Define $\delta := {}^{\varepsilon}\!/\!\sqrt{4|J|} > 0$ and consider $y := x - \delta\sum_{j\in J} e_j$. It is $\|y-x\|_2^2 = \delta^2 \cdot |J| = {}^{\varepsilon^2}\!/_4$, hence $\|y-x\|_2 < \varepsilon$. From the discussion above follows that $\hat{t}$ is independent of the entries of x with indices in J, as long as they are less than or equal than $\hat{t}$. Because $\delta > 0$, these entries in y are strictly smaller than $\hat{t}$, therefore $\hat{t}$ is an intermediate separator of y. As a consequence, proj_C is differentiable in y, and hence proj_C is differentiable almost everywhere.

(d) Follows exactly like in (b). □

Clearly, the gradients for proj_H and for proj_L are special cases of the gradients of proj_C and $\operatorname{proj}_{L_I}$, respectively. Therefore, they need no separate handling in the computation of the overall gradient. Exploiting the special structure of the matrices involved and the readily sorted input, the gradient computation can be further optimized as described in the following result.

Theorem 42. Let $x^* \in \mathbb{R}^n$ be sorted in descending order, and $x \mapsto \operatorname{proj}_D(x)$ be differentiable in x^*. Let $h \in \mathbb{N}$ denote the number of iterations Algorithm 4.3 needs to terminate. In every iteration of the algorithm, store the following values (mentioned lines reference Algorithm 4.3):

- $d_i \in \mathbb{N}$ denoting the current dimensionality as determined by Lines 23 and 25.
- $\delta_i := \sqrt{{}^{\rho}\!/_{\varphi}} \in \mathbb{R}$ where ρ and φ are determined in Line 5.
- $r(i) := y - m_I \in \mathbb{R}^{d_i}$ as computed in Line 9.

Let $N := d_h$ denote the number of non-vanishing entries in the result, and define $s(i) := (r(i)_1,\ \dots,\ r(i)_N)^T \in \mathbb{R}^N$ as being the vector of the first N entries of each $r(i)$. Further, for $i \in \{1,\dots,h\}$ let

$$A_i := \delta_i E_N - {}^{\delta_i}\!/_{d_i} \cdot J_{N\times N} - \alpha_i s(i)s(i)^T + {}^{\alpha_i}\!/_{d_i} \cdot s(i)s(i)^T J_{N\times N} \in \mathbb{R}^{N\times N} \text{ where } \alpha_i := {}^{\delta_i}\!/_{\|r(i)\|_2^2},$$

and define $A := \prod_1^{i=h} A_i = A_h \cdots A_1 \in \mathbb{R}^{N\times N}$. Then the gradient of $x \mapsto \operatorname{proj}_D(x)$ is given by ${}^{\partial \operatorname{proj}_D(x)}\!/_{\partial x}\big|_{x^*} = \operatorname{diag}(A,0)$, that is a block diagonal matrix where the quadratic submatrix with row and column indices from 1 to N is given by A, and the remaining entries are all vanishing.

Proof. First consider one single iteration $i \in \{1,\dots,h\}$ of Algorithm 4.3, and write $d := d_i$, $\delta := \delta_i$, $\alpha := \alpha_i$ and $r := r(i)$ for short. With Lemma 41, the gradient $G_C \in \mathbb{R}^{n\times n}$ of the projection onto both H and C is of the form $G_C := E_n - {}^1\!/_d \cdot uu^T - \operatorname{diag}(v)$. Here $u := \sum_{i=1}^d e_i$ and $v := e - u$, and no explicit index set I is required because of the overall input x^* and all intermediate vectors being sorted, see the proof of Theorem 38. Let $q := (r_1,\dots,r_d,0,\dots,0)^T \in \mathbb{R}^n$ be a copy of r padded by zeros to achieve full dimensionality n. The gradient of the projection onto L and in general L_I is given by $G_L := \delta E_n - \alpha qq \in \mathbb{R}^{n \times n} \in \mathbb{R}^{n \times n}$ using Lemma 41. The gradient of the whole iteration is given by the chain rule to yield

$$G := G_L G_C = \delta E_n - \delta/d \cdot uu^T - \delta \operatorname{diag}(v) - \alpha qq^T + \alpha/d \cdot qq^T uu^T + \alpha qq^T \operatorname{diag}(v).$$

Let $O \in \mathbb{R}^{(n-d)\times(n-d)}$ be the matrix with all entries being zero, then G is a block diagonal matrix of a matrix from $\mathbb{R}^{d \times d}$ and O: First note that $E_n - \operatorname{diag}(v) = \operatorname{diag}(E_d, O)$, $uu^T = \operatorname{diag}(J_{d\times d}, O)$, and $qq^T = \operatorname{diag}(rr^T, O)$. Therefore, $qq^T \operatorname{diag}(v) = \operatorname{diag}(rr^T, O_d) \cdot \left(\begin{smallmatrix} 0 & 0 \\ 0 & E_{n-d} \end{smallmatrix}\right) = 0$, and $qq^T uu^T = \operatorname{diag}(S, O_d)$ where $S := rr^T \cdot J_{d\times d} \in \mathbb{R}^{d\times d}$. Substitution then yields

$$\begin{aligned} G &= \delta \operatorname{diag}(E_d, O_d) - \delta/d \cdot \operatorname{diag}(J_{d\times d}, O_d) - \alpha \operatorname{diag}(rr^T, O_d) + \alpha/d \cdot \operatorname{diag}(S, O_d) \\ &= \operatorname{diag}\left(\delta E_d - \delta/d \cdot J_{d\times d} - \alpha rr^T + \alpha/d \cdot S, O_d\right). \end{aligned}$$

By denoting the gradient of iteration i by matrix $G_i \in \mathbb{R}^{n\times n}$ for $i \in \{1, \dots, h\}$ and by application of the chain rule follows that the gradient of all iterations is given by $\hat{G} := \prod_1^{i=h} G_i$. In $\hat{G}$, all entries but the top left submatrix of dimensionality $N \times N$ are vanishing, where $N := d_h$. This is because the according statement applies to G_i for the top left submatrix of dimensionality $d_i \times d_i$, and $d_1 > \dots > d_h$ holds, and only the according entries survive the matrix multiplication.

Therefore it is sufficient to compute only the top left $N \times N$ entries of the gradients of the individual iterations, as the remaining entries are not relevant for the final gradient. This is reflected by the definition of matrices A_i for $i \in \{1, \dots, h\}$ from the claim. □

The gradient can thus be computed using matrix-matrix multiplications, where the matrices are square and the edge length is the number of nonzero entries in the result of the projection. The computation is therefore more efficient than using the $n \times n$ matrices of the individual projections. However, when the target degree of sparseness is low, and thus the amount of nonzero entries N in the result of the projection is large, gradient computation can become very inefficient. Most learning algorithms require only the product of the gradient with a vector. In this situation, the procedure can be sped up:

Corollary 43. Let $x \in \mathbb{R}^n$ be sorted in descending order, and $s \mapsto \operatorname{proj}_D(s)$ be differentiable in x. Then the product of the gradient of $s \mapsto \operatorname{proj}_D(s)$ in x with an arbitrary vector can be computed using vector operations only.

Proof. First note that because of the associativity of the matrix product it is enough to consider the product of the gradient $G \in \mathbb{R}^{n\times n}$ of one iteration of Algorithm 4.3 with one vector $y \in \mathbb{R}^n$. Because of the statements of Theorem 42, it suffices to consider the top left $N \times N$ entries of G and the first N entries of y, as all other entries vanish.

Therefore, let $A := \delta E_N - \delta/d \cdot J_{N\times N} - \alpha ss^T + \alpha/d \cdot ss^T J_{N\times N} \in \mathbb{R}^{N\times N}$ be the non-vanishing block of G as given by Theorem 42, let $u := J_{N\times 1} \in \mathbb{R}^N$ be the vector of ones such that $uu^T = J_{N\times N}$, and let $z := (y_1, \ldots, y_N)^T \in \mathbb{R}^N$ denote the vector with the first entries of y. Using matrix product associativity and distributivity over multiplication with a scalar yields

$$Az = \delta\left(z - {}^1\!/\!_d \langle z, u\rangle \cdot u\right) + \alpha\left({}^1\!/\!_d \langle s, u\rangle \langle z, u\rangle - \langle s, z\rangle\right) s,$$

where $\langle z, u\rangle = \sum_{i=1}^N z_i$ and $\langle s, u\rangle = \sum_{i=1}^N s_i$. Hence Az can be computed in-place from z by subtraction of a scalar value from all entries, rescaling by δ, and adding a scaled version of s. □

Theorem 42 only states the gradient structure for sorted vectors and with respect to the non-negative projection. The gradients for unsorted vectors, or when non-negative solutions are not required, can easily be recovered from the special gradient.

Lemma 44. Let $x^* \in \mathbb{R}^n$ be a point, let $\tau \in S_n$ such that $y^* := P_\tau x^*$ is sorted in descending order and let $x \mapsto \operatorname{proj}_D(x)$ be differentiable in y^* with gradient $B \in \mathbb{R}^{n\times n}$ as given by Theorem 42. Then $x \mapsto \operatorname{proj}_D(x)$ is differentiable in x^* with gradient ${}^{\partial \operatorname{proj}_D(x)}\!/\!_{\partial x}\big|_{x^*} = P_\tau^T B P_\tau$.

Proof. First note that $x^* = P_{\tau^{-1}} y^*$ and $P_\tau^T = P_{\tau^{-1}} = P_\tau^{-1}$. As τ is fixed in a neighborhood of x^* when its entries are pairwise unequal, the projection from arbitrary points is only a linear transformation of the projection of sorted points almost everywhere. Therefore, the projection is differentiable in x^*, and because $\operatorname{proj}_D(x^*) = P_\tau^T \operatorname{proj}_D(P_\tau x^*) = P_\tau^T \operatorname{proj}_D(y^*)$ follows

$$\frac{\partial \operatorname{proj}_D(x)}{\partial x} = \frac{\partial P_\tau^T \operatorname{proj}_D(P_\tau x)}{\partial x} = P_\tau^T \cdot \frac{\partial \operatorname{proj}_D(P_\tau x)}{\partial P_\tau x} \cdot \frac{\partial P_\tau x}{\partial x} = P_\tau^T \cdot \frac{\partial \operatorname{proj}_D(P_\tau x)}{\partial P_\tau x} \cdot P_\tau$$

with the chain rule. □

The gradient for arbitrary points is therefore the gradient of the accordingly sorted point, where rows and columns have been permuted according to the sorting permutation. Because only the top left $N \times N$ submatrix of the gradient of the sorted point is populated with values that may be nonzero, only rows and columns with indices from $\tau^{-1}(1), \ldots, \tau^{-1}(N)$ have to be populated in the final gradient.

Lemma 45. Let $x^* \in \mathbb{R}^n$ and let $x \mapsto \operatorname{proj}_D(x)$ be differentiable in $|x^*|$. Then $x \mapsto \operatorname{proj}_L(x)$ is differentiable in x^* with ${}^{\partial \operatorname{proj}_L(x)}\!/\!_{\partial x}\big|_{x^*} = \operatorname{diag}(\operatorname{sgn}(x)) \cdot \left({}^{\partial \operatorname{proj}_D(x)}\!/\!_{\partial x}\big|_{|x^*|}\right) \cdot \operatorname{diag}(\operatorname{sgn}(x))$.

Proof. Follows analogously to Lemma 44, using $|x| = \operatorname{sgn}(x) \circ x = \operatorname{diag}(\operatorname{sgn}(x))x$. □

An efficient realization of the gradient of the sparseness-enforcing projection operator, when non-negativity is not required, is thus to compute the gradient for the absolute value of the input point, and remembering the signs of the individual entries. After carrying out the non-negative projection and computation of its gradient, the actual gradient can be recovered by inverting the signs of the rows and columns where the original point had negative entries.

Algorithm 4.5: Computation of $\operatorname{proj}_D(x)$ for $x \in \mathbb{R}^n$, original variant [26, 63].

Input: $x \in \mathbb{R}^n$ and $\lambda_1, \lambda_2 \in \mathbb{R}_{>0}$ with $\lambda_2 \leq \lambda_1 \leq \sqrt{n}\lambda_2$.
Output: $s \in \operatorname{proj}_D(x)$.

```
  // Project onto target hyperplane
1 $r := x + 1/n \cdot (\lambda_1 - e^T x)\, e$;
  // Project onto target hypersphere
2 $s := m + \delta(r - m)$ with $\delta > 0$ such that $\|s\|_2 = \lambda_2$;
  // Enter recursion unless solution from $D$ already found
3 if $I := \{\, i \in \{1,\dots,n\} \mid s_i \geq 0 \,\} \neq \{1,\dots,n\}$ then
      // Remove negative entries from $s$
4     $d := |I|$ and write $I = \{i_1,\dots,i_d\}$;
5     Let $\tilde{x} \in \mathbb{R}^d$ with $\tilde{x}_j = s_{i_j}$ for $j \in \{1,\dots,d\}$;
      // Recursion with reduced dimension
6     Let $\tilde{s} \in \operatorname{proj}_{\tilde{D}}(\tilde{x}) \subseteq \mathbb{R}^d$, where $\tilde{D} := \{\, a \in \mathbb{R}^d_{\geq 0} \mid e^T a = \lambda_1$ and $\|a\|_2 = \lambda_2 \,\}$;
      // Construct result by stuffing zero entries
7     Let $s := \sum_{j=1}^{d} \tilde{s}_j e_{i_j} \in \mathbb{R}^n$ be the result;
8 end
```

4.6 Comparison with Alternative Methods

An alternative algorithm for computing the sparseness-enforcing projection operator π was proposed in this chapter. The implicit description given by Algorithm 4.2 is quite simple, and has been shown rigorously to compute the correct result and to always terminate. Based on the symmetries of the sparseness measure σ, an improved method was given in Algorithm 4.3. Furthermore, it was shown that π is differentiable almost everywhere, and that gradient computation can be realized efficiently by multiplication of quadratic matrices with edge length equal to the number of nonzero entries in the result of the projections. In the situation where only the product of the gradient with an arbitrary vector is required, the multiplication can be carried out implicitly which reduces computational complexity.

4.6.1 Hoyer's Original Algorithm

For comparison of these results with previous results, consider Algorithm 4.5 as originally proposed by [26]. It is essentially the same algorithm as in [63], except that in the latter publication at most one entry is set to zero, and recursion is always entered with the working dimensionality being reduced by exactly one. Note that it is also possible to compute projections onto simplexes by recursively setting negative entries to zero [70]. Hence the original algorithm can be rendered to be very similar to Algorithm 4.2 by using the method of [70] for performing simplex projections. However, when the method of [71] is used the computational complexity for π can be significantly reduced.

It is noteworthy that Algorithm 4.3 always requires at most the same number of iterations of alternating projections as Algorithm 4.5. In the latter algorithm, a projection onto the non-negative orthant $\mathbb{R}^n_{\geq 0}$ is employed to achieve vanishing coordinates in the working vector, which can be expressed formally as $\mathrm{proj}_{\mathbb{R}^n_{\geq 0}}(x) = \max(x,\ 0)$. Algorithm 4.3 yields zero entries as result of a simplex projection, which can be written as $\mathrm{proj}_C(x) = \max(x - \hat{t} \cdot e,\ 0)$ with $\hat{t} \in \mathbb{R}$ chosen accordingly, see Lemma 28. The results of Lemma 34 have led to the conclusion that $\hat{t} \geq 0$ in the cases considered in this chapter. It is obvious that carrying out the operation $\max(x - \hat{t} \cdot e,\ 0)$ where $\hat{t} \geq 0$ decreases the L_0 pseudo-norm more than just by using $\max(x,\ 0)$. Hence with induction for the number of non-vanishing entries in the working vector, the number of iterations the proposed algorithm needs to terminate is bounded from above by the number of iterations the original method needs to terminate given the same input.

Numerical experiments have been carried out to assess the performance of the improved Algorithm 4.3 and the original version given in Algorithm 4.5. First, the required number of alternating projection iterations for computing projections onto D was measured. For this, random vectors with a sparseness of 0.15 were sampled and input to both algorithms to yield the best approximations with unit L_2 norm and a target sparseness degree of $\sigma^* := 0.90$. In doing so, the exact same random vectors were used for both algorithms to yield comparable results. During the run-time of the algorithms, the number of iterations necessary to compute the result were counted and the respective progress of the working dimensionality reduction was recorded. This process was carried out independently for dimensionalities up to one million, several orders of magnitude greater than what was needed in practice for this work. For each dimensionality one thousand random vectors were used to be able to make statistically robust statements.

The number of iterations Algorithm 4.5 and Algorithm 4.3 needed to terminate given the random vectors as input is depicted in Figure 4.9. The number of iterations grows very slowly with the problem dimensionality for both algorithms. This was already observed by [26] for the original algorithm; only between 12 and 14 iterations for input vectors with one million entries are needed. With Algorithm 4.3, this can be improved to only 9 to 10 iterations, which amounts to roughly 30% fewer iterations. Due to the small slope in the curves, it can be conjectured that this quantity is at most logarithmic in the problem dimensionality n. When this holds, both algorithms would have at most quasilinear time complexity, since each iteration can be carried out in time linear with respect to the number of non-vanishing entries in the working vector. Because of the sorting in the beginning of Algorithm 4.3, quasilinearity is also the lower bound of time complexity [23].

The progress of working dimensionality reduction for problem dimensionality $n = 1000$ is shown in Figure 4.10, averaged over the 1000 input vectors from the experiment. After the

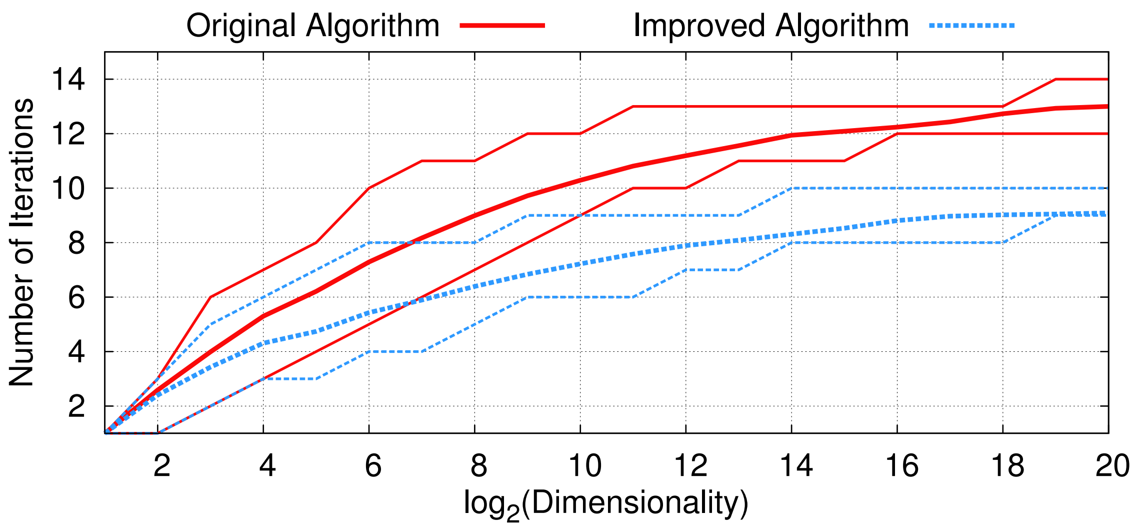


Figure 4.9: Comparison of the number of iterations that the original Algorithm 4.5 and the improved Algorithm 4.3 require for projections onto D. Input vectors with sparseness 0.15 were sampled randomly and their projections with a target sparseness of 0.90 were computed. Thick lines mark the mean number of iterations, and thin lines show the minimum and maximum number of iterations. Adapted from [66].

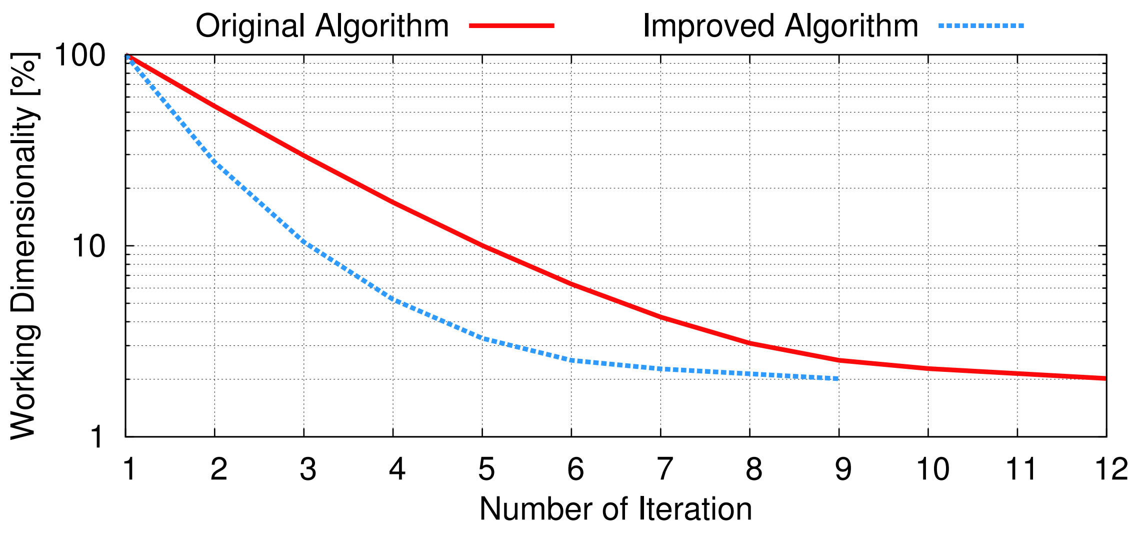


Figure 4.10: Progress of working dimensionality reduction of the original Algorithm 4.5 and the improved Algorithm 4.3. Both algorithms were input with random vectors with 1000 entries and an initial sparseness of 0.15 to compute projections with sparseness 0.90. Standard deviations were less than 1% and have been omitted for improved visibility. The proposed algorithm reduces dimensionality more quickly and terminates earlier than the original algorithm. Adapted from [66].

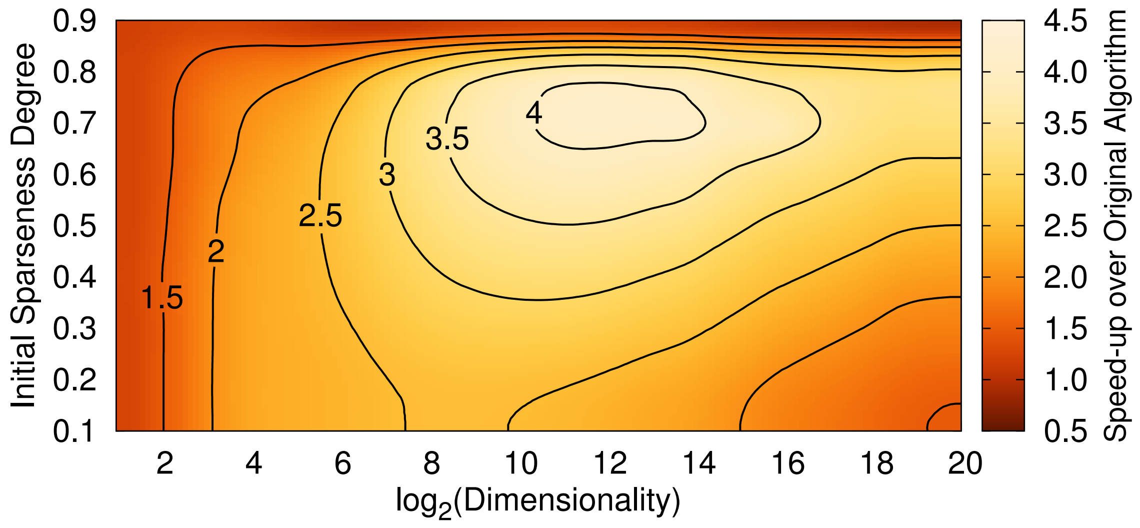


Figure 4.11: Relative speed-up of the improved algorithm over the original algorithm when run on a computer for a variety of input dimensionality and initial vector sparseness. The target sparseness for the projection was 0.90. The improved variant is at least three times faster than the original one for a broad range of applications. Adapted from [66].

first iteration of both algorithms, that is after projecting onto H and L, the working dimensionality still matches the input dimensionality as here no values are set to zero explicitly. Starting with the second iteration, dimensions are discarded by projecting onto $\mathbb{R}^n_{\geq 0}$ in Algorithm 4.5 and onto C in Algorithm 4.3, which yields vanishing entries in the working vectors. When Algorithm 4.5 is used, about 54% of all entries are nonzero after the second iteration. This improves to only 27% using Algorithm 4.3. This trend continues in subsequent iterations and the final working dimensionality is reached more quickly with the improved algorithm. Although using Algorithm 4.1 to perform the simplex projection is more expensive than just setting negative entries to zero in the orthant projection, the overhead quickly amortizes because of the boost in dimensionality reduction.

For a final experiment, both algorithms were implemented as C++ programs, where a highly optimized Basic Linear Algebra Subprograms (BLAS) library [74, 75, 76] was used for the matrix and vector operations. These programs were then run on an Intel Core i7-980X processor to determine the relative speed-up of the improved algorithm due to using simplex projections and the access to unit-stride arrays. For dimensionalities up to one million entries, a set of vectors with varying initial sparseness were sampled. The magnitude of these sets of vectors for every combination of dimensionality and initial sparseness was set such that the run-time of the algorithms was several orders of magnitudes greater than the latency time of the operating system. Next, the absolute time needed for the algorithms to compute the projections with a target sparseness of 0.90 were measured, and their ratio was taken to compute the relative speed-up. The results of this experiment are depicted in Figure 4.11. A speed-up factor of at least three is achieved for vectors with dimensionality greater than 2^8 and an initial sparseness greater than 0.40. For low initial sparseness, as achieved by randomly sampled vectors, a speed-up factor of at least 2 can be achieved for a broad spectrum of dimensionality between 2^4 and 2^{14}.

4.6.2 Further Improvements

The algorithms for the sparseness projection π discussed in this chapter were geometrically motivated. Recently, an algorithm based on the analytical method of Lagrange multipliers was proposed [77], but not rigorously proved to be correct. Further, the algorithm requires sorting of the argument and hence its run-time complexity is at least quasilinear and does not improve on the complexity class of the algorithms proposed in this chapter.

An extension of the results of [77] has shown, however, that it is also possible to compute π by casting the projection as a root-finding problem, similar to the method of [72] for projections onto canonical simplexes. More formally, in [78] it was shown that when $x \in \mathbb{R}^n_{\geq 0} \setminus D$ is given such that its projection onto the target set $D = S^{(\lambda_1, \lambda_2)}_{\geq 0}$ is unique, then there is exactly one number $\alpha^* \in \mathbb{R}$ such that the projection can be expressed as

$$\operatorname{proj}_D(x) = \tfrac{\lambda_2}{\left\lVert \max(x - \alpha^* \cdot e,\ 0) \right\rVert_2} \cdot \max\left(x - \alpha^* \cdot e,\ 0\right).$$

This representation can also be obtained by showing that in each step of the alternating projections method, the working vector can be expressed explicitly in terms of the input vector. Further, α^* is the unique root of the auxiliary function

$$\Psi\colon \left[0,\ \max_{i \in \{1,\dots,n\}} x_i\right) \to \mathbb{R}, \qquad \alpha \mapsto \frac{\left\lVert \max\left(x - \alpha \cdot e,\ 0\right) \right\rVert_1}{\left\lVert \max\left(x - \alpha \cdot e,\ 0\right) \right\rVert_2} - \frac{\lambda_1}{\lambda_2},$$

which is strictly decreasing in the relevant interval [78]. By exploiting the special structure of Ψ and using the Bisection method it was further possible to construct a linear time and constant space algorithm to find α^* and in turn the result of the sparseness projection without needing to sort the argument [78]. It can be inferred that the sparseness projection can be accomplished in at most $8n + 200$ multiply-accumulate operations and $10n + 50$ memory accesses where n denotes the problem dimensionality by inspecting the algorithm given in [78]. This estimate is a worst case upper bound, and it is often underrun in practical applications.

4.7 Discussion

Hoyer's sparseness measure σ is appealing because it is a smooth approximation to the L_0 pseudo-norm, which is a natural sparseness measure but not even continuous. Although algorithms for computing projections onto the sets in which σ attains a constant value have already been proposed, a mathematically satisfactory proof was still lacking. This chapter analyzed the properties of σ and the corresponding sparseness projection π, proposed new algorithms for computation of π, and rigorously proved their correctness. Since the target sets for the projection are non-convex, it is not guaranteed that standard algorithms such as alternating projections compute correct solutions. Instead, it had to be demonstrated that each intermediate projection step does not tamper the solution set and that the number of iterations is bounded from above to guarantee termination in finite time. It was further shown in this chapter that π can be cast almost everywhere as differentiable vector-valued function. Again, because of the non-convexity of the target sets, existence of the gradient had to be shown explicitly rather than through standard arguments.

A comparison with Hoyer's original algorithm for the projection has shown that the algorithm proposed here requires fewer iterations of alternating projections, and also that it is several times faster when implemented as a computer program. Using analytical rather than geometrical methods it was moreover shown recently that the sparseness-enforcing projection operator can be computed in linear time and constant space in the problem dimensionality. This result can be understood as an extension of the work presented in this chapter. When considered in isolation, the analytical method lacks a deeper insight into the intermediate steps carried out during the projection. The methods motivated by geometry are far more intuitive and provide a solid basis for when other projection problems are investigated in the future.

5 Sparse Coding for Fast Classification

In this chapter, a model which is called *Sparse Coding for Fast Classification (SCFC)* is proposed. It extends Non-Negative Matrix Factorization with Sparseness Constraints with inference and classification capabilities. In doing so, SCFC provides widespread possibilities of controlling the individual capabilities of the model. Eventually, an algorithm for training an efficient classifier based on the sparse information processing paradigm is yielded.

This chapter is structured as follows. First, the techniques that SCFC builds upon are reviewed in greater detail than in Section 2.4 and their applicability to real-time classification tasks is analyzed. Then the SCFC model is proposed as an extension of these techniques. It is demonstrated that SCFC is able to reproduce results from unsupervised sparse coding, and that its classification capabilities are superior to ordinary MLPs given the same learning set. The chapter concludes with a discussion of SCFC's advantages and possible issues. This chapter contains material previously published in [79].

5.1 Non-Negative Matrix Factorization with Sparseness Constraints

Non-Negative Matrix Factorization with Sparseness Constraints (NMFSC) [26] is an extension of the linear, generative model termed Non-Negative Matrix Factorization (NMF) [41] to incorporate explicit sparseness constraints. Similar to Section 2.3 and Section 2.4, let X, W and H be non-negative matrices containing data samples, bases and code words, respectively. Then NMFSC is the problem of minimizing $E_{\text{NMF}}(W,H) := \|X - WH\|_F^2$ by variation of W and H, such that

$$\sigma(We_i) \overset{!}{=} \sigma_W \text{ for all } i \in \{1,\dots,n\} \text{ and } \sigma(H^T e_i) \overset{!}{=} \sigma_H \text{ for all } i \in \{1,\dots,n\}.$$

Here, $\sigma_W \in (0,\ 1)$ and $\sigma_H \in (0,\ 1)$ denote the target degree of sparse connectivity and sparse activity, respectively. E_{NMF} is convex in W and in H, when the other quantity is kept fixed, but it is not convex in W and H simultaneously. As heuristics, E_{NMF} is minimized via alternating application of the Bold Driver algorithm, see Appendix C.3.2. For ensuring that both W

and H fulfill the sparseness constraints, the sparseness-enforcing projection operator π from Chapter 4 is applied to the columns of W and the rows of H, respectively:

$$\begin{aligned}
\Pi_W \colon \mathbb{R}_{\geq 0}^{d\times n} \to \mathbb{R}_{\geq 0}^{d\times n}, \qquad & W \mapsto \tilde{W} \text{ where } \tilde{W} e_i := \pi\left(W e_i,\ \sigma_W,\ \text{non-negative, keep\_norm}\right),\\
\Pi_H \colon \mathbb{R}_{\geq 0}^{n\times M} \to \mathbb{R}_{\geq 0}^{n\times M}, \qquad & H \mapsto \tilde{H} \text{ where } \tilde{H}^T e_i := \pi\left(H^T e_i,\ \sigma_H,\ \text{non-negative, unit\_norm}\right).
\end{aligned}$$

In the definition of $\tilde{W}$ and $\tilde{H}$, it is $i \in \{1,\dots,n\}$. The gradient of the objective function is given by the following result:

Theorem 46. It is

$$\left(\frac{\partial E_{\text{NMF}}}{\partial W}\right)^T = -2\,(X - WH)\,H^T \in \mathbb{R}^{d\times n} \text{ and } \left(\frac{\partial E_{\text{NMF}}}{\partial H}\right)^T = -2W^T\,(X - WH) \in \mathbb{R}^{n\times M}.$$

Proof. First note that $E_{\text{NMF}} = \|\text{vec}(X - WH)\|_2^2 =: f_{\text{vec}}$ is vectorizable. With the chain rule and the properties of the Kronecker product as stated in Theorem 55 and Lemma 56 in Appendix B one yields

$$\begin{aligned}
\frac{\partial f_{\text{vec}}}{\partial\,\text{vec}(W)} &= \frac{\partial\,\|\text{vec}(X - WH)\|_2^2}{\partial\,\text{vec}(X - WH)} \cdot \frac{\partial\,\text{vec}(X - WH)}{\partial\,\text{vec}(W)} = -2\,\text{vec}(X - WH)^T \cdot (H^T \otimes E_d)\\
&= -2\left[(H \otimes E_d)\,\text{vec}(X - WH)^T\right]^T = -2\,\text{vec}\left((X - WH)H^T\right)^T,
\end{aligned}$$

and analogously $\partial f_{\text{vec}}/\partial\,\text{vec}(H) = -2\,\text{vec}\left(W^T(X - WH)\right)^T$. Therefore, the claim follows with Theorem 67 from Appendix B. □

The complete optimization method is summarized in Algorithm 5.1. First, matrices W and H are initialized randomly and projected to initially fulfill the sparseness constraints. During initialization, the columns of W are scaled to unit L_2 norm, while H is scaled implicitly by Π_H. Then, the initial step sizes for W and H are set to one, and the main optimization loop is entered. There, W and H are updated in an alternating fashion. The step sizes are adjusted using the Bold Driver heuristic, see Algorithm C.1, such that the objective function is non-increasing by the updates to W and H, respectively. Application of the sparseness-enforcing projection operator via Π_W and Π_H, respectively, guarantees that the sparseness constraints are fulfilled as loop-invariant.

5.1.1 Expected Code Word Sparseness

During optimization, the rows of the code word matrix H are enforced to be sparsely populated using the projection operator Π_H. Hence the influence of each basis on the input samples is limited, or in other words each basis is active for only a fraction of all samples. This property is also known as *lifetime sparseness* [80]. On the contrary, each column of H describes the

Algorithm 5.1: Non-Negative Matrix Factorization with Sparseness Constraints [26].

Input: Data matrix $X \in \mathbb{R}_{\geq 0}^{d \times M}$, target dimensionality of code words $n \in \mathbb{N}$, target sparseness degrees $\sigma_W, \sigma_H \in (0,\ 1)$.
Output: Matrix of bases $W \in \mathbb{R}_{\geq 0}^{d \times n}$ and matrix of code words $H \in \mathbb{R}_{\geq 0}^{n \times M}$, locally minimizing E_{NMF} subject to sparseness constraints.

```
  // Initialize matrix of bases $W$
1 $W_R \in_R \mathcal{N}^{d \times n}$; $W_R := |W_R|$; for $i := 1$ to $n$ do $W_R e_i := W_R e_i / \|W_R e_i\|_2$; $W := \Pi_W(W_R)$;
  // Initialize matrix of code words $H$
2 $H_R \in_R \mathcal{N}^{n \times M}$; $H_R := |H_R|$; $H := \Pi_H(H_R)$;
  // Optimize $W$ and $H$ subject to sparseness constraints
3 $\eta_W := 1$; $\eta_H := 1$;
4 while true do
5     $(W,\ \eta_W) :=$ bolddriver($W \mapsto E_{\text{NMF}},\ \Pi_W,\ W,\ \eta_W$);
6     $(H,\ \eta_H) :=$ bolddriver($H \mapsto E_{\text{NMF}},\ \Pi_H,\ H,\ \eta_H$);
7 end
```

activity of the entire population of bases for one sample, which is also called *population sparseness* [80]. It was shown empirically that these two quantities are uncorrelated when sparse coding models with implicit sparseness constraints are used [80]. In the situation with explicit sparseness constraints considered here, however, it can be shown that both values are connected.

After application of Π_H, each row has unit L_2 norm and an appropriate L_1 norm so that the target sparseness of σ_H is achieved. The L_1 norm is given by $\ell_H := \sqrt{M} - \sigma_H \cdot (\sqrt{M} - 1)$, see Table 4.1. In other words, $\left\|H^T e_i\right\|_1 = \ell_H$ and $\left\|H^T e_i\right\|_2 = 1$ hold for all $i \in \{1, \ldots, n\}$. By interchanging the order of summation, the empirical mean value of the L_1 norm of the code words can be computed:

$$M \cdot \mathbb{E}(\|h\|_1) = \sum_{j=1}^{M} \left\|He_j\right\|_1 = \sum_{j=1}^{M} \sum_{i=1}^{n} \left|e_i^T He_j\right| = \sum_{i=1}^{n} \left\|H^T e_i\right\|_1 = n\ell_H.$$

As the square root is a nonlinearity, a closed form solution for the empirical mean value of the L_2 norm of the code words cannot be deduced. However, the empirical mean value of the *squared* L_2 norm can be computed analogously:

$$M \cdot \mathbb{E}\left(\|h\|_2^2\right) = \sum_{j=1}^{M} \left\|He_j\right\|_2^2 = \sum_{j=1}^{M} \sum_{i=1}^{n} \left(e_i^T He_j\right)^2 = \sum_{i=1}^{n} \left\|H^T e_i\right\|_2^2 = n.$$

Note that there is no simple formula for $\mathbb{E}(\|h\|_2)$, but it is bounded from above because of $\mathbb{E}(\|h\|_2)^2 \leq \mathbb{E}\left(\|h\|_2^2\right)$, which holds for every random variable with finite second moment [81]. Hence the approximation

$$\mathbb{E}(\|h\|_2) \approx \sqrt{\mathbb{E}\left(\|h\|_2^2\right)}$$

Table 5.1: Empirical mean $\mathbb{E}(\sigma(h))$ of code word sparseness of NMFSC applied to the MNIST learning set, in comparison with the derived approximation $\widehat{\mathbb{E}}(\sigma(h))$ for different values of the row sparseness σ_H of the matrix of code words H. Δ denotes the absolute error of the approximation relative to the code word sparseness.

σ_H	0.100	0.200	0.300	0.400	0.500	0.600	0.700	0.800
$\mathbb{E}(\sigma(h))$	0.105	0.219	0.337	0.454	0.571	0.685	0.795	0.900
$\widehat{\mathbb{E}}(\sigma(h))$	0.114	0.228	0.341	0.455	0.569	0.683	0.797	0.911
Δ	0.008	0.009	0.005	0.001	0.002	0.002	0.002	0.010

is valid when the variance of $\|h\|_2$ is small, that is if $0 \approx \operatorname{Var}(\|h\|_2) = \mathbb{E}\big(\|h\|_2^2\big) - \mathbb{E}(\|h\|_2)^2$. Using a Taylor expansion it can be shown that the expected value of a ratio can be approximated by a ratio of expected values [82, 83], that is $\mathbb{E}\left(\|h\|_1/\|h\|_2\right) \approx \mathbb{E}(\|h\|_1)/\mathbb{E}(\|h\|_2)$. Therefore the empirical mean sparseness of code words can be approximated by a product of the row sparseness σ_H and a fraction:

$$\begin{aligned}\mathbb{E}(\sigma(h)) &= \frac{\sqrt{n} - \mathbb{E}\left(\|h\|_1/\|h\|_2\right)}{\sqrt{n}-1} \approx \frac{\sqrt{n} - \mathbb{E}(\|h\|_1)/\sqrt{\mathbb{E}(\|h\|_2^2)}}{\sqrt{n}-1} = \frac{\sqrt{n} - \sqrt{n/M}\cdot \ell_H}{\sqrt{n}-1} \\ &= \frac{\sqrt{n} - \sqrt{n/M}\left(\sqrt{M} - \sigma_H \cdot (\sqrt{M}-1)\right)}{\sqrt{n}-1} = \sigma_H \frac{\sqrt{n} - \sqrt{n/M}}{\sqrt{n}-1} =: \widehat{\mathbb{E}}(\sigma(h)).\end{aligned}$$

Here, $\widehat{\mathbb{E}}(\sigma(h))$ is used to denote the approximate term. This expression converges increasingly to $\sigma_H \cdot \sqrt{n}/(\sqrt{n}-1)$ for $M \to \infty$, which is slightly greater than σ_H. Therefore, the lifetime sparseness and the population sparseness are about equal since $\sqrt{n}/(\sqrt{n}-1)$ is about one when n is large.

To estimate the accuracy of this approximation, the resulting code word matrices H of the initial experiments of NMFSC applied to the MNIST data set as described in Chapter 3 were analyzed. For this, several values for σ_H were used to compute a sparse representation with $n := 64$ filters of the $M := 60\,000$ samples from the MNIST learning set.

The results are given in Table 5.1, where σ_H denotes the input parameter to NMFSC, $\mathbb{E}(\sigma(h))$ denotes the measured empirical mean of code word sparseness, $\widehat{\mathbb{E}}(\sigma(h))$ its approximation as computed above, and $\Delta := \left|\mathbb{E}(\sigma(h)) - \widehat{\mathbb{E}}(\sigma(h))\right|$ is the absolute error of the approximation. It is evident that the approximation is quite accurate, with a maximum absolute error of $\Delta \leq 0.010$ for the entire range depicted in Table 5.1. It should be noted that the values for $\sigma_H = 0.900$ have been omitted, as in this case there are samples which cannot be reproduced at all by NMFSC, and hence all entries of the according code words vanish, which is not a valid input to Hoyer's sparseness measure.

5.1.2 Suitability of NMFSC for Classification Tasks

Two fundamental problems arise when intending to employ NMFSC for classification tasks. The first one is the problem of code word inference. During the learning phase, for each data sample which is stored in a certain column of the data matrix X, the associated sparse code word is given by the corresponding column of the matrix of code words H. As H is a latent variable, its existence is guaranteed mathematically during the learning phase. However, its state cannot be observed during recall phase, when data samples that were unknown during the learning phase are fed to the model in order to compute a classification decision.

For unknown samples, no sparse code words that fit the input data and the model exist inherently, they have to be inferred explicitly. Formally, if $x \in \mathbb{R}^d$ is an input sample to be classified and $W \in \mathbb{R}^{d \times n}$ is the matrix of bases as determined during the learning phase, then a code word $h \in \mathbb{R}^n$ which minimizes $\|x - Wh\|_2^2$ has to be found. Additionally, h must fulfill a sparseness constraint. For this, the result from Section 5.1.1 using the target sparseness σ_H can be applied, to yield the following optimization problem:

$$\|x - Wh\|_2^2 \overset{!}{\to} \min\nolimits_h \text{ such that } \sigma(h) \overset{!}{=} \sigma_H \tfrac{\sqrt{n}}{\sqrt{n}-1}.$$

With original NMF, when no sparseness constraints are involved, code word inference is achieved by projection onto the bases [44, 84, 45] or by solving a non-negative least squares problem [85]. However, it is not guaranteed that this will yield a sparse code word with NMFSC. Another approach would be to carry out Algorithm 5.1, but projecting the code word h instead of the rows of H, and omitting updates to W and thus keeping the matrix of bases constant. As the sparseness projection is essentially the application of a soft-thresholding operation with a threshold determined by its argument, see Chapter 4 and [78], this is very similar to the iterative soft-thresholding algorithm from the domain of *compressed sensing* [86], see [69] for the corresponding case where projections onto L_0 pseudo-norm constraints are used. Since the optimization problem considered here is more complex due to the data-dependent threshold, it can be assumed that algorithms faster than carrying out several iterations of projected gradient descent are not available. Unfortunately, this optimization is very time-consuming and not real-time capable unless the optimization problem itself is relaxed.

The second problem that arises when employing NMFSC for classification tasks is poor discriminatory performance. Both NMF and NMFSC are unsupervised learning algorithms, they are not aware of any class labels that may be associated with the data samples. Therefore, sparse code words from samples with different class labels are not enforced to be well-separable. In fact, when using the sparse code words as features for subsequent classification, confusion among the classes is very high.

A heuristic for adding classification capabilities to NMFSC was described in Section 2.4.4 as proposed by [29]. There, all code words belonging to samples of the same class are enforced to be located within a hypersphere around the mean value of all code words of that particular class. The radius of the hypersphere is a parameter of the algorithm. In other words, the method can be interpreted as supervised clustering, where the number of clusters is equal to the number of classes. However, there is no guarantee that a distribution of code words exists for which both the reproduction error is minimized and the code words can be arranged in such a simple pattern. The requirement to achieve state-of-the-art classification performance makes using more sophisticated methods mandatory.

5.1.3 Non-Negativity versus Explicit Sparseness Constraints

As explained in Section 2.4.2, non-negativity induces sparse representations on certain data sets. The sparseness is merely a side product, and cannot be expected in the general case. By introduction of explicit sparseness constraints, NMFSC is guaranteed to compute sparse representations. In the definition of Hoyer's sparseness measure σ, there is no difference between vectors with only non-negative entries and arbitrary vectors. Likewise, the sparseness-enforcing projection can be used for both non-negative and unrestricted results. Therefore it is not necessary to insist on non-negativity if merely a sparse representation is desired.

Moreover, it is beneficial to normalize training samples to zero mean and unit variance in classification scenarios. This is because if all entries of an input data point are positive, then the filters processing the input data will be updated during learning by either increasing or decreasing all weights altogether [87]. In this case, a weight vector cannot change its direction other than by zigzagging, which takes a long time to converge [87]. Another argument is that a nonzero mean in the training samples causes the Hessian of the objective function to have a very large eigenvalue, which produces an error surface that is steep in some directions and shallow in others [88, 87]. If the nonlinear transfer functions and the weight vectors are scaled accordingly, then a scaling of the input data to unit variance prevents that the transfer function is evaluated in its saturated region, which would result in very small gradients such that training time is excessively increased [87]. Therefore, training data normalization is mandatory for fast training of classifiers using gradient-based methods, and enforcing non-negativity only prevents fast training.

5.2 Supervised Matrix Factorization with Sparseness Constraints and Fast Inference

Combining the ideas of Non-Negative Matrix Factorization with Sparseness Constraints [26], Sparse Encoding Symmetric Machine [33] and a two-layer MLP [89] can overcome the drawbacks of inefficient code word inference and poor classification performance. The Sparse Coding for Fast Classification model proposed in this chapter can be characterized as supervised matrix factorization with sparseness constraints and fast inference. It aims to compute

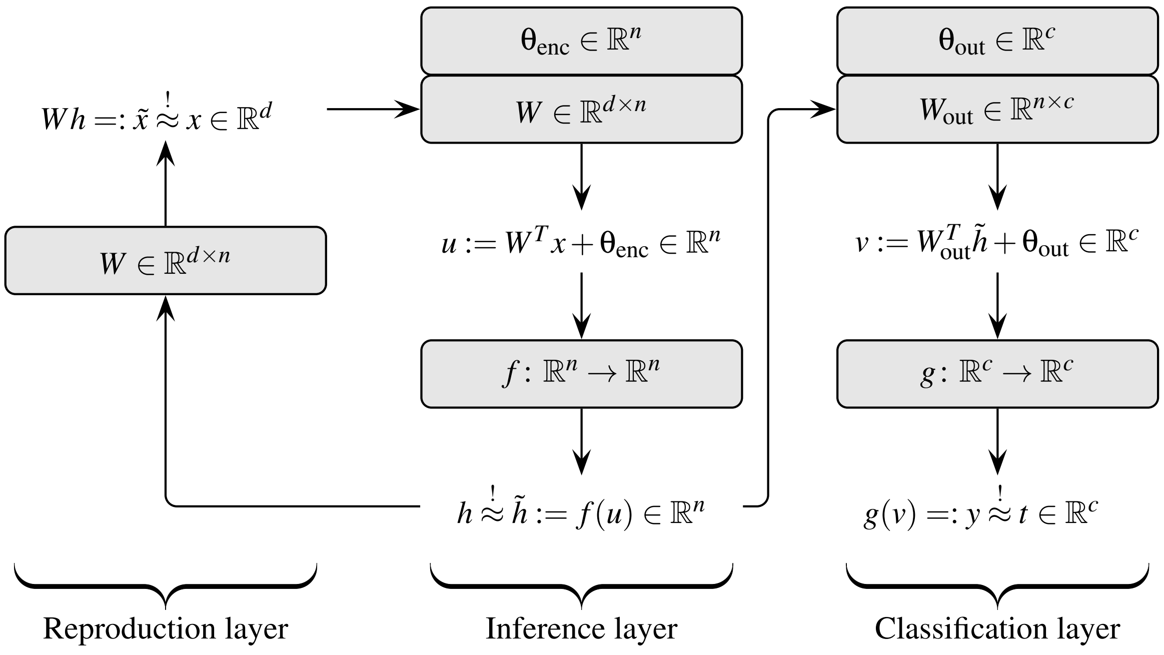

$\theta_{\text{enc}} \in \mathbb{R}^n$
$W \in \mathbb{R}^{d\times n}$
$\theta_{\text{out}} \in \mathbb{R}^c$
$W_{\text{out}} \in \mathbb{R}^{n\times c}$
$Wh =: \tilde{x} \overset{!}{\approx} x \in \mathbb{R}^d$
$W \in \mathbb{R}^{d\times n}$
$u := W^T x + \theta_{\text{enc}} \in \mathbb{R}^n$
$v := W_{\text{out}}^T \tilde{h} + \theta_{\text{out}} \in \mathbb{R}^c$
$f\colon \mathbb{R}^n \to \mathbb{R}^n$
$g\colon \mathbb{R}^c \to \mathbb{R}^c$
$h \overset{!}{\approx} \tilde{h} := f(u) \in \mathbb{R}^n$
$g(v) =: y \overset{!}{\approx} t \in \mathbb{R}^c$
Reproduction layer
Inference layer
Classification layer

Figure 5.1: SCFC architecture consisting of a reproduction path ($h \mapsto \tilde{x}$) and an inference path ($x \mapsto \tilde{h}$). The architecture is extended by a classification path ($x \mapsto \tilde{h} \mapsto y$) if a teacher signal t is available. In this supervised operating mode, SCFC resembles a two-layer MLP with sparseness constraints. Adapted from [79].

discriminative, sparse representations that can be employed in real-time classification tasks. A classifier based on the sparse information processing paradigm can be realized by incorporating teacher signals if any are associated with the input samples.

A sketch of the architecture of SCFC is given in Figure 5.1 for an individual sample $x \in \mathbb{R}^d$ with associated teacher signal $t \in \mathbb{R}^c$. The network dynamics are given by the following quantities. There is a *matrix of bases* $W \in \mathbb{R}^{d\times n}$ which is used to compute an approximation $\tilde{x} := Wh \in \mathbb{R}^d$ from a *latent code word* $h \in \mathbb{R}^n$. The matrix of bases is used as a *matrix of filters* in conjunction with a threshold vector $\theta_{\text{enc}} \in \mathbb{R}^n$ for computation of an *internal representation* $u := W^T x + \theta_{\text{enc}} \in \mathbb{R}^n$.

This internal representation is used to compute an approximation $\tilde{h} := f(u) \in \mathbb{R}^n$ to the latent code word h. The approximation results from using a transfer function $f\colon \mathbb{R}^n \to \mathbb{R}^n$. In the *unsupervised operating mode*, this data flow forms a feedback system through the reproduction path $h \mapsto \tilde{x}$ and the inference path $x \mapsto \tilde{h}$. Therefore, W is used for both encoding and decoding and hence the architecture is symmetric, or in other words its weights are tied. This approach is similar to Principal Component Analysis [28], Restricted Boltzmann Machines for deep auto-encoder networks [55] and Sparse Encoding Symmetric Machine [33].

In the *supervised operating mode*, the *classification path* $x \mapsto y$ is activated. It consists of a *matrix of weights* $W_{\text{out}} \in \mathbb{R}^{n \times c}$ and a vector of thresholds $\theta_{\text{out}} \in \mathbb{R}^c$ to compute a vector $v := W_{\text{out}}^T \tilde{h} + \theta_{\text{out}} \in \mathbb{R}^c$ from the approximated code word $\tilde{h}$. This vector is mapped through another transfer function $g\colon \mathbb{R}^c \to \mathbb{R}^c$ to yield a *classification decision* $y := g(v) \in \mathbb{R}^c$, which should approximate the teacher signal $t \in \mathbb{R}^c$. Hence, a two-layer MLP is formed together with the inference path. This is known to be a universal approximator [90, 91, 92], which means every sufficiently smooth data distribution can be approximated with arbitrary precision, provided the code word dimensionality n is large enough. Therefore, better classification results than compared with the simple approach from Section 2.4.4 can be expected.

5.2.1 Objective Function of the SCFC Architecture

In the following, one error function is defined for each of the three approximations in the SCFC model. Each error term consists of a sum over the individual samples and the corresponding target values which should be approximated. Deviations are measured using the squared error similarity measure, see Section C.6. For this, let $x^{(1)}, \ldots, x^{(M)} \in \mathbb{R}^d$ be the input samples, let $t^{(1)}, \ldots, t^{(M)} \in \mathbb{R}^c$ be the associated class labels, and let $h^{(1)}, \ldots, h^{(M)} \in \mathbb{R}^n$ be the associated latent code words. Further, let $X \in \mathbb{R}^{d \times M}$ denote the data matrix, let $T \in \mathbb{R}^{c \times M}$ denote the matrix of teacher signals and let $H \in \mathbb{R}^{n \times M}$ denote the matrix of latent code words, such that $x^{(\mu)} = Xe_\mu$, $t^{(\mu)} = Te_\mu$, and $h^{(\mu)} = He_\mu$ for $\mu \in \{1, \ldots, M\}$. The individual vectors are hence written next to each other in the corresponding matrices.

The overall objective function E_{SCFC} is a convex combination of three individual error functions, E_R for reproduction, E_I for inference and E_C for classification:

$$E_{\text{SCFC}} := (1 - \alpha_I - \alpha_C)E_R + \alpha_I E_I + \alpha_C E_C \text{ where } 0 \leq \alpha_I, \alpha_C \text{ and } \alpha_I + \alpha_C \leq 1.$$

The variables α_I and α_C control the trade-off between the error functions. Their role in the optimization strategy will be discussed in Section 5.2.3.

To enforce that the input data samples can be approximated using the code words, the *reproduction error* is defined as the squared error between the original samples from X and their approximations:

$$E_R := \textstyle\sum_{\mu=1}^{M} E_R^{(\mu)}, \text{ where } E_R^{(\mu)} := \left\| x^{(\mu)} - Wh^{(\mu)} \right\|_2^2 \text{ for } \mu \in \{1, \ldots, M\}.$$

Note that $E_R = E_{\text{NMF}} = \|X - WH\|_F^2$ with this definition.

The *inference error* measures the compatibility of the approximated code words and their latent counterparts by means of a squared error:

$$E_I := \textstyle\sum_{\mu=1}^{M} E_I^{(\mu)}, \text{ where } E_I^{(\mu)} := \left\| h^{(\mu)} - f\left(W^T x^{(\mu)} + \theta_{\text{enc}}\right) \right\|_2^2 \text{ for } \mu \in \{1, \ldots, M\}.$$

Table 5.2: Dependency matrix of the individual error terms with respect to variable quantities. For example, E_R depends on W and H, but not on θ_{enc}, W_{out} or θ_{out}.

	H	W	θ_{enc}	W_{out}	θ_{out}
E_R	•	•			
E_I	•	•	•		
E_C		•	•	•	•

The discrepancy between the teacher signals and their approximations is measured by the *classification error*:

$$E_C := \sum_{\mu=1}^{M} E_C^{(\mu)}, \text{ where } E_C^{(\mu)} := \left\| t^{(\mu)} - g\left(W_{\text{out}}^T \tilde{h}^{(\mu)} + \theta_{\text{out}}\right) \right\|_2^2 \text{ for } \mu \in \{1, \ldots, M\}.$$

The aim of the remainder of this section is to deduce a gradient-based optimization algorithm for the overall error function E_{SCFC}. Table 5.2 shows which quantities the individual error functions are dependent on. Because E_{SCFC} is linear in these error terms, it is sufficient to consider the gradients for E_R, E_I and E_C separately. With the same argument, analysis can be restricted to individual samples, as the individual errors are sums of errors of the samples. Further, it is evident that from $E_R = E_{\text{NMF}}$ follows that the gradients of E_R with respect to W and H are as given in Theorem 46. As E_R does not depend on any other quantity, it suffices to derive the gradients of E_I and E_C.

The next result presents formulas for these gradients.

Theorem 47. Let $x \in \mathbb{R}^d$ be an individual sample with associated teacher signal $t \in \mathbb{R}^c$ and latent code word $h \in \mathbb{R}^n$. Let the quantities $u, \tilde{h} \in \mathbb{R}^n$ and $v, y \in \mathbb{R}^c$ be given as depicted in Figure 5.1. Let $F_I := \left\| h - \tilde{h} \right\|_2^2$ be the inference error and $F_C := \| t - y \|_2^2$ be the classification error for the sample x. Then the following holds:

(a) The gradient of the inference error F_I is given by

$$\partial F_I / \partial \theta_{\text{enc}} = -2\,(h - f(u))^T f'(u) \in \mathbb{R}^{1 \times n}, \quad (\partial F_I / \partial W)^T = x \cdot (\partial F_I / \partial \theta_{\text{enc}}) \in \mathbb{R}^{d \times n},$$

$$\text{and } \partial F_I / \partial h = 2\,(h - f(u))^T \in \mathbb{R}^{1 \times n}.$$

(b) The gradient of the classification error F_C is given by

$$\partial F_C / \partial \theta_{\text{out}} = -2\,(t - g(v))^T g'(v) \in \mathbb{R}^{1 \times c}, \quad (\partial F_C / \partial W_{\text{out}})^T = \tilde{h} \cdot (\partial F_C / \partial \theta_{\text{out}}) \in \mathbb{R}^{n \times c},$$

$$\partial F_C / \partial \theta_{\text{enc}} = -2 f'(u)^T W_{\text{out}} g'(v)^T (t - g(v)) \in \mathbb{R}^{1 \times n},$$

$$\text{and } (\partial F_C / \partial W)^T = x \cdot (\partial F_C / \partial \theta_{\text{enc}}) \in \mathbb{R}^{d \times n}.$$

Proof. (a) Application of the chain rule yields

$$\frac{\partial F_I}{\partial \theta_{\text{enc}}} = \frac{\partial \|h - f(u)\|_2^2}{\partial (h - f(u))} \cdot \frac{\partial (h - f(u))}{\partial u} \cdot \frac{\partial (W^T x + \theta_{\text{enc}})}{\partial \theta_{\text{enc}}} = -2(h - f(u))^T f'(u) \in \mathbb{R}^{1 \times n}.$$

Analogously,

$$\begin{aligned}
\frac{\partial F_I}{\partial \operatorname{vec}(W^T)} &= \frac{\partial \|h - f(u)\|_2^2}{\partial (h - f(u))} \cdot \frac{\partial (h - f(u))}{\partial u} \cdot \frac{\partial (W^T x + \theta_{\text{enc}})}{\partial \operatorname{vec}(W^T)} \\
&= -2(h - f(u))^T f'(u) \cdot \frac{\partial (x^T \otimes E_n) \operatorname{vec}(W^T)}{\partial \operatorname{vec}(W^T)} \\
&= -2(h - f(u))^T f'(u)(x^T \otimes E_n) \\
&= -2\left[(x \otimes E_n) \operatorname{vec}\left(f'(u)^T (h - f(u))\right)\right]^T \\
&= -2 \operatorname{vec}\left(f'(u)^T (h - f(u)) x^T\right)^T \in \mathbb{R}^{1 \times dn},
\end{aligned}$$

and hence with Theorem 67 from Appendix B follows

$$\left(\partial F_I / \partial W\right)^T = -2x(h - f(u))^T f'(u) = x \cdot \left(\partial F_I / \partial \theta_{\text{enc}}\right) \in \mathbb{R}^{d \times n}.$$

Eventually,

$$\frac{\partial F_I}{\partial h} = \frac{\partial \|h - f(u)\|_2^2}{\partial (h - f(u))} \cdot \frac{\partial (h - f(u))}{\partial h} = 2(h - f(u))^T \in \mathbb{R}^{1 \times n}.$$

(b) $\partial F_C / \partial \theta_{\text{out}}$ and $\partial F_C / \partial W_{\text{out}}$ follow exactly as in (a). The gradients for quantities from the inference layer follow from the chain rule:

$$\begin{aligned}
\frac{\partial F_C}{\partial \theta_{\text{enc}}} &= \frac{\partial \|t - g(v)\|_2^2}{\partial (t - g(v))} \cdot \frac{\partial (t - g(v))}{\partial v} \cdot \frac{\partial W_{\text{out}}^T \tilde{h} + \theta_{\text{out}}}{\partial \tilde{h}} \cdot \frac{\partial f(u)}{\partial u} \cdot \frac{\partial W^T x + \theta_{\text{enc}}}{\partial \theta_{\text{enc}}} \\
&= -2(t - g(v))^T g'(v) W_{\text{out}}^T f'(u) \in \mathbb{R}^{1 \times n}.
\end{aligned}$$

Then, analogously to (a) one obtains

$$\partial F_C / \partial \operatorname{vec}(W^T) = -2 \operatorname{vec}\left(f'(u)^T \cdot W_{\text{out}} \cdot g'(v)^T \cdot (t - g(v)) \cdot x^T\right)^T \in \mathbb{R}^{1 \times dn},$$

and the claim follows with Theorem 67. □

5.2.2 Optimization for Local Transfer Functions

A transfer function is called local when it is an entry-wise extension of a function $\mathbb{R} \to \mathbb{R}$ to a vector-valued function $\mathbb{R}^n \to \mathbb{R}^n$. Appendix C.5 provides a rigorous definition for this property. When the transfer functions $f\colon \mathbb{R}^n \to \mathbb{R}^n$ and $g\colon \mathbb{R}^c \to \mathbb{R}^c$ of the SCFC model are local, then gradient computation can be optimized using matrix notation. In this case, the inference error and the classification error for the entire data set can be rewritten as follows:

$$E_I = \left\|H - f\left(W^T X + \theta_{\text{enc}} \cdot J_{1\times M}\right)\right\|_F^2 \quad \text{and} \quad E_C = \left\|T - g\left(W_{\text{out}}^T \tilde{H} + \theta_{\text{out}} \cdot J_{1\times M}\right)\right\|_F^2.$$

Here, $J_{1\times M}$ is used for repeating the threshold vectors θ_{enc} and θ_{out} over all M samples, and f and g are applied element-wise on matrices from $\mathbb{R}^{n\times M}$ and $\mathbb{R}^{c\times M}$, respectively. Let

$$U := W^T X + \theta_{\text{enc}} \cdot J_{1\times M} \in \mathbb{R}^{n\times M} \text{ and } \tilde{H} := f(U) \in \mathbb{R}^{n\times M},$$
$$V := W_{\text{out}}^T \tilde{H} + \theta_{\text{out}} \cdot J_{1\times M} \in \mathbb{R}^{c\times M} \text{ and } Y := g(V) \in \mathbb{R}^{c\times M},$$

then $E_I = \left\|H - \tilde{H}\right\|_F^2$ and $E_C = \|T - Y\|_F^2$. This notation makes the computation of the gradients of E_I and E_C particularly efficient:

Theorem 48. When f and g are local transfer functions, then the gradients of the SCFC architecture can be simplified as follows:

(a) When f is local, then

$$\left({}^{\partial E_I}/_{\partial \theta_{\text{enc}}}\right)^T = -2G_I \cdot J_{M\times 1} \in \mathbb{R}^n,\ \left({}^{\partial E_I}/_{\partial W}\right)^T = -2X \cdot G_I^T \in \mathbb{R}^{d\times n},$$
$$\text{and } \left({}^{\partial E_I}/_{\partial H}\right)^T = 2(H - f(U)) \in \mathbb{R}^{n\times M}.$$

Here, the matrix G_I is defined as $G_I := f'(U) \circ (H - f(U)) \in \mathbb{R}^{n\times M}$. $f(U)$ and $f'(U)$ denote entry-wise evaluation of f and f', respectively. $\circ$ denotes the Hadamard product.

(b) When g is local, then

$$\left({}^{\partial E_C}/_{\partial \theta_{\text{out}}}\right)^T = -2G_C \cdot J_{M\times 1} \in \mathbb{R}^c \quad \text{and} \quad \left({}^{\partial E_C}/_{\partial W_{\text{out}}}\right)^T = -2\tilde{H} \cdot G_C^T \in \mathbb{R}^{n\times c},$$

with $G_C := g'(V) \circ (T - g(V)) \in \mathbb{R}^{c\times M}$. If additionally f is local, then

$$\left({}^{\partial E_C}/_{\partial \theta_{\text{enc}}}\right)^T = -2G_{IC} \cdot J_{M\times 1} \in \mathbb{R}^n \quad \text{and} \quad \left({}^{\partial E_C}/_{\partial W}\right)^T = -2X \cdot G_{IC}^T \in \mathbb{R}^{d\times n},$$

where $G_{IC} := f'(U) \circ (W_{\text{out}} G_C) \in \mathbb{R}^{n\times M}$.

Proof. (a) With the chain rule, and Lemma 60 and Lemma 69 from Appendix B follows

$$\begin{aligned}\frac{\partial E_I}{\partial \theta_{\text{enc}}} &= \frac{\partial \left\|\text{vec}(H - \tilde{H})\right\|_2^2}{\partial \text{vec}(H - \tilde{H})} \cdot \frac{\partial \text{vec}(H - f(U))}{\partial \text{vec}(U)} \cdot \frac{\partial \text{vec}\left(W^T X + \theta_{\text{enc}} \cdot J_{1\times M}\right)}{\partial \theta_{\text{enc}}} \\ &= -2 \,\text{vec}(H - \tilde{H})^T \cdot \text{diag}(\text{vec}(f'(U))) \cdot (J_{M\times 1} \otimes E_n) \\ &= -2\left[(J_{1\times M} \otimes E_n) \cdot \text{diag}(\text{vec}(f'(U))) \cdot \text{vec}(H - \tilde{H})\right]^T \\ &= -2\left[(J_{1\times M} \otimes E_n) \cdot \text{vec}(f'(U) \circ (H - \tilde{H}))\right]^T \\ &= -2\left[(f'(U) \circ (H - \tilde{H})) \cdot J_{M\times 1}\right]^T \in \mathbb{R}^{1\times n}.\end{aligned}$$

Completely analogously but with $X^T \otimes E_n$ instead of $J_{M\times 1} \otimes E_n$ follows

$$\partial E_I / \partial \operatorname{vec}(W^T) = -2 \operatorname{vec}\left[(f'(U) \circ (H - \tilde{H})) \cdot X^T \right]^T \in \mathbb{R}^{1\times nd},$$

and the gradient for W is yielded by applying Theorem 67. The gradient for H follows easily from Theorem 67.

(b) $\partial E_C / \partial \theta_{\text{out}}$ and $\partial E_C / \partial W_{\text{out}}$ are deduced exactly as in (a). In a similar manner follows

$$\begin{aligned}
\frac{\partial E_C}{\partial \theta_{\text{enc}}} &= \frac{\partial \|\operatorname{vec}(T-Y)\|_2^2}{\partial \operatorname{vec}(T-Y)} \frac{-\partial \operatorname{vec}(g(V))}{\partial \operatorname{vec}(V)} \frac{\partial \operatorname{vec}\left(W_{\text{out}}^T \tilde{H} E_M\right)}{\partial \operatorname{vec}(\tilde{H})} \frac{\partial \operatorname{vec}(f(U))}{\partial \operatorname{vec}(U)} \frac{\partial \operatorname{vec}\left(E_n \theta_{\text{enc}} J_{1\times M}\right)}{\partial \theta_{\text{enc}}} \\
&= -2 \operatorname{vec}(T-Y)^T \cdot \operatorname{diag}(\operatorname{vec}(g'(V))) \cdot (E_M \otimes W_{\text{out}}^T) \cdot \operatorname{diag}(\operatorname{vec}(f'(U))) \cdot (J_{M\times 1} \otimes E_n) \\
&= -2 \left[(J_{1\times M} \otimes E_n) \cdot \operatorname{diag}(\operatorname{vec}(f'(U))) \cdot (E_M \otimes W_{\text{out}}) \cdot \operatorname{vec}(G_C) \right]^T \\
&= -2 \left[(J_{1\times M} \otimes E_n) \cdot \operatorname{diag}(\operatorname{vec}(f'(U))) \cdot \operatorname{vec}(W_{\text{out}} G_C E_M) \right]^T \\
&= -2 \left[(J_{1\times M} \otimes E_n) \cdot \operatorname{vec}(G_{IC}) \right]^T = -2 \left(G_{IC} J_{M\times 1} \right)^T \in \mathbb{R}^{1\times n}.
\end{aligned}$$

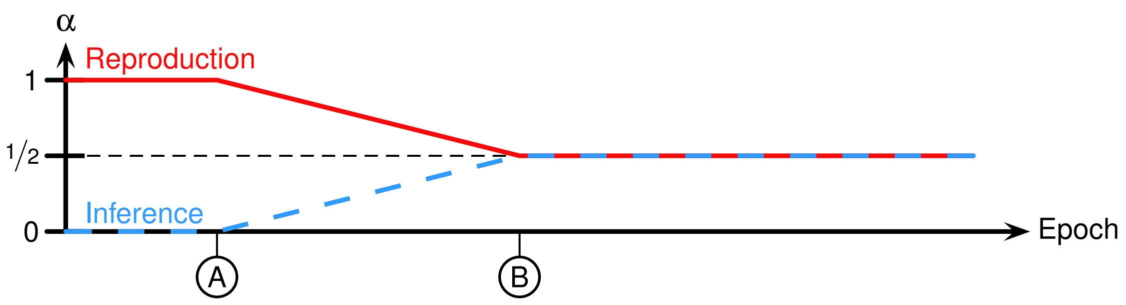
Like in (a), $\partial E_C / \partial W$ can be computed by replacing $J_{M\times 1} \otimes E_n$ with $X^T \otimes E_n$ in the equation above. □

5.2.3 Optimization Strategy

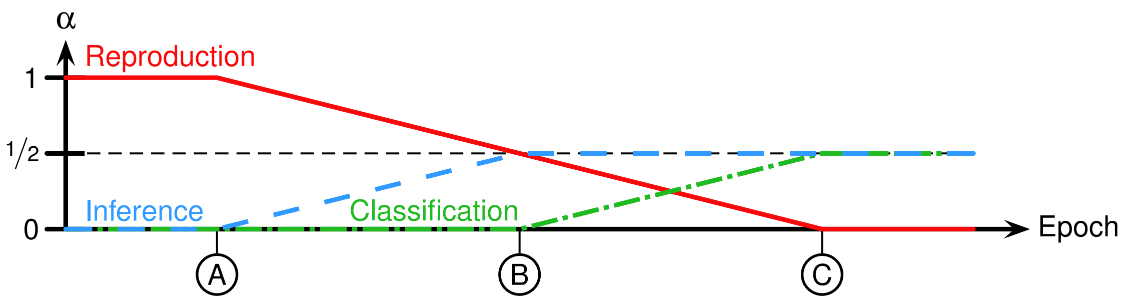
The proposed optimization strategy for minimization of SCFC's objective function E_{SCFC} is an alternating gradient descent procedure. The matrix of bases W and the matrix of code words H are randomly initialized and fed through sparseness projections as in Algorithm 5.1, but here no absolute value is taken since SCFC does not require non-negativity. The other variables are initialized randomly. During the training process, the degrees of freedom W, θ_{enc}, H, W_{out} and θ_{out} are tuned one after another. The step sizes for the individual gradient descents are chosen automatically using the Bold Driver heuristic. Application of the sparseness-enforcing projection operators Π_W and Π_H to W and H, respectively, guarantees the sparseness constraints are met as loop-invariant.

In contrast to similar optimization techniques, the trade-off constants α_I and α_C are not fixed throughout the optimization process. The concrete values for the trade-off variables are given by a function $\alpha\colon \mathbb{N} \to \mathbb{R} \times \mathbb{R}$, $\nu \mapsto (\alpha_I, \alpha_C)$. Here, ν denotes the number of the current *epoch* which denotes one pass through to entire learning set to update all parameters.

A sketch of the strategy for defining α is shown in Figure 5.2. When there is no ground truth available with the investigated data set, only the unsupervised operating mode can be employed where $\alpha_C \equiv 0$ throughout the entire process. In the first phase, only reproduction is optimized for, that is $\alpha_I = 0$ and hence $E_{\text{SCFC}} = E_{\text{NMF}}$. After rudimentary reproduction capabilities are achieved, the second phase starts and α_I is gradually increased to reach its



(a) Unsupervised operating mode.



(b) Supervised operating mode.

Figure 5.2: Strategy for choosing trade-off parameters α_I and α_C during the optimization process for two different operating modes. In the unsupervised operating mode, ground truth signals are either not available or ignored. From the beginning until A, only reproduction is optimized for. Then inference capabilities are slowly enforced until an intermediate level at B is reached. In the supervised operating mode, a third phase is added, in which reproduction is traded for classification capabilities until C is reached. From there on, reproduction is completely neglected to achieve the best possible classification performance.

maximum of $1/2$. Therefore, the impact of E_I on the overall error E_{SCFC} grows stronger slowly with each epoch. This is motivated by the phenomenon that for small values of α_I good reproduction capabilities can be obtained quickly, while inference capabilities converge only slowly to an optimum. In the remainder of the optimization, $E_{\text{SCFC}} = (E_R+E_I)/2$ holds until convergence.

When teacher signals are available, a classifier can be trained using the supervised operating mode. It is identical to the unsupervised operating mode until α_I reaches its maximum. From this point on, α_C is gradually increased until $\alpha_I + \alpha_C = 1$. Thus, E_R has no impact on E_{SCFC} in the end, and reproduction capabilities are neglected completely in order to achieve the best possible classification performance. If both reproduction and classification capabilities were required, the optimization would get stuck in a poor local minimum where neither of the two would work in a satisfactory manner.

5.2.4 Transfer Functions for Sparse Code Word Inference

Special attention has to be paid to the transfer function f used for code word inference. If f is chosen as hyperbolic tangent, then only poor inference capabilities can result. A better choice is the exponentiated hyperbolic tangent transfer function TanHypPot as defined in Section C.5, where the exponent is odd and greater or equal to three. With this function, a small plateau with very small slope emerges in a neighborhood of zero, mimicking the effect of applying a soft-shrinkage function after an ordinary hyperbolic tangent. This results in a depression of activities with small magnitude, favoring sparseness of the transfer function evaluation result. Exponentiation has the advantage over the soft-shrinkage function that a differentiable transfer function results as needed for efficient optimization via gradient descent.

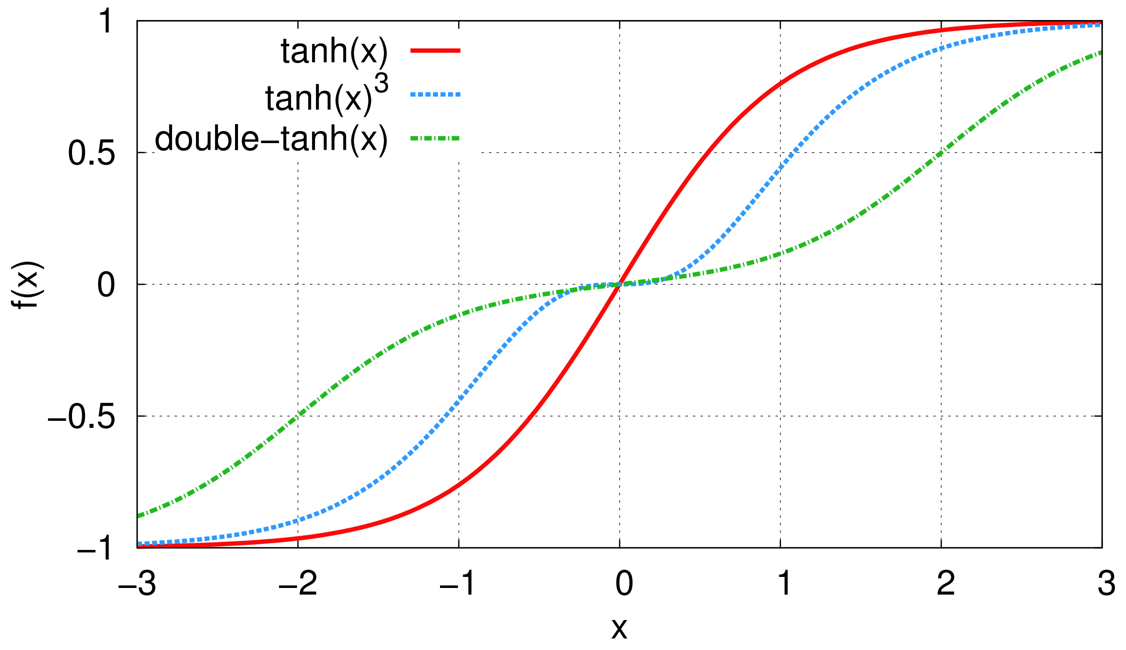
A similar transfer function has been proposed by [93]. The double hyperbolic tangent is given by double-tanh: $\mathbb{R} \to \mathbb{R}, x \mapsto {}^1\!/_2 \cdot (\tanh(x+2) + \tanh(x-2))$. This function features a lowered activity around zero, but its gradient only vanishes asymptotically and not in $x = 0$. This would be beneficial for sparse coding as units without activity would then not gain any more activity with gradient-based learning and a stable state could be achieved.

Plots of these functions are depicted in Figure 5.3a, and plots of related rectification functions are given in Figure 5.3b. These functions are inspired by the leaky integrate-and-fire (LIF) neuron model, and using them enforces zero activations for sparse representations [94]. The rectifier transfer function is given by rectifier: $\mathbb{R} \to \mathbb{R},\ x \mapsto \max(0,\ x)$. It enforces vanishing activity for negative input values. A smoother variant is the softplus transfer function, softplus: $\mathbb{R} \to \mathbb{R}, x \mapsto \log(1 + \exp(x))$, which however never attains zero values [94].

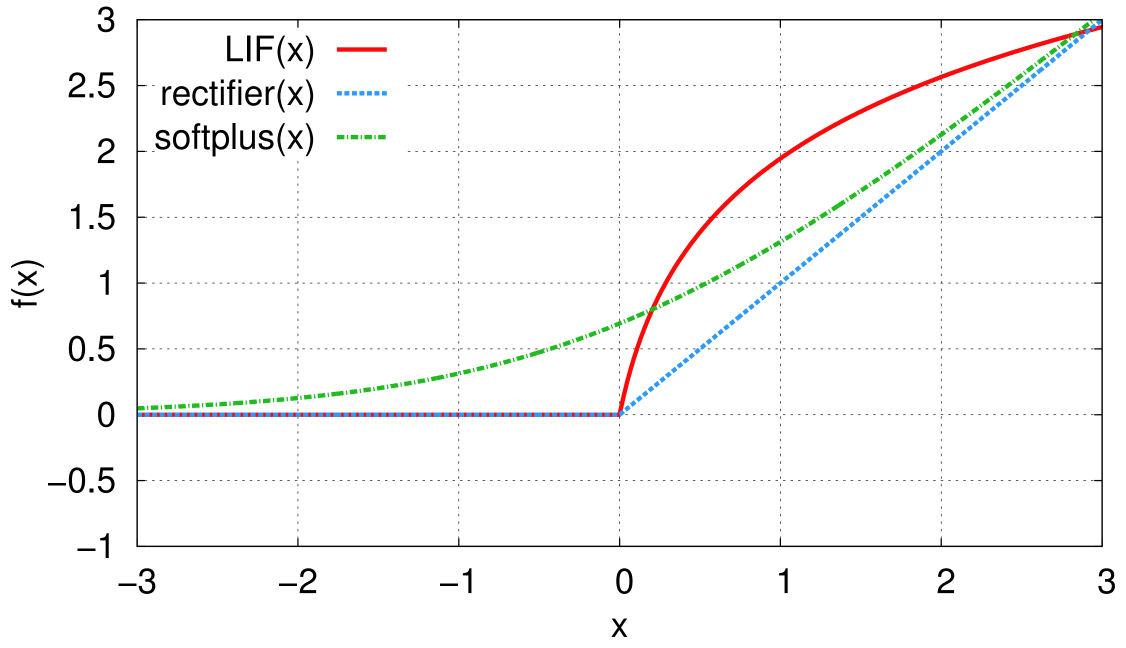
Note that all these transfer functions and hence the information processing is purely local. This means that the transfer functions are evaluated entrywise on a vector, and thus no information is interchanged between individual entries. A model that inherently uses non-local transfer functions for sparse code word inference is proposed and discussed in Chapter 7.

5.2.5 Space Complexity

Because the SCFC training procedure is a batch algorithm, it needs to hold all samples of the entire learning set and the corresponding intermediate quantities in main memory for efficient processing. An analysis of the space complexity of the proposed algorithm is here carried out to estimate its memory requirements. As the size of the learning set M is usually much larger than the occurring dimensionalities, only terms that depend on M are considered and other terms that account for only small quantities are neglected. This results in approximately $S := 2dM + 2cM + 4nM$ scalar values that have to be stored simultaneously. The terms $2dM$ and $2cM$ represent the data matrix X and the corresponding teacher signals T, respectively, plus matrices involved for evaluation of the error functions E_R and E_C. The matrix of code words H plays a central role. Two instances are needed for evaluation of the inference error



(a) Sigmoid transfer functions. The exponentiated hyperbolic tangent and the double hyperbolic tangent have low activity in a neighborhood of zero, while the standard hyperbolic tangent is almost linear there.



(b) Rectification transfer functions. The leaky integrate-and-fire (LIF) transfer function is biologically inspired. The rectifier and the softplus transfer functions have been employed for supervised learning.

Figure 5.3: An overview of transfer functions used for sparse representations.

Table 5.3: Memory requirements in gigabyte of SCFC for various numbers of learning samples M and numbers of hidden units n. Input and output dimensionality were chosen to match that of the MNIST data set of handwritten digits.

M \ n	32	64	128	256	512	1024	2048
60 000	0.767	0.824	0.939	1.17	1.62	2.54	4.37
540 000	6.90	7.42	8.45	10.5	14.6	22.9	39.3
13 500 000	173	185	211	263	366	572	984

E_I, and two additional instances are required for the computations of the gradients and an efficient implementation of the Bold Driver algorithm, such that multiple gradient evaluation is prevented.

It is obvious that S scales only linearly with the number of samples M. The constants occurring in S, however, depend on the input, hidden, and output dimensionality, respectively. As an example consider the MNIST database of handwritten digits as introduced in Chapter 3. Here, it is $M = 60\,000$, $d = 784$ and $c = 10$. Table 5.3 gives an estimate of memory requirements when double precision floating point numbers with 8 byte per scalar value are employed, in dependence on the number of hidden units n. In addition, memory requirements for data sets augmented with virtual samples as discussed in Chapter 3 are given. While the requirements for the original data set with 60 000 samples are quite conservative, boosting the data set size results in excessive memory requirements. Therefore, application of SCFC is restricted to situations where the number of learning samples is manageable.

5.2.6 Parallelization

An advantage of certain batch algorithms is that they can easily be parallelized because they process the entire learning set simultaneously. If two or more processors are available for performing computations, they can be employed to work on independent subproblems, which decreases overall running time. Software libraries that are specialized for multi-processor computations can be used to reduce implementation efforts. As the SCFC training algorithm makes extensive use of numerical linear algebra operations, the Basic Linear Algebra Subprograms (BLAS) [74] are employed in a high-performance variant [75, 76].

Apart from that, there are other portions of the training algorithm that can easily be parallelized using the OpenMP framework [95]. This is because most of the time operations are carried out on datums that are independent of each other, for example, transfer function evaluations on an entire matrix, or computation of the sparseness-enforcing projection operator on the individual rows or columns of a matrix.

Table 5.4: Speed-up $S(N)$ of SCFC when N processors are used compared to when only one processor is used as projected by Amdahl's law.

N	1	2	4	8	16	32	64	512	4096	∞
$S(N)$	1	2	4	7.8	15.1	28.5	51.1	168	236	250

To determine the theoretical maximum speed-up by parallelization, the effective degree of parallelism was estimated as follows. The algorithm is run until convergence, measuring both the elapsed real time R and the elapsed processor time P. On a uniprocessor system, both values are about equal. There is only a slight discrepancy due to context changes by the operating system. If $N > 1$ processors are employed and fully occupied by the algorithm, then the number of processors times the elapsed real time equals the processor time, $P = N \cdot R$. The parallel efficacy E can then be defined as the ratio $E := {}^{P}\!/\!_{N \cdot R} \in [0,\ 1]$. If an algorithm contains operations that cannot be parallelized, then E is significantly less than 1.

The measurements were carried out on a variety of systems with $N \in \{2,\ 4,\ 8,\ 12,\ 32\}$ and resulted in a mean efficacy of $E = 99.6\%$. Hence almost the entire algorithm can be implemented in parallel. Using Amdahl's law [96], a projection of the maximum possible speed-up S when N processors are employed can be computed with the relation $S(N) = (1 - E + {}^{E}\!/\!_{N})^{-1}$. Sample values of S for the efficacy E as determined above are given in Table 5.4 for a variety of the number of processors employed. For small N, the speed-up is almost linear in the number of processors. When N increases, speed-ups increase only sublinearly, and asymptotically saturate at 250. This means that it is impossible to accelerate execution beyond this point, unless the algorithm itself is made more efficient such that E gets closer to 100%.

5.3 Experiments

The performance of SCFC was evaluated on two data sets. The first data set consisted of natural image patches, similar to the data described in Section 2.1. The second data set was the MNIST database of handwritten digits as introduced in Chapter 3.

The architecture was fixed as follows. The transfer function for inference f was set to a hyperbolic tangent raised to the third power to facilitate inference of sparse code words, see Section 5.2.4. For the classification layer, the transfer function g was set to an ordinary hyperbolic tangent transfer function. Because both transfer functions were local by this choice, the optimizations from Section 5.2.2 could be employed. Since the Bold Driver heuristic is used for training, step sizes for gradient descent are adjusted automatically during parameter tuning. Training was finished when no significant progress could be achieved any more regarding the overall objective function E_{SCFC} on the learning set.

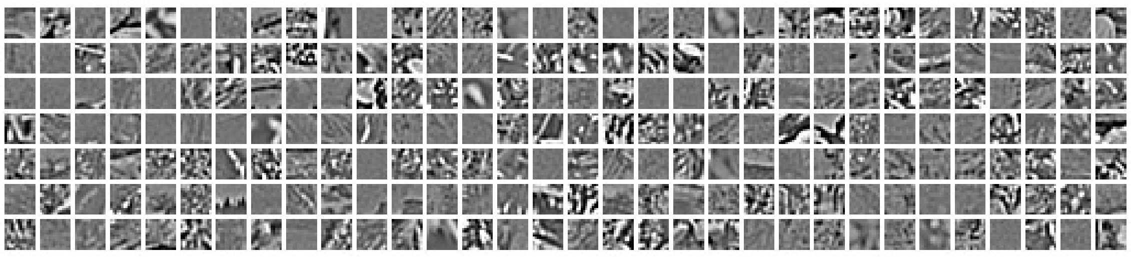
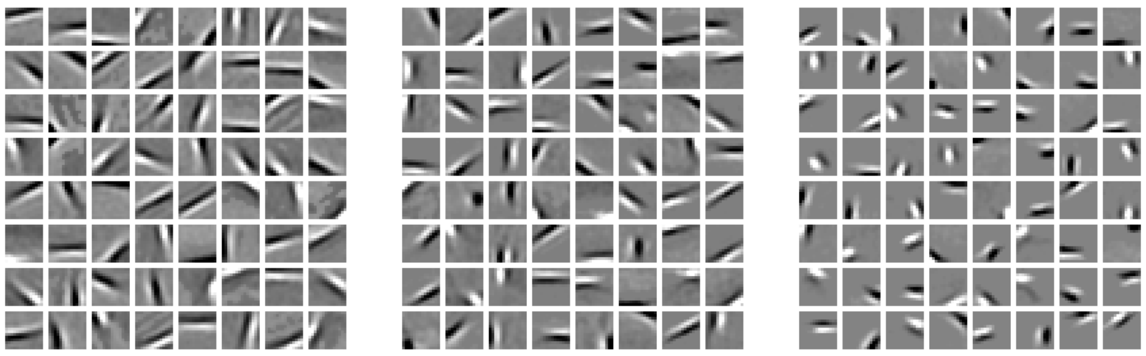


Figure 5.4: A random selection of natural image patches of size 14×14 pixels extracted from the whitened images of Olshausen and Field [10, 14].



(a) $\sigma_W = 0.35$. (b) $\sigma_W = 0.50$. (c) $\sigma_W = 0.65$.

Figure 5.5: Resulting filter matrices W of the unsupervised operating mode of SCFC applied to small patches of natural images. Parameters of the algorithm were $d := 196$, $M := 30\,000$, $n := 64$, and $\sigma_H := 0.85$, and three different values for σ_W as depicted beneath the images. The filters are instances of Gabor filters up to pixel-wise affine-linear transformations. Adapted from [79].

5.3.1 Natural Image Patches

To verify that SCFC was able to produce results similar to the experiments of Olshausen and Field [10, 97], a sparse representation based on natural image patches was computed. With the classical sparse coding models, this is known to yield Gabor-like filters, see Section 2.1. For the experiments described here, the original whitened images from [14] were used to extract $M := 30\,000$ natural image patches of size 14×14 pixels. Therefore, the input dimensionality d equals 196. A subset of samples from this data set is depicted in Figure 5.4. Because this data set did not contain class labels, SCFC was run in the unsupervised operating mode.

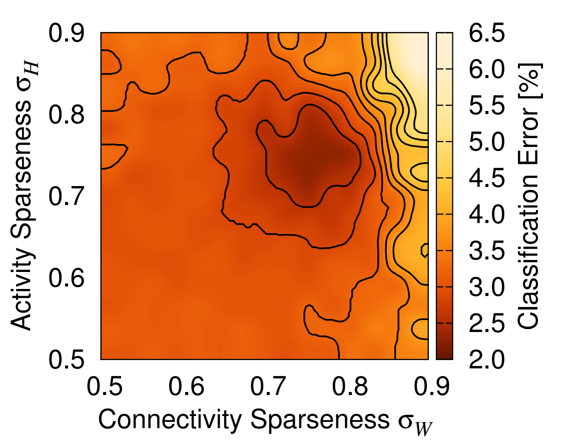
The filter matrices W learned for parameters $n := 64$ and $\sigma_H := 0.85$ are shown in Figure 5.5 for varying degrees of sparse connectivity σ_W. The filters shown are exact instances of Gabor filters up to pixel-wise affine-linear transformations. This can be seen by determining the Gabor hyperparameters that best match the sparse filters. These parameters were computed by maximizing the pixel-wise correlation coefficient ρ between a Gabor filter instance and a given filter from the corresponding column of W. Because of the non-smoothness of the mapping from the Gabor hyperparameter to the resulting quantized filter, the derivative-free optimization algorithm of Nelder and Mead [98] was used to solve this optimization problem. This yielded $\rho = 0.987 \pm 0.005$ for $\sigma_W = 0.65$, $\rho = 0.981 \pm 0.014$ for $\sigma_W = 0.50$ and $\rho = 0.972 \pm 0.019$ for $\sigma_W = 0.35$, where all correlation coefficients ρ denote mean $\pm$ standard deviation over all 64 filters. As all correlation coefficients are very close to unity, there is hardly any visual difference between the original filters and their Gabor filter approximations. The approximation works better for increased values of σ_W. This is because the random noise in W is then reduced and higher correlation coefficients can be achieved.

5.3.2 MNIST Database of Handwritten Digits

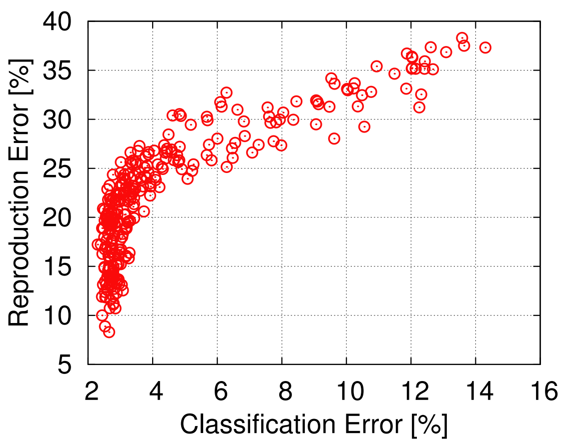
The memory requirement of SCFC scales linearly with the number of samples in the learning set. Current machines are not able to process the jittered or elastically distorted MNIST learning set within reasonable time due to memory constraints, see Table 5.3 for a detailed analysis. Additionally, convergence takes a large number of epochs for large learning sets due to the batch nature of the algorithm, as explained in Appendix C.2. Therefore in all experiments reported here, only the original MNIST learning set with $M = 60\,000$ samples was employed.

For the first experiments of SCFC on the MNIST data set, the sparseness degrees σ_W and σ_H were varied between 0.10 and 0.90 in steps of size 0.05, totaling in 289 combinations. In each run, $n := 192$ filters were computed in a supervised manner using the original MNIST learning set. The final classifiers were then used to compute the classification error on the evaluation set. The results of this experiment are depicted in Figure 5.6a. There is a small plateau, at approximately $(\sigma_W, \sigma_H) \in [0.7,\ 0.8]^2$, where an optimal classification error of 2.3% was achieved. The classification error is increasing with increasing distance from the plateau, and maximal errors are achieved for extreme values of σ_W and σ_H. This shows that sparseness degrees have to be chosen with care to prevent suboptimal classification performance.

The effect of the smooth unsupervised pre-training as induced by the optimization strategy described in Section 5.2.3 has also been analyzed. In doing so, the reproduction errors of all runs were recorded prior to the positive weighting of the classification error, that is at point B in Figure 5.2. A comparison of the reproduction errors and the ultimately achieved classification errors after training is given in Figure 5.6b. The trend is that best classification performance is only achieved for low reproduction errors. When the reproduction error was high, the classification error was high as well. This shows that unsupervised pre-training helps to precondition the optimization problem so that finding a good minimum with respect to the classification error is even possible, and hence justifies the proposed optimization strategy.



(a) Classification error on evaluation set dependent on sparseness degrees σ_W and σ_H, shown for combinations where $(\sigma_W, \sigma_H) \in [0.5,\ 0.9]^2$.



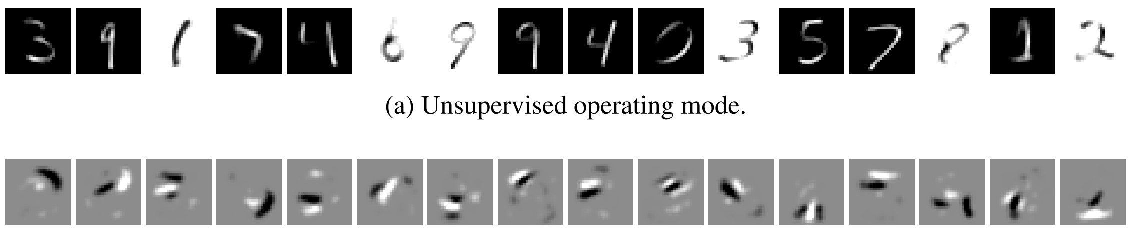
(b) Scatter plot of classification error and reproduction error, for all combinations of σ_W and σ_H.

Figure 5.6: Results of the experiments where 192 filters were computed with supervised SCFC on the MNIST learning set for all feasible combinations of the sparseness degree of activity σ_H and the sparseness degree of connectivity σ_W. Adapted from [79].

For the final experiments with the SCFC architecture, the number of hidden units was set to $n := 1000$ to achieve two times overcomplete sparse representations. Then a two-fold cross-validation was carried out to find a combination of σ_W and σ_H to use on the entire learning set. The cross-validation was repeated five times to compensate for the random initialization of the degrees of freedom. The range for the cross-validation was restricted to the plateau where $(\sigma_W, \sigma_H) \in [0.7,\ 0.8]^2$ as found earlier. This yielded ten validation errors for every combination, where the median of all ten values was processed further. The combination with $\sigma_W := 0.75$ and $\sigma_H := 0.80$ achieved the best median validation performance.

Using this combination of sparseness degrees, SCFC was run both in unsupervised and in supervised operating mode on the entire MNIST learning set. For both operating modes, the runs were carried out twenty times to compensate for random effects. To compute a classifier from the unsupervised run, ten linear SVM classifiers [99, 100] were trained in a one-vs.-all fashion on the inferred sparse code words. This yielded a classification error of $1.7\% \pm 0.10\%$ on the evaluation set. As SCFC produces a ready-to-use classifier in the supervised operating mode, no additional classifier had to be trained on the inferred code words for this variant. The classifier computed by SCFC achieved an evaluation error of $1.4\% \pm 0.11\%$, which is significantly better than the result from the unsupervised run.

This boost in performance can be explained by the incorporation of the teacher signals into the optimization of the filters. With this information the filters can be adapted to the class distribution of the samples, and thus they become *discriminative*. If the filters are only tuned such that an information-preserving representation is found, they are only *reproductive*. This



(a) Unsupervised operating mode.

(b) Supervised operating mode.

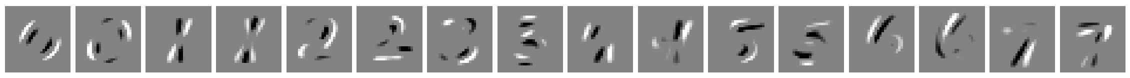
Figure 5.7: Subset of the filters from SCFC's (a) unsupervised and (b) supervised operating mode on the MNIST learning set, where parameters were $n := 1000$, $\sigma_W := 0.75$ and $\sigma_H := 0.80$. The entries of the filters from the unsupervised variant are either all non-negative or all non-positive, while the filters from the supervised variant feature both positive and negative values in adjacent regions and can thus be considered contrast detectors. Adapted from [79].

property does not guarantee that inferred code words are significantly different for samples of different classes, and hence these code words are suboptimal for subsequent classification. The difference between the two types of filter optimization becomes very clear by inspecting their morphological characteristics.

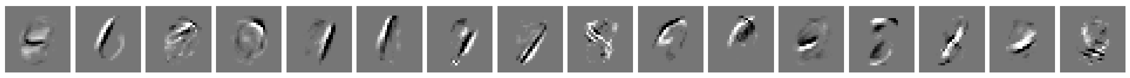
A subset of the filters from both operating modes is visualized in Figure 5.7. The filters from the unsupervised operating mode resemble samples from the learning set, and possess either only non-positive or only non-negative entries. Although the filters were initialized with random numbers from both the positive and negative domain and the unconstrained variant of the sparseness-enforcing projection operator was used, a minimum of the reproduction error is only achieved when all non-vanishing entries possess the same sign. The filters from the supervised operating mode are morphologically completely different. During optimization of the classification error they transform into local blobs, where regions of inverse sign are adjacent. The maximum filter response is hence achieved when large contrasts in the appropriate regions are present in the samples to be classified.

It can hence be concluded that the supervised operating mode on handwritten digits induces filters similar to those produced by the unsupervised operating mode applied to natural image patches. From this point of view the local contrast detectors seem to be biologically more plausible than the filters that resemble entire digits. The resemblance with the principal components of the individual classes, see Figure 3.7b, and the filters that induce independent components with low sparseness of activity, see Figure 3.8, is also notable. At first glance, the SCFC filters look like sparse versions of the higher principal components starting with the second principal component.

For comparison, the sparseness-enforcing projection operator was applied to the second and third principal components of the first eight classes of digits to yield their projections with a sparseness degree of 0.75. The result is depicted in Figure 5.8a. The sparsened principal



(a) Sparsified versions of the second and third principal component of the first eight classes. The contrast fields here are globally distributed over the entire digit range, and individual digits can still be recognized.



(b) Sparsified versions of the filters from Independent Component Analysis. The resulting contrast fields are local, but concentrated in the middle of the sample region rather than spatially well-distributed.

Figure 5.8: Filters from the unsupervised analysis in Chapter 3 of the MNIST data set fed through the sparseness-enforcing projection operator for a target sparseness degree of 0.75. These filters are compared with the results of SCFC, see Figure 5.7.

components feature contrast detectors, but are distributed over the entire input image. Hence they are not local and they still resemble entire digits, in contrast to the SCFC filters. The same processing was applied to the filters from Independent Component Analysis, see Figure 5.8b. Here, the contrast fields are clearly local, but they seem to be concentrated in the middle of the sample region. Also, vertical bars are predominant and the variability of the filters is rather low. The SCFC filters, on the other hand, are well-distributed over the entire sample region and feature a large variety of orientations. The SCFC filters are hence unique compared to what standard unsupervised learning algorithms produce.

5.4 Discussion

This chapter has analyzed Non-Negative Matrix Factorization with Sparseness Constraints in greater detail, and concluded that NMFSC is not suited inherently to real-time classification tasks because no procedure is known for efficient code word inference and because code words are not well-separable since class labels are not considered during optimization. A new model called Sparse Coding for Fast Classification was proposed in this chapter to address these drawbacks. In doing so, an inference path and a classification path were added to enrich the classical linear generative models. Further, an original transfer function was introduced to enable computation of sparsely populated code words. Finally, the space complexity and the possibility to parallelize the proposed batch learning algorithm were analyzed, and results on two data sets were reported.

It was demonstrated that, similar to classical sparse coding models, Gabor-like filters result when training on natural image patches. The application of SCFC to handwritten digits has shown that teacher signal incorporation helps to significantly improve generalization capabilities in classification tasks. Moreover, the morphological characteristics of the learned filters are completely different compared to when class labels are ignored. A comparison with the filters computed by standard unsupervised learning algorithms shows that the properties of the local contrast detectors found by SCFC are unique.

The SCFC architecture features significant advantages over previous approaches, as it is the first model that fulfills all criteria given in Section 2.4 except that the learning protocol is batch learning and not online learning. Unfortunately, this one missing property is the striking disadvantage of SCFC when processing very large data sets. The fact that a latent matrix of sparse code words has to be stored in main memory for all samples of the learning set has undesirable consequences, as then also numerous intermediate quantities such as the gradient of the objective function for that matrix have to be stored as well. As the memory requirements are linear, they may be acceptable from a complexity theorist's point of view. However, the constants the linear term is multiplied with are very large, such that only data sets with a small number of training examples can be handled on current computer systems.

This shortcoming is addressed in the following chapters. Chapter 6 proposes to ignore the sparse activity property and to concentrate on pure classification tasks. As an extension of these ideas, the sparse activity property and reproduction capabilities are re-incorporated in a new model in Chapter 7, where only the currently processed datum has to be stored in main memory. The key to this achievement are the theoretical results on the analytical properties of the sparseness-enforcing projection operator which were derived in Chapter 4.

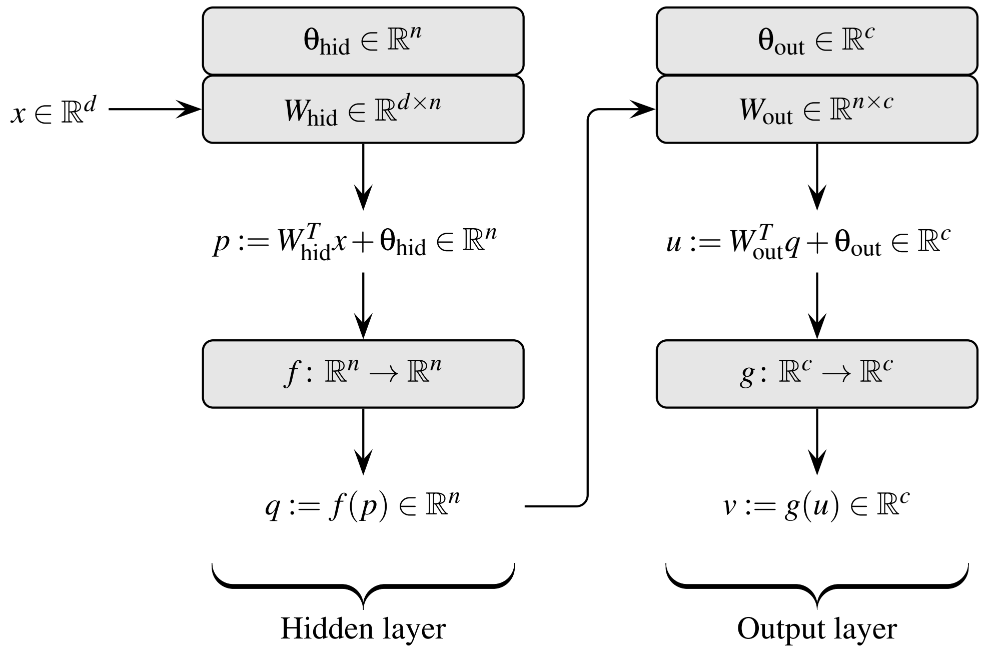
6 Sparsely Connected MLPs

This chapter proposes altering a conventional Multi-Layer Perceptron (MLP) so that the sparse connectivity property is fulfilled. This new model is denoted sparsely connected MLP or *Sparse MLP (SMLP)*. In this approach, sparse activity is neglected. It will, however, be featured in an extension of SMLPs proposed in Chapter 7.

Sparse MLPs inherit several advantages of conventional MLPs, such as the ability to handle huge data sets very efficiently. As generalization capabilities increase with the learning set size, excellent performance can be achieved with standard MLPs. Sparse connectivity additionally provides a convenient method to reduce the computational complexity of classification by one order of magnitude. This speed-up can be obtained by exploiting the sparse data flow due to the sparse connectivity. In other words, less data has to be processed with SMLPs to compute a classification decision. This effect is quantified in Chapter 8. It will be shown that in practice the restriction of only a low number of synaptic connections does not prevent SMLPs from achieving decent classification capabilities, and that learning time does not increase disproportionately.

Because two-layer MLPs are already universal approximators despite their architectural simplicity [90, 91, 92], analysis will be restricted here to two layers. It was shown recently that using more than two layers can result in an improved generalization performance [59]. This, however, has the severe drawback that a huge number of synaptic connections is required. For example, the largest network of [59] had twelve million synaptic connections. As a consequence, computational complexity of classification exceeds what can be tolerated for real-time capable systems. A detailed comparison of the computational complexity of SMLPs and these huge MLPs is included in Chapter 8.

The remainder of this chapter is organized as follows. First, a formal definition of sparsely connected Multi-Layer Perceptrons is given. In addition, simple formulas to compute the gradient of the corresponding objective function when considering multi-class problems is given. This also includes the information on the Hessian that is required for the Optimal Brain Damage (OBD) technique [50], which was introduced in Section 2.4.9. Finally, a three-staged procedure for the training of SMLPs is proposed. The experimental section reports results on the MNIST database of handwritten digits and compares them with standard MLPs and the OBD technique. The chapter concludes with a discussion of the obtained results. The material of this chapter is based on the publications [101] and [66].



$\theta_{\text{hid}} \in \mathbb{R}^n$
$x \in \mathbb{R}^d$
$W_{\text{hid}} \in \mathbb{R}^{d\times n}$
$p := W_{\text{hid}}^T x + \theta_{\text{hid}} \in \mathbb{R}^n$
$f\colon \mathbb{R}^n \to \mathbb{R}^n$
$q := f(p) \in \mathbb{R}^n$
Hidden layer
$\theta_{\text{out}} \in \mathbb{R}^c$
$W_{\text{out}} \in \mathbb{R}^{n\times c}$
$u := W_{\text{out}}^T q + \theta_{\text{out}} \in \mathbb{R}^c$
$g\colon \mathbb{R}^c \to \mathbb{R}^c$
$v := g(u) \in \mathbb{R}^c$
Output layer

Figure 6.1: Architecture of a two-layer MLP. The input sample x is fed through the hidden layer to yield the internal representation q. The procedure is repeated for the output layer, which yields the classification decision v. In an SMLP, the weight matrix of the hidden layer W_{hid} is restricted so to be sparsely populated.

6.1 Architecture

The architecture and dynamics of a two-layer MLP is shown in Figure 6.1. The first layer is called the *hidden layer*, it here has $n \in \mathbb{N}$ hidden units. The degrees of freedom of this layer are a weight matrix $W_{\text{hid}} \in \mathbb{R}^{d\times n}$ and a threshold vector $\theta_{\text{hid}} \in \mathbb{R}^n$. The parameters of the *output layer* are given by another weight matrix $W_{\text{out}} \in \mathbb{R}^{n\times c}$ and another threshold vector $\theta_{\text{out}} \in \mathbb{R}^c$. There are two transfer functions $f\colon \mathbb{R}^n \to \mathbb{R}^n$ and $g\colon \mathbb{R}^c \to \mathbb{R}^c$ for the hidden layer and output layer, respectively. See Section C.5 for an overview of commonly used transfer functions.

Given an input sample $x \in \mathbb{R}^d$ which is to be classified, the hidden layer computes an *internal representation* $q := f\left(W_{\text{hid}}^T x + \theta_{\text{hid}}\right) \in \mathbb{R}^n$ by multiplication with the weight matrix of the hidden layer, adding the corresponding threshold vector and applying the transfer function f. The process is then repeated by feeding the internal representation through the output layer to yield the classification decision $v := g\left(W_{\text{out}}^T q + \theta_{\text{out}}\right) \in \mathbb{R}^c$.

The network is adapted on a learning set by minimizing the deviation between the classification decision and the label of individual samples. In doing so, the degrees of freedom $\mathcal{W} := (W_{\text{hid}}, \theta_{\text{hid}}, W_{\text{out}}, \theta_{\text{out}})$ are tuned to optimize an objective function E_{SMLP}. This objective function is defined to be the empirical mean of a similarity measure $s\colon \mathbb{R}^c \times \mathbb{R}^c \to \mathbb{R}$ over the entire learning set, see Appendix C for a detailed discussion. The function s measures the deviation between the network output v given a fixed sample and the teacher signal corresponding to that sample. To incorporate sparse connectivity in an SMLP, feasible solutions are required to fulfill the following sparseness constraint:

$$\sigma(W_{\text{hid}} e_i) \overset{!}{=} \sigma_W \text{ for all } i \in \{1, \ldots, n\}.$$

Here, $\sigma_W \in (0,\ 1)$ is a parameter that controls the connectivity sparseness. This constraint is essentially the sparse connectivity constraint of Non-Negative Matrix Factorization with Sparseness Constraints [26] and Sparse Coding for Fast Classification, each neuron in the hidden layer shall receive only a limited amount of data from the input samples.

The neurons from the *output layer* are intentionally not restricted so to be sparsely connected. The final weighting of the MLP's internal representation is done by the output layer, and here all information can be considered relevant to be compressed into the classification decision. If the output layer was sparsely connected, a large part of the hidden units would not contribute to the final decision-making which would result in poor classification capabilities. Moreover, since the output dimensionality is usually much less than the input dimensionality, that is $c \ll d$ holds, W_{out} is much smaller than W_{hid} with respect to the number of entries. In the overall performance, restricting W_{out} to be sparsely populated provides a speed-up only negligible compared to what can be achieved when W_{hid} is restricted.

Projected gradient descent is the proposed method for minimizing E_{SMLP} subject to explicit sparseness constraints. The gradient information is collected in the following result:

Theorem 49. Let $x \in \mathbb{R}^d$ be a sample with corresponding teacher signal $t \in \mathbb{R}^c$. Moreover, let $s\colon \mathbb{R}^c \times \mathbb{R}^c \to \mathbb{R}$ be a similarity measure and define $F_{\text{SMLP}} := s(v,t)$ to be the sample-wise objective function. Let $s'(v,t) := {}^{\partial s}\!/\!_{\partial v} \in \mathbb{R}^{1\times c}$ denote the gradient of s with respect to v. Then:

(a) The gradients of the output layer are

$$\left({}^{\partial F_{\text{SMLP}}}\!/\!_{\partial \theta_{\text{out}}}\right)^T = g'(u)^T \cdot s'(v,t)^T \in \mathbb{R}^c \text{ and } \left({}^{\partial F_{\text{SMLP}}}\!/\!_{\partial W_{\text{out}}}\right)^T = q \cdot \left({}^{\partial F_{\text{SMLP}}}\!/\!_{\partial \theta_{\text{out}}}\right) \in \mathbb{R}^{n\times c}.$$

(b) The gradients of the hidden layer are

$$\left({}^{\partial F_{\text{SMLP}}}\!/\!_{\partial \theta_{\text{hid}}}\right)^T = f'(p)^T \cdot W_{\text{out}} \cdot \left({}^{\partial F_{\text{SMLP}}}\!/\!_{\partial \theta_{\text{out}}}\right) \in \mathbb{R}^n$$
$$\text{and } \left({}^{\partial F_{\text{SMLP}}}\!/\!_{\partial W_{\text{hid}}}\right)^T = x \cdot \left({}^{\partial F_{\text{SMLP}}}\!/\!_{\partial \theta_{\text{hid}}}\right) \in \mathbb{R}^{d\times n}.$$

Proof. Completely analogous to Theorem 47 from Chapter 5. □

Because the gradient of the output layer emerges in the gradient of the hidden layer, the gradient descent procedure is also known as *backpropagation* in this context [89]. Linearity makes the generalization to the empirical mean over the entire learning set straightforward.

6.2 Simplifications and Higher Derivatives

The MLPs as described previously are inherently suited for classification tasks where more than two classes have to be handled. Therefore, heuristics like one-vs.-one and one-vs.-all classification are superfluous in this framework. Further, all class information can be used simultaneously in the tuning of the network parameters, while in the heuristics only binary teacher signals are available. This additional information is considered by the learning algorithm in a natural way, so that superior classification performance can be achieved.

In the multi-class situation considered here, the teacher signals are given as one-of-c codes, and the transfer function of the output layer is set to the softmax transfer function. As this transfer function is non-local, class membership information is shared among all neurons in the output layer and from there backpropagated into the hidden layer. Hence all neurons in the hidden layer can be adapted accordingly. Further, the output vector of the softmax transfer function can directly be interpreted as confidence of the neural network [53].

As was observed by [53, 102], the softmax transfer function forms a natural pairing together with the cross entropy similarity measure. For classification problems, the cross entropy function is also preferable over the squared error similarity measure as better generalization capabilities can be achieved [58]. It is also this pairing that allows deriving compact expressions for the diagonal entries of the Hessian of the objective function, despite a non-local transfer function being used. This information is crucial for the implementation of the Optimal Brain Damage method, which can be considered an early precursor of the sparsely connected MLP proposed here. The next result gathers the necessary information for efficient implementation of both sparsely connected MLPs as proposed in this work and OBD:

Theorem 50. Consider a two layer MLP where the similarity measure is the cross entropy, the transfer function of the output layer is the softmax transfer function and all teacher signals are represented by one-of-c codes, that is $s = \text{CE}$, $g = \text{Softmax}$ and in particular $\sum_{i=1}^{c} t_i = 1$ for all t. Then the following holds in the situation of Theorem 49:

(a) The gradients of the output layer can be simplified to

$$\left({}^{\partial F_{\text{SMLP}}}/_{\partial\theta_{\text{out}}}\right)^T = v - t \in \mathbb{R}^c \text{ and } \left({}^{\partial F_{\text{SMLP}}}/_{\partial W_{\text{out}}}\right)^T = q \cdot (v-t)^T \in \mathbb{R}^{n\times c}.$$

(b) Let $H_{\text{out}} := {}^{\partial^2 F_{\text{SMLP}}}/_{\partial\, \text{vec}(W_{\text{out}})^2} \in \mathbb{R}^{nc\times nc}$ denote the Hessian of F_{SMLP} with respect to the vectorized weight matrix of the output layer. Then its main diagonal, reshaped to a matrix of equal size as of the weight matrix, is given by

$$H_{\text{out}}^D = (q \circ q)\cdot(v - v\circ v)^T \in \mathbb{R}^{n\times c}.$$

(c) Let $H_{\text{hid}} := {}^{\partial^2 F_{\text{SMLP}}}/_{\partial\, \text{vec}(W_{\text{hid}})^2} \in \mathbb{R}^{dn\times dn}$ denote the Hessian with respect to the vectorized weight matrix of the hidden layer. When the transfer function f of the hidden layer is local as extension of a function $\tilde{f}\colon \mathbb{R} \to \mathbb{R}$, then the reshaped main diagonal of the Hessian is given by

$$H^D_{\text{hid}} = (x \circ x) \cdot z^T \in \mathbb{R}^{d \times n}, \text{ where } z \in \mathbb{R}^n \text{ is given, using } h^{(i)} := W^T_{\text{out}} e_i \in \mathbb{R}^c, \text{ by}$$

$$z_i := \tilde{f}'(p_i)^2 \cdot \left(\langle h^{(i)} \circ h^{(i)},\ v \rangle - \langle h^{(i)},\ v \rangle^2 \right) + \tilde{f}''(p_i) \cdot \langle h^{(i)},\ v - t \rangle \in \mathbb{R} \text{ for } i \in \{1, \ldots, n\}.$$

Proof. This proof uses the notation and results from Theorem 49.

(a) When $s = \text{CE}$, then $s'(v,t) = -(t \oslash v)^T \in \mathbb{R}^{1 \times c}$ where $\oslash$ denotes the entry-wise quotient, see Lemma 78 in Appendix C. When $g = \text{Softmax}$, then $g'(u) = \text{diag}(v) - vv^T$ by Lemma 73. With Theorem 49, Lemma 60 from Appendix B and $\sum_{i=1}^c t_i = 1$ follows

$$\begin{aligned}
\left(\partial F_{\text{SMLP}}/\partial \theta_{\text{out}}\right)^T &= -\left(\text{diag}(v) - vv^T\right) \cdot (t \oslash v) = v \cdot \langle v,\ t \oslash v \rangle - v \circ t \oslash v \\
&= \left(\textstyle\sum_{i=1}^c v_i \cdot t_i/v_i\right) v - v \oslash v \circ t = v - t \in \mathbb{R}^c.
\end{aligned}$$

Application of Theorem 49 then yields the gradient for W_{out}.

(b) Using (a), the chain rule and Theorem 55 yields the full Hessian as

$$\begin{aligned}
H_{\text{out}} &= \frac{\partial \operatorname{vec}\left(E_c (v - t) q^T\right)}{\partial \operatorname{vec}\left(W^T_{\text{out}}\right)} = (q \otimes E_c) \cdot \frac{\partial g(u)}{\partial u} \cdot \frac{\partial \operatorname{vec}\left(E_c W^T_{\text{out}} q\right)}{\partial \operatorname{vec}\left(W^T_{\text{out}}\right)} \\
&= (q \otimes E_c) \cdot g'(u) \cdot \left(q^T \otimes E_c\right) \in \mathbb{R}^{nc \times nc}.
\end{aligned}$$

As diagonal elements of H_{out} suffice for the claim, consider $e_i^T H_{\text{out}} e_i$ for $i \in \{1, \ldots, nc\}$. With Euclidean division, there exist $a \in \mathbb{N}$ and $b \in \{0, \ldots, c-1\}$ such that $i - 1 = ac + b$. This division and the definition of the Kronecker product imply $\left(q^T \otimes E_c\right) e_i = q_{a+1} e_{b+1} \in \mathbb{R}^c$, which is a canonical basis vector scaled by an entry of q. With Lemma 56 follows $e_i^T (q \otimes E_c) = \left(\left(q^T \otimes E_c\right) e_i\right)^T = q_{a+1} e^T_{b+1} \in \mathbb{R}^{1 \times c}$. Therefore,

$$e_i^T H_{\text{out}} e_i = q^2_{a+1} e^T_{b+1} g'(u) e_{b+1} = q^2_{a+1} e^T_{b+1} \left(\text{diag}(v) - vv^T\right) e_{b+1} = q^2_{a+1} \left(v_{b+1} - v^2_{b+1}\right).$$

Now let $D := (q \circ q) \cdot (v - v \circ v)^T \in \mathbb{R}^{n \times c}$. The dimension of D is equal to the dimension of W_{out}, therefore it is sufficient to show that D contains the corresponding diagonal elements of the Hessian. For this, let $a \in \{1, \ldots, n\}$ and $b \in \{1, \ldots, c\}$ be row and column indices, respectively. Then $e_a^T D e_b = \left(e_a^T (q \circ q)\right) \cdot \left(\left(v^T - v^T \circ v^T\right) e_b\right) = q_a^2 \cdot \left(v_b - v_b^2\right)$, and the claim follows by comparison with the diagonal elements of H_{out} as computed above.

(c) Assume f is local as extension of $\tilde{f}\colon \mathbb{R} \to \mathbb{R}$, and let $\tilde{f}'(p) \in \mathbb{R}^n$ and $\tilde{f}''(p) \in \mathbb{R}^n$ denote its first and second derivative, respectively, applied entry-wise to p. Hence $f'(p) = \text{diag}\left(\tilde{f}'(p)\right)$. With (a), Theorem 49, the chain rule, the Hadamard product rule and Theorem 55 follows

$$\begin{aligned}
H_{\text{hid}} &= \frac{\partial \operatorname{vec}\left(E_n f'(p) W_{\text{out}} (v - t) x^T\right)}{\partial \operatorname{vec}\left(W^T_{\text{hid}}\right)} = (x \otimes E_n) \cdot \frac{\partial \tilde{f}'(p) \circ (W_{\text{out}}(v - t))}{\partial \operatorname{vec}\left(W^T_{\text{hid}}\right)} \\
&= (x \otimes E_n) \cdot \left[\text{diag}\left(\tilde{f}'(p)\right) \cdot \frac{\partial W_{\text{out}}(v - t)}{\partial \operatorname{vec}\left(W^T_{\text{hid}}\right)} + \text{diag}\left(W_{\text{out}}(v - t)\right) \cdot \frac{\partial \tilde{f}'(p)}{\partial \operatorname{vec}\left(W^T_{\text{hid}}\right)} \right].
\end{aligned}$$

The derivatives in the last expression are given as

$$\begin{aligned}\frac{\partial W_{\text{out}}(v-t)}{\partial \operatorname{vec}\left(W_{\text{hid}}^T\right)} &= W_{\text{out}} \cdot \frac{\partial g(u)}{\partial u} \cdot \frac{\partial W_{\text{out}}^T q}{\partial q} \cdot \frac{\partial f(p)}{\partial p} \cdot \frac{\partial \operatorname{vec}\left(E_n W_{\text{hid}}^T x\right)}{\partial \operatorname{vec}\left(W_{\text{hid}}^T\right)} \\ &= W_{\text{out}} \cdot g'(u) \cdot W_{\text{out}}^T \cdot f'(p) \cdot \left(x^T \otimes E_n\right),\end{aligned}$$

and

$$\frac{\partial \tilde{f}'(p)}{\partial \operatorname{vec}\left(W_{\text{hid}}^T\right)} = \frac{\partial \tilde{f}'(p)}{\partial p} \cdot \frac{\partial \operatorname{vec}\left(E_n W_{\text{hid}}^T x\right)}{\partial \operatorname{vec}\left(W_{\text{hid}}^T\right)} = \operatorname{diag}\left(\tilde{f}''(p)\right) \cdot \left(x^T \otimes E_n\right).$$

Therefore the Hessian is

$$H_{\text{hid}} = (x \otimes E_n) \cdot \left[f'(p) W_{\text{out}} g'(u) W_{\text{out}}^T f'(p) + \operatorname{diag}\left((W_{\text{out}}(v-t)) \circ \tilde{f}''(p)\right)\right] \cdot \left(x^T \otimes E_n\right).$$

Let $H_{\text{hid}}^{(1)} := f'(p) W_{\text{out}} g'(u) W_{\text{out}}^T f'(p) \in \mathbb{R}^{n \times n}$ and $H_{\text{hid}}^{(2)} := \operatorname{diag}\left((W_{\text{out}}(v-t)) \circ \tilde{f}''(p)\right) \in \mathbb{R}^{n \times n}$. Let $i \in \{1, \dots, c\}$ and define $h^{(i)} := W_{\text{out}}^T e_i \in \mathbb{R}^c$. Then

$$e_i^T H_{\text{hid}}^{(1)} e_i = \tilde{f}'(p_i)^2 \left(h^{(i)}\right)^T \left(\operatorname{diag}(v) - v v^T\right) h^{(i)} = \tilde{f}'(p_i)^2 \left(\left\langle h^{(i)} \circ h^{(i)},\ v\right\rangle - \left\langle h^{(i)},\ v\right\rangle^2\right) \in \mathbb{R}$$

and $e_i^T H_{\text{hid}}^{(2)} e_i = e_i^T \left(W_{\text{out}}(v-t)\right) \cdot \tilde{f}''(p_i) = \tilde{f}''(p_i) \cdot \left\langle h^{(i)},\ v-t\right\rangle \in \mathbb{R}$. Therefore, the vector $z \in \mathbb{R}^n$ from the claim consists exactly of the diagonal elements of $H_{\text{hid}}^{(1)} + H_{\text{hid}}^{(2)}$, and the claim follows exactly as in (b). □

Because the gradients of the output layer emerge in the expression of the gradients of the hidden layer, the results from Theorem 50(a) can be directly substituted into the expressions of Theorem 49(b) for further simplification.

6.3 Training Algorithm

This section proposes a three-staged training procedure for sparsely connected MLPs. First, the degrees of freedom are initialized by pre-training on a small subset of the learning set. Then, a modified version of the backpropagation algorithm is used for fine-tuning on the entire learning set. In a final post-processing stage, irrelevant synaptic connections from the hidden layer are pruned, and the weight matrix of the hidden layer is converted into a sparse data structure. An overview of the process is shown in Figure 6.2.

The training algorithm has the following parameters. The input dimensionality d and the output dimensionality c are determined from the corresponding dimensionalities of the learning set. The dimensionality of the internal representation n is the crucial parameter for the performance and the computational complexity of the resulting classifiers. Finally, the target sparseness degree of connectivity $\sigma_W \in (0,\ 1)$ has to be chosen which also has great impact on the computational complexity of the run-time of the classifier.

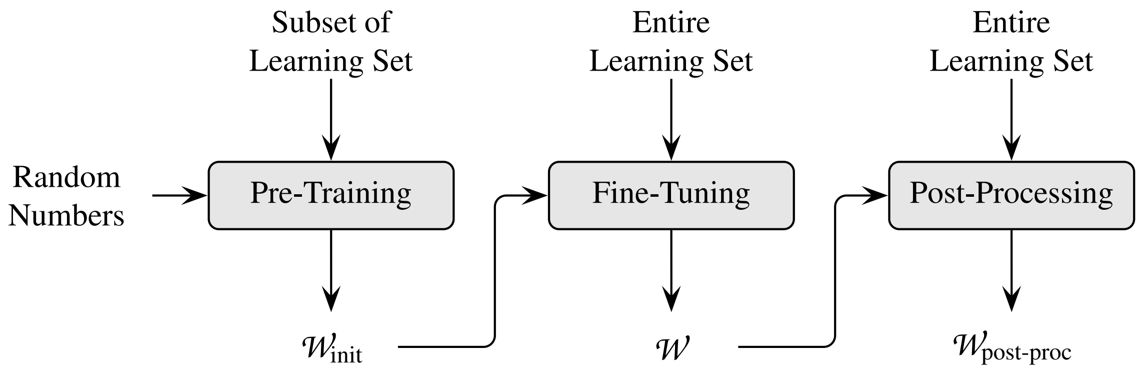


Figure 6.2: Overview of the procedure for training of sparsely connected MLPs. Three different instances of the degrees of freedom $\mathcal{W} = (W_{\text{hid}}, \theta_{\text{hid}}, W_{\text{out}}, \theta_{\text{out}})$ are computed during the three-staged process.

6.3.1 Pre-Training

A proper initialization prior to actual training is crucial for achieving good performance. When only random numbers were used, training would get stuck in a poor local minimum. This is because of the bad-natured optimization problem that has to be solved. Incorporation of sparseness constraints makes good minima difficult to find using simple gradient descent methods. Therefore, the first step is to determine a set of parameters that is already in the vicinity of a good minimum, such that gradient-based fine-tuning can finally find the way to an acceptable solution of the optimization problem.

Because of its close relationship with SMLPs, SCFC is the method of choice for pre-training. Additionally to sparse connectivity, SCFC also features sparse activity of individual hidden units over the entire learning set. This causes each hidden unit to cover a distinct region of the manifold the input samples lie on, and therefore breaks the symmetry among different hidden units. Because of the memory requirements of SCFC, it can only be run on a subset of the learning set. The size of this subset is chosen such that all intermediate variables still fit into main memory using the estimate from Section 5.2.5. The remaining parameters of SCFC are chosen as follows. The number of hidden units n and the target degree of sparse connectivity σ_W as chosen for SMLP training are simply handed over to SCFC. The target degree of sparse activity σ_H is set to an intermediate value, such that the unsupervised operating mode of SCFC can still find a representation that preserves most of the information of the samples input to SCFC. When the number of hidden units is large compared to the intrinsic dimensionality of the samples, then $\sigma_H := 0.85$ can be used as guideline. The transfer function for code word inference is set to a cubed hyperbolic tangent, and the transfer function for classification is set to a hyperbolic tangent such that the optimizations from Section 5.2.2 can be used.

SCFC is then run on a small subset of the learning set in its unsupervised operating mode until convergence. The degrees of freedom of SCFC are in fact initialized with randomly sampled numbers. Nevertheless, the special optimization strategy that first minimizes the reproduction error in conjunction with the sparseness projection applied to the code word matrix makes the approach mostly invariant to the actual random initialization. On convergence, an instance of degrees of freedom $\mathcal{W}_{\text{init}}$ is handed over to the fine-tuning phase. If the supervised operating mode of SCFC was used, learning would result in parameters that minimize the classification error only for the subset used, making it difficult to escape from when training on the entire learning set afterwards.

6.3.2 Fine-Tuning

The SMLP degrees of freedom are initialized from the SCFC results given by $\mathcal{W}_{\text{init}}$. The hidden layer's transfer function is set to the transfer function used for SCFC code word inference. The parameters of the output layer are initialized by training linear Support Vector Machines [100] in a one-vs.-all fashion. The transfer function of the output layer is set to the softmax transfer function, and the similarity measure from the SMLP objective function is set to the cross entropy function such that the optimizations from Theorem 50 can be used.

After these initializations, a projected gradient descent procedure is used to tune the network parameters on the entire learning set. In each epoch, a fixed number of samples is drawn randomly from the learning set and presented to the network. The network parameters are updated after each presented sample by using the online learning protocol.

The weight matrix of the hidden layer W_{hid} is sparsely populated after pre-training. When the original backpropagation algorithm is used, W_{hid} loses this property. This is because the sparseness constraints are a real restriction which is not automatically fulfilled when the objective function is minimized. To ensure that sparse connectivity persists during learning and hence also after learning, the sparseness-enforcing projection operator π is applied to the columns of W_{hid} to achieve the target sparseness degree σ_W. The projection is parameterized so to achieve normalized projections from the entire space $\mathbb{R}^d$, that is $\omega :=$ arbitrary and $\eta :=$ unit_norm. This guarantees sparse connectivity, and further that the network does not unlearn its performance, because the sparseness projection finds the closest possible sparsely populated weight matrix.

It is crucial that the application of the sparseness-enforcing projection operator is timed right. The correct timing is achieved by choosing an appropriate number of samples that are presented during each epoch. If too few samples are presented before the projection, then W_{hid} will not be modified effectively. In this case, the projection will just reset the weight matrix to the state before sample presentation. On the other hand, if too many samples are presented between projections, the changes to W_{hid} become too intense. This results in divergence, analogous to the situation when the step size is set too large. In practice, it is sufficient to carry out the projection after about 20 000 sample presentations to yield good results.

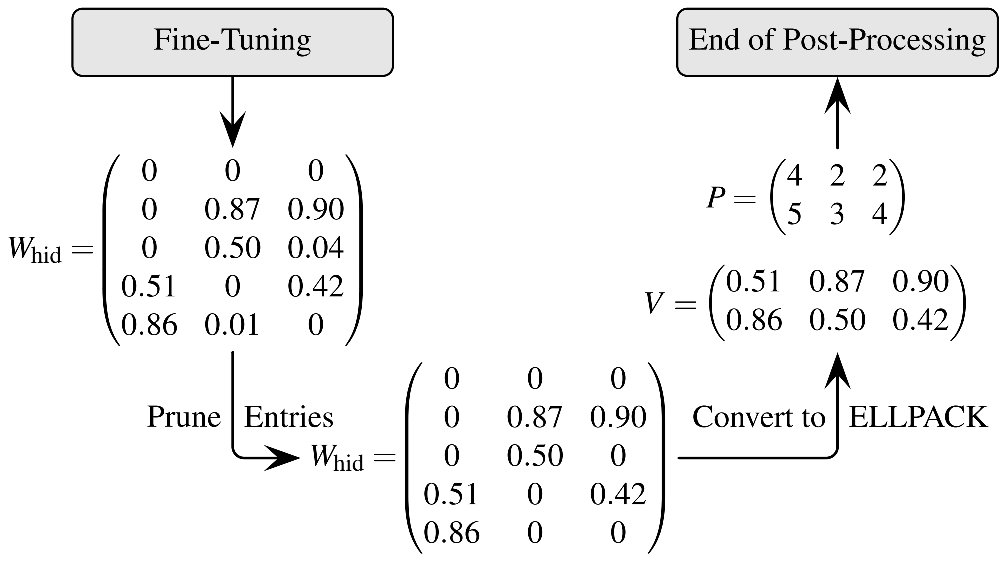

Fine-Tuning
End of Post-Processing
$W_{\text{hid}} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0.87 & 0.90 \\ 0 & 0.50 & 0.04 \\ 0.51 & 0 & 0.42 \\ 0.86 & 0.01 & 0 \end{pmatrix}$
Prune Entries
$W_{\text{hid}} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0.87 & 0.90 \\ 0 & 0.50 & 0 \\ 0.51 & 0 & 0.42 \\ 0.86 & 0 & 0 \end{pmatrix}$
Convert to ELLPACK
$V = \begin{pmatrix} 0.51 & 0.87 & 0.90 \\ 0.86 & 0.50 & 0.42 \end{pmatrix}$
$P = \begin{pmatrix} 4 & 2 & 2 \\ 5 & 3 & 4 \end{pmatrix}$

Figure 6.3: Illustration of the post-processing stage of SMLP training. The fine-tuning phase produces a sparsely populated weight matrix W_{hid} (here in the illustration with a sparseness of 0.70 with respect to Hoyer's σ in each column). The columns of W_{hid} are then projected onto an L_0 pseudo-norm constraint for maximum sparseness (here using a threshold of $0.05625 = {}^{0.90}/_{2^4}$). Finally, W_{hid} is converted to the ELLPACK storage format for sparse matrices.

The fine-tuning phase is finished when no more significant improvement on the learning set can be observed. If the projection was carried out onto a convex set, convergence would be guaranteed theoretically [103]. However, as the target set for the sparseness-enforcing projection operator is not convex, there is no rigorous argument for convergence in the mathematical sense. As is common practice in neural network training [53], the step size is multiplicatively annealed after every learning epoch, such that a termination criterion based on relative objective function value change will be satisfied eventually.

6.3.3 Post-Processing

After convergence of the fine-tuning, a final post-processing step is carried out to prune irrelevant synaptic connections and convert the hidden layer's weight matrix W_{hid} to a sparse data structure. The final set of degrees of freedom $\mathcal{W}_{\text{post-proc}}$ can then be used to directly realize a classifier. An illustration of the steps performed during post-processing is given in Figure 6.3. Although the weight matrix is already sparsely populated, it can be made even sparser by explicitly setting entries with very low absolute value to zero. This can also be considered a projection onto an L_0 pseudo-norm constraint, see Section 4.2.

The threshold value for pruning and thus the number of surviving active synaptic connections is determined as follows. The threshold is first initialized with the maximum absolute value of all entries in W_{hid}. Then all entries with absolute value less than the threshold are set to zero. Finally, the classifier altered this way is applied to the learning set and the classification performance is recorded. If the classification performance did not degenerate above another threshold from the performance of the original classifier, this threshold is used and the procedure stops. Otherwise, the threshold is halved, and the procedure restarts with pruning the entries using the decreased threshold. Note that the algorithm terminates at the latest when the pruning threshold is lower than the minimum value of all absolute entries of W_{hid} which do not vanish. The only relevant parameter of this method is hence the allowed degradation in classification performance. The higher this tolerance is set, the fewer connections can be removed in general, and vice versa.

Because the sparseness constraint of connectivity was formulated to apply on all columns of the weight matrix independently, all columns of the hidden layer's weight matrix possess about the same number of non-vanishing entries after pruning. A modified ELLPACK/ITPACK storage format [104] is ideal for storing a sparse matrix with this particular structure and for implementation of an algorithm that computes the product of such a matrix with a densely populated vector. The assumption is that there are at most $k \in \mathbb{N}$ nonzero entries in each column of W_{hid}. The sparse data structure then consists of a matrix $P \in \mathbb{N}^{k \times n}$ that stores the row indices of all non-vanishing entries, and a matrix $V \in \mathbb{R}^{k \times n}$ where the corresponding values are stored. The column indices of the entries are given implicitly by the corresponding indices in P and V. The entries in the columns of P are sorted in ascending order by convention.

Note that there is wasted space in this storage format if not all columns of W_{hid} possess exactly k non-vanishing entries. In this situation, the matrices P and V are stuffed with zero entries to always guarantee access to well-defined memory. If, however, the L_0 pseudo-norms of the columns of W_{hid} are sufficiently close to k, as usually is the case due to the special sparseness constraint, then the overhead is negligible. Note that the original ELLPACK storage format was defined analogously, but assuming a maximum nonzero entry number per row and thus storing rows consecutively, and imposing no constraints on the order of column indices [104]. The format proposed here is merely a slight adaptation to the specific structure of SMLPs.

6.4 Experiments

The proposed training algorithm for sparsely connected MLPs was evaluated on the MNIST database of handwritten digits as introduced in Chapter 3. Additionally, conventional MLPs and Optimal Brain Damage were evaluated and all approaches were compared with each other. Since the memory consumption of these approaches does not depend on the size of the learning set, the augmented variants of the MNIST learning set could also be used.

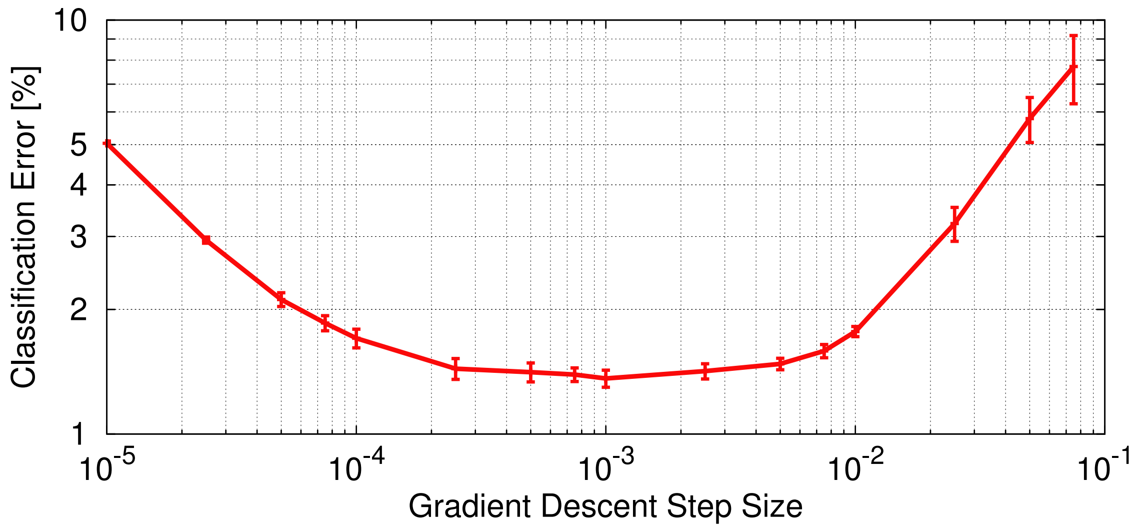
The results of [58] can be taken as a guideline for which classification performance can be expected. The authors have used two-layer MLPs with 800 hidden units and achieved classification errors of 1.6%, 1.1% and 0.7% on the evaluation set when training on the original learning set, a learning set augmented by samples generated by affine distortions, and a learning set enriched with elastically distorted samples, respectively. In the experiments described below, the number of hidden units was chosen to be $n := 1000$, corresponding to two times overcomplete internal representations as shown in Chapter 3. As was demonstrated by [59], generation of artificial training samples is beneficial for neural networks with up to twelve million synaptic connections. Hence if an adequate number of learning samples is used, generalization performance is likely to improve compared to when only 800 hidden units were used. The chosen number of hidden units was moreover confirmed through cross-validation.

6.4.1 Conventional Two-Layer MLPs

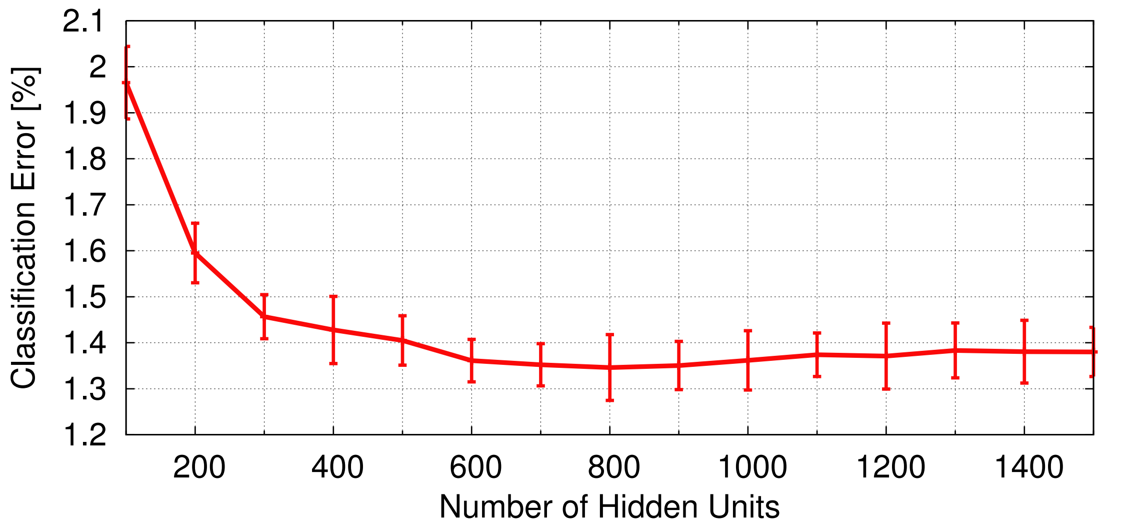
For baseline results, conventional densely connected two-layer MLPs with randomly initialized weights have been trained on the three variants of the MNIST learning set and their classification performance was measured on the MNIST evaluation set. Before training, two-fold cross-validation was carried out on the jittered learning set to determine optimal values for the gradient descent step size and the number of hidden units. Each run of the cross-validation was repeated ten times to yield results invariant to random effects. Hence, for each pair of candidates twenty experiment outcomes were available. The results of the cross-validation are depicted in Figure 6.4. The combination of step size and number of hidden units that yielded the best median validation results were 10^{-3} and 800, respectively. For training on the entire learning set, the number of hidden units was set to 1000 since more distinct samples are available and cross-validation has shown that an increased number of hidden units does not taint the generalization performance. Subsequently, two-fold cross-validation was performed on both the original and the elastically distorted MNIST learning sets to determine the best step size for 1000 hidden units. For the original MNIST learning set this step size was 10^{-3}, while for the elastically distorted learning set the winning step size was $5 \cdot 10^{-3}$.

The final training methodology was as follows. Each variant of the learning set was treated independently from all other variants. Using the step size as determined during cross-validation, 55 runs of MLP training were carried out for each variant of the learning set. After training, all classifiers were applied to the evaluation set to determine the evaluation error. Then, the best four and the worst four results were discarded and not included in further analysis. Hence, a random sample of size 47 was achieved, where 15% of the original data were trimmed away. Finally, the median classification error and the corresponding standard deviation were computed. The results for all three variants of the learning set are given in Table 6.1.

Similar to [56, 57], augmenting the learning set with samples jittered by one pixel halves the number of misclassifications compared to when only the original learning set is used. This shows that MLPs are not inherently able to learn invariants and thus rely heavily on prior



(a) Validation error depending on the step size of stochastic gradient descent for $n := 1000$ hidden units. The best results were obtained for a step size of 10^{-3}. Step sizes in a small neighborhood of this value yielded about equal validation performance.



(b) Validation error depending on the number of hidden units for a step size of 10^{-3}. When there are more than 600 hidden units, hardly any improvement in the validation performance can be observed. Since merely half the amount of samples for training was available for two-fold cross-validation, the number of hidden units was set to one thousand for training on the entire learning set.

Figure 6.4: Results of ten runs of two-fold cross-validation on the jittered MNIST learning set. Here, two slices of the validation error for different choices of the gradient descent step size and the number of hidden units are shown. Error bars indicate $\pm$ one standard deviation from the median of classification errors achieved during cross-validation, where for each point twenty results were available.

Table 6.1: Evaluation results of conventional MLPs trained on three variants of the MNIST learning sets. The errors give median and standard deviation of 47 classifiers.

Variant of learning set	original	jittered	elastic
Classification error [%]	1.8 ± 0.09	0.88 ± 0.03	0.61 ± 0.04

knowledge. Further, when the 13 500 000 samples from the elastically distorted learning set are used, classification performance increases even more. Generalization capabilities hence improve with learning set size. A comparison with the results of [58] where only 800 hidden units were used shows that the mean performance of the classifiers trained on the original learning set decreased while the performance of the classifiers trained on the augmented learning set increased. Hence 1000 hidden units lead to overfitting when the learning set with only 60 000 samples is used, but the additional hidden units help improve the generalization performance for larger learning sets. Because of the poor results when only the original samples were used, this variant was not included in further experiments.

Additional trainings on the jittered MNIST learning set were carried out for the number of hidden units n chosen from the set $\{100, 200, \ldots, 800, 900\}$ to assess its impact on the generalization capabilities. For each value of n, fifty-five training runs were performed to account for random effects, and the resulting classifiers were applied to the MNIST evaluation set and then the best four and worst four were trimmed away. The results of this experiment are given in Table 6.2, together with the result achieved above for $n = 1000$. A median error of 1.0% can be achieved for $n = 300$ already, and the error remains below that mark for more hidden units. The best result was achieved for $n = 1000$, for $n = 600$ there were only three more misclassifications. Therefore, to save about forty percent of the time needed for classification with nearly equal classification performance it would be sufficient to use only 600 hidden units. As will be shown in the next section, however, the SMLP algorithm is able to achieve far better results.

Another experiment evaluated whether a different weight initialization scheme than using random numbers would produce better classifiers. For this, $n := 1000$ samples from the learning set were picked at random to initialize the n columns of the hidden layer's weight matrix. The output layer continued to be initialized using random numbers. Refer to Section C.4 for a detailed discussion of this variant of unsupervised pre-training. The previous training methodology was applied in an identical manner, that is fifty-five classifiers were trained and then 15% were trimmed away. On average, this approach yielded an error rate of $0.91\% \pm 0.05\%$, which is slightly worse compared to when random numbers are used. When the SCFC algorithm was used instead of the replicated learning samples, a generalization error of $0.91\% \pm 0.06\%$ was obtained. This particular experiment is discussed in greater detail in the next section.

Table 6.2: Classification error on the MNIST evaluation set for trainings of densely connected MLPs on the jittered MNIST learning set. Here, the number of hidden units n was varied. The best results were obtained for $n = 1000$, whereas for $n = 600$ there were only three more misclassifications.

Hidden units n	100	200	300	400	500
Evaluation error [%]	1.6 ± 0.12	1.2 ± 0.10	1.0 ± 0.07	0.97 ± 0.05	0.94 ± 0.04
Hidden units n	600	700	800	900	1000
Evaluation error [%]	0.91 ± 0.04	0.92 ± 0.04	0.92 ± 0.04	0.91 ± 0.03	0.88 ± 0.03

These results show that in the situation of the jittered MNIST learning set, an unsupervised initialization scheme is not better than simply using random numbers. Since 540 000 training samples are available here, this might be enough data to start from random numbers and still achieve good generalization performance.

Further experiments were conducted to assess whether it is possible to achieve a neural network with sparse connectivity using very simple methods. The starting point for these experiments were the classifiers trained on the jittered learning set where a median error of 0.88% was achieved. In the first experiment, two-thirds of all synaptic connections in the hidden layer were removed through an L_0 pseudo-norm projection. This increased the median classification error on the evaluation set to 1.2%. Hence classification results degraded while a major part of synaptic connections still remained. In the second experiment, an L_0 projection was used to eliminate only that many synaptic connections so that the classification error did not increase. After that, the connectivity rate was 54.3% in the median of all classifiers. Therefore only very high connectivity rates can be achieved without tainting the classification performance. In a final experiment, the sparseness-enforcing projection operator was applied to the columns of the hidden layer's weight matrix. The projection operator was parameterized such that the results had the same L_2 norm as the input vectors, and the target degree of sparseness was set to 0.75. After the projection, the classifiers were applied to the evaluation set, achieving a very high median classification error of 2.8%. Summing up, all these naive heuristics could not accomplish to produce a classifier with both a low degree of synaptic connectivity and a low number of misclassifications on the evaluation set. Hence more sophisticated methods have to be employed to produce such classifiers.

6.4.2 Sparsely Connected MLPs

Sparsely connected MLPs have been trained using the method proposed in Section 6.3 on the jittered and the elastically distorted MNIST learning sets for all sparseness degrees between 0.65 and 0.95 in steps of size 0.05. Because of the analysis of the sample sparseness in

Chapter 3, lower sparseness degrees were not used. The original learning set was not employed as it already induced poor generalization capabilities with conventional MLPs. The step size for projected gradient descent was determined using cross-validation as already described for densely connected MLPs. The result was that for all combinations of learning set variant and sparseness degree, $5 \cdot 10^{-3}$ yielded the lowest median error during cross-validation. The duration of fine-tuning until convergence was about equal to the time it took conventional MLPs to converge. The pre-training was carried out using SCFC on 60 000 randomly selected samples from the respective learning sets. This required considerably less time than the MLP training on the full learning sets. As the post-processing step consists mainly of multiple prediction of class memberships, this last stage was the shortest to carry out.

Likewise to the experiments for the conventional MLP, 55 classifiers were trained for each sparseness degree both on the jittered and the elastically distorted learning set, and 15% of the raw results were trimmed away. The outcome of these runs is given in Table 6.3. Additionally, the results from Section 6.4.1 are included with the sparseness degree denoted as "none". Besides, the ratio of active synaptic connections in the hidden layer was counted after the post-processing stage and the number of all active synaptic connections including the output layer in the neural network was determined.

The classification capabilities are not only comparable to that of the densely connected MLPs, but they are even better for some sparseness degrees. The best performance could be obtained for $\sigma_W = 0.75$ for both enriched variants of the learning set. This value also turned out to be optimal for the SCFC experiments described in Section 5.3.2. Classification performance is poor for a very sparse connectivity where $\sigma_W = 0.95$. However, as only about three percent of all connections in the hidden layer were allowed to be active, this can be regarded an extreme case where better results cannot be expected. It is also noteworthy that the connectivity rate in the hidden layer is always higher for the elastically distorted learning set compared to the jittered learning set. This is due to the post-processing step, where no more synaptic connections could be pruned without increasing the error on the learning set.

The boost in generalization performance can be explained by the bias/variance decomposition of the generalization error that was discovered by [105]. Forcing the hidden layer to be sparsely connected significantly decreases the effective number of degrees of freedom of the model. Because the model is less complex, classifiers tend not to overfit when adapted on a learning set and generalization capabilities improve on average.

A subset of the filters in the hidden layer from the trainings on the jittered MNIST learning set is depicted in Figure 6.5, including ones from the training of conventional MLPs. While for conventional MLPs all entries in the filters are nonzero, sparseness has a significant impact on the appearance of the SMLP filters. The shape of individual digits can be recognized for $\sigma_W = 0.65$, together with adjacent regions of inverse signs. These contrast detector areas prevail even for higher sparseness degrees, until they become hardly visible at $\sigma_W = 0.90$. Hence these resulting filters are similar to those found in Chapter 5, although those were smoother. The lack of smoothness may result from the number of samples in the learning set

Table 6.3: Results of sparsely connected MLPs trained on the jittered and the elastically distorted MNIST learning set. The sparseness degree σ_W is the decisive parameter of the training algorithm as proposed in Section 6.3. The connectivity rate and the number of synaptic connections were determined after training had finished.

Variant of learning set	Sparseness degree σ_W	Connectivity rate in hidden layer [%]	Total number of active synaptic connections	Classification error on evaluation set [%]
jitter	none	100 ± 0.0	795 000	0.88 ± 0.03
jitter	0.65	18.1 ± 0.06	153 000	0.89 ± 0.07
jitter	0.70	14.9 ± 0.07	128 000	0.85 ± 0.07
jitter	0.75	12.1 ± 0.08	106 000	0.81 ± 0.05
jitter	0.80	9.5 ± 0.06	85 000	0.84 ± 0.05
jitter	0.85	7.1 ± 0.05	67 000	0.91 ± 0.06
jitter	0.90	3.8 ± 0.05	41 000	0.93 ± 0.08
jitter	0.95	2.4 ± 0.03	30 000	1.58 ± 0.11
elastic	none	100 ± 0.0	795 000	0.61 ± 0.04
elastic	0.65	19.2 ± 0.13	162 000	0.63 ± 0.05
elastic	0.70	16.5 ± 0.11	140 000	0.59 ± 0.06
elastic	0.75	14.3 ± 0.09	123 000	0.58 ± 0.06
elastic	0.80	10.1 ± 0.10	90 000	0.62 ± 0.05
elastic	0.85	8.4 ± 0.08	77 000	0.66 ± 0.05
elastic	0.90	4.6 ± 0.05	47 000	0.70 ± 0.07
elastic	0.95	3.5 ± 0.04	38 000	1.42 ± 0.15

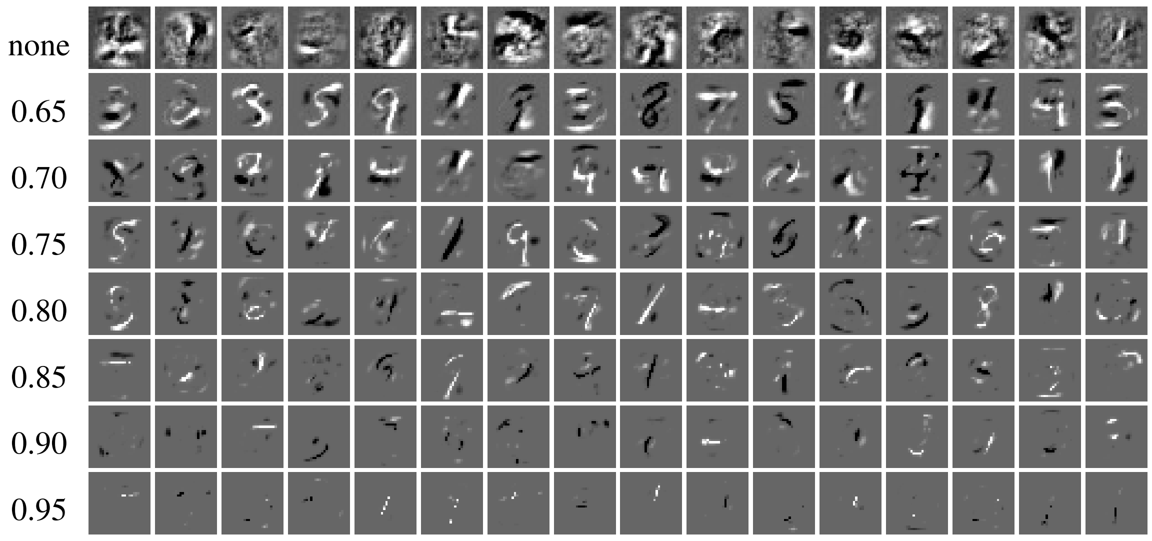
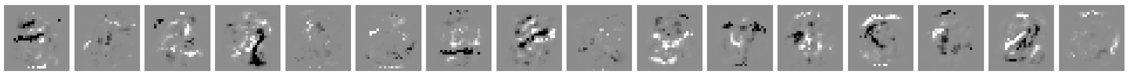


Figure 6.5: Exemplary filters resulting from SMLP training on MNIST as proposed in this chapter. The sparseness degree of connectivity σ_W was set as depicted beneath the rows, where "none" indicates filters from a conventional MLP for comparison. The emergence of contrast detectors as in Chapter 5 was reproduced, although here the filters are less smooth due to the large number of training samples.

being nine times higher than with the SCFC experiments, and it may be harder to sustain such a morphology when the classification error needs to be minimized on such a large number of data points. Finally, for $\sigma_W = 0.95$ the nonzero entries of the filters are hardly visible. This is because there are mostly fewer than 20 nonzero entries of the total $28^2 = 784$ entries in the filters. It is hence not surprising that there is a large performance drop for such a high sparseness degree.

The training procedure proposed in Section 6.3 was verified using additional experiments on the jittered learning set. In a first experiment, the proposed training algorithm was carried out with $\sigma_W = 0.75$ except for the pre-training phase being skipped and initializing the degrees of freedom for the fine-tuning phase from random numbers. The training did converge, however, classification results degraded to $1.1\% \pm 0.07\%$ in the mean of 47 runs. This is significantly worse compared to the 0.81% error from the sparsely connected MLP and the 0.88% error from the conventional MLP where no sparseness projections were used. Additionally, the resulting filters in the hidden layer resemble a random pattern with little structure, see Figure 6.6a. This indicates that pre-training is crucial when sparse connectivity is demanded.

In another experiment, the original training process was used, including pre-training with SCFC, but the sparseness projection of the hidden layer's weight matrix was omitted during fine-tuning. The weight matrix was sparsely populated after initialization by SCFC using a sparseness degree of $\sigma_W = 0.75$. As the fine-tuning phase was merely an unconstrained


(a) Results from the experiment where random numbers were used to initialize the degrees of freedom instead of pre-training. The filters possess only very little structure with a dominating random pattern.

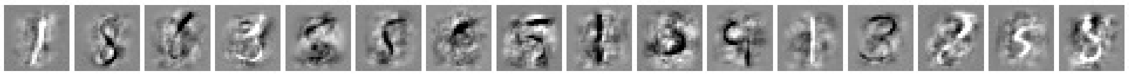

(b) Results from when no sparseness projection was carried out during fine-tuning. Individual digits can be recognized and also small regions with areas of inverse sign, but the filters are distorted by random clouds. The sparse connectivity achieved through pre-training is lost after this unconstrained backpropagation.

Figure 6.6: Resulting filters in the hidden layer for auxiliary experiments that were performed to verify the proposed three-staged training procedure.

optimization of the classification error, sparse connectivity was neither preferred nor penalized by the optimization algorithm. In effect, the weight matrix was no more sparsely populated after plain backpropagation. Examples of the resulting filters in the hidden layer are depicted in Figure 6.6b. The filters now resemble prototypes of individual handwritten digits, but they are overlaid with random clouds.

Regarding the classification performance, an error of $0.91\% \pm 0.06\%$ was achieved on the evaluation set. This is only slightly worse than the result of the conventional MLP, but significantly worse compared to the sparsely connected MLP with $\sigma_W = 0.75$. Hence in this case the pre-training did not provide any significant benefit over random initialization of the filters. Although in some setups unsupervised pre-training did help to increase classification results [55], it does not seem necessary when a large number of training examples is available [59].

In a final experiment, the entire training process as proposed in Section 6.3 was carried out, with the application of the sparseness-enforcing projection operator during fine-tuning being replaced with a simple projection onto an L_0 pseudo-norm constraint, see Section 4.2. The number of non-vanishing entries $\kappa \in \mathbb{N}$ in each hidden unit with respect to the L_0 projection was set to achieve the same connectivity rates as in the experiments where the sparseness-enforcing projection operator was used as given in Table 6.3.

Fifty-five classifiers were trained on the jittered MNIST learning set for each value of κ and then applied to the MNIST evaluation set to trim 15% away, which resulted in 47 classifiers. The classification results of these neural networks are given in Table 6.4. Clearly, the achieved classification capabilities are worse compared to the original training algorithm that used a σ projection. The result for the highest sparseness with a connectivity rate of only 2.4%, however, is substantially better compared to the proposed approach. The simple L_0 projection is hence more stable in the threshold region of unreasonable high sparseness degrees, which is yet not of practical interest because of the bad classification results.

Table 6.4: Results of sparse MLP training when a projection onto an L_0 constraint is used instead of a σ constraint. The number of non-vanishing entries of each hidden unit κ was chosen to match the connectivity rates of the original experiments.

Connectivity rate in hidden layer [%]	Nonzero entries per hidden unit κ	Classification error on evaluation set [%]
18.1	142	0.97 ± 0.06
14.9	117	0.99 ± 0.04
12.1	95	1.03 ± 0.03
9.5	75	1.04 ± 0.07
7.1	56	1.06 ± 0.08
3.8	30	1.16 ± 0.08
2.4	19	1.29 ± 0.09

6.4.3 Optimal Brain Damage

The Optimal Brain Damage method as described in Section 2.4.9 was also evaluated on the MNIST database of handwritten digits. Because no experimental results of this approach on the MNIST database have been published, it was reimplemented and applied independently of the original publication. The essence of the OBD method is to prune synaptic connections that are not relevant for the classification decision *after* training of the neural network has converged. Once a certain fraction of connections has been removed, the network is retrained again until convergence, holding removed connections to zero weight. This alternating fashion of training and pruning is carried out until the target connectivity rate is achieved.

In contrast to the sparsely connected MLPs as proposed in this work, OBD can be considered a destructive algorithm. The measure used to assess connection relevance is linear in the corresponding diagonal entry of the Hessian of the objective function and the squared magnitude of the connection. If a connection was removed by error, it is then not possible to recover. Because in SMLPs the sparseness-enforcing projection operator is only applied after each epoch rather than after each sample presentation, connections can be restored during learning if it helps to decrease the objective function.

The following methodology was used for the experiments. First, the conventional MLPs as trained for the experiments from Section 6.4.1 were used for initialization. Then, the diagonal of the Hessian of the objective function on the entire learning set was computed using Theorem 50. This information was used to compute the relevance of each synaptic connection from the hidden layer using the expressions from Section 2.4.9. Next, all synaptic connections were ranked based on their relevance and the lower 50% were removed. The networks were then

Table 6.5: Results of 47 runs of the Optimal Brain Damage method applied to the jittered MNIST learning set. After convergence of the trainings, 50% of the still active synaptic connections were removed. This was repeated until only 3.125% of all possible connections in the hidden layer remained.

Connectivity rate in hidden layer [%]	Total number of synaptic connections	Classification error on evaluation set [%]
100	795 000	0.88 ± 0.03
50	403 000	0.89 ± 0.04
25	207 000	0.88 ± 0.05
12.5	109 000	0.89 ± 0.04
6.25	60 000	0.94 ± 0.05
3.125	35 000	1.03 ± 0.06

retrained while holding the synaptic weight of all connections previously removed to zero. The step size of stochastic gradient descent for the retrainings was chosen to match that of the training of the original densely connected MLP. After training convergence, the next 50% of active connections were removed and the process was repeated until only $2^{-5} = 3.125\%$ of possible connections in the hidden layer remained. Because more sparse connectivity lead to bad classification performance for SMLPs, this was chosen as the lower end of connectivity rates for OBD.

One disadvantage of OBD is that before pruning the neural network has to reach a minimum of the objective function such that the approximation for connection relevance holds. This implies that to achieve a connectivity rate of 3.125% while halving the number of connections in each pruning step, six complete runs of neural network training have to be subsequently carried out. As connection removal is not without effect on the objective function, all trainings take significant time until a new minimum is reached. Therefore running OBD takes up to six times longer than training a conventional MLP or a sparsely connected MLP, dependent on the target connectivity rate. For this reason, the elastically distorted MNIST learning set was not used to benchmark OBD. Instead, the jittered learning set was used exclusively which is however sufficient to compare OBD with SMLPs and standard MLPs.

Using this methodology, 55 runs of OBD were carried out on the jittered MNIST learning set and the top four and bottom four results were trimmed away. The outcome of the middle 47 runs is given in Table 6.5. Clearly, the performance is equal to the densely connected MLP for connectivity rates of 50%, 25% and 12.5% except for minor numerical deviations. For a connectivity rate of 6.25% the classification error increases by a small amount, and for a connectivity rate of 3.125% the error is slightly greater than 1.0%.

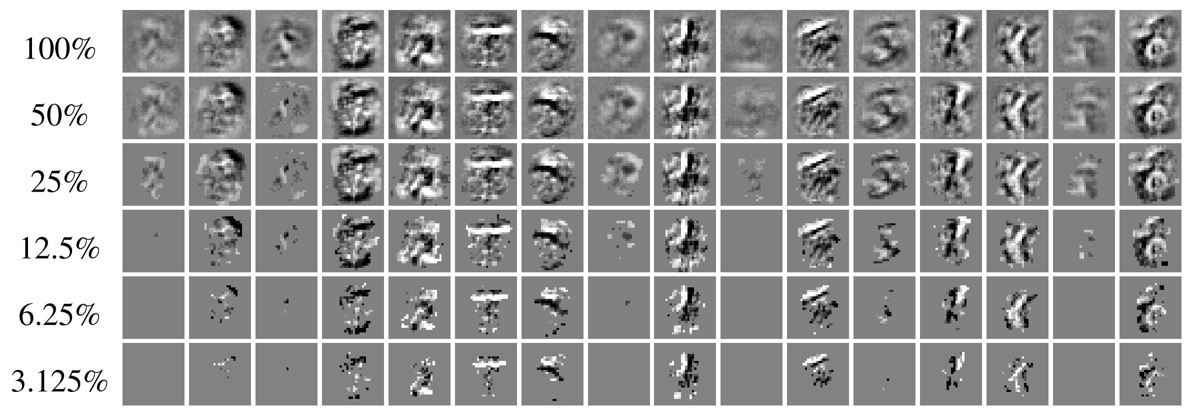


Figure 6.7: Visualization of how filters in the hidden layer of a neural network change during Optimal Brain Damage training. The rows show different stages of pruning and are labeled with the corresponding connectivity rate of the hidden layer. The columns show individual neurons of the hidden layer. The neurons are thinned out with increasing sparseness of connectivity, and some are removed entirely as all synaptic weights vanish.

A visualization of how the neurons change in the hidden layer during OBD training is given in Figure 6.7. It can be seen that the filters are thinned out by setting the entries with the smallest absolute values to zero. There are also filters where all the entries are set to zero at a connectivity rate of 12.5%. In contrast to SMLPs, the sparseness constraints are formulated to affect the entire weight matrix of the hidden layer rather than its independent columns.

The advantage is that this allows neurons in the hidden layer to be completely removed. The drawback, however, is that it is not guaranteed that all remaining columns in the weight matrix are equally sparsely populated. Therefore, a sparse storage format other than ELLPACK has to be employed to store the synaptic connections. Because the storage format is then not well-structured but arbitrary, the run-time of the matrix-vector multiplication during recall phase is higher in practice because of more irregular memory access patterns. The second drawback is that the neural network now computes an embedding in a space with fewer dimensions than before. It is not obvious that fewer dimensions always suffice to achieve the desired generalization capabilities.

6.5 Discussion

A method for efficient training of sparsely connected MLPs based on projected gradient descent has been proposed in this chapter. Experimental results demonstrated that the approach is not only competitive to conventional densely connected MLPs, but also outperforms them

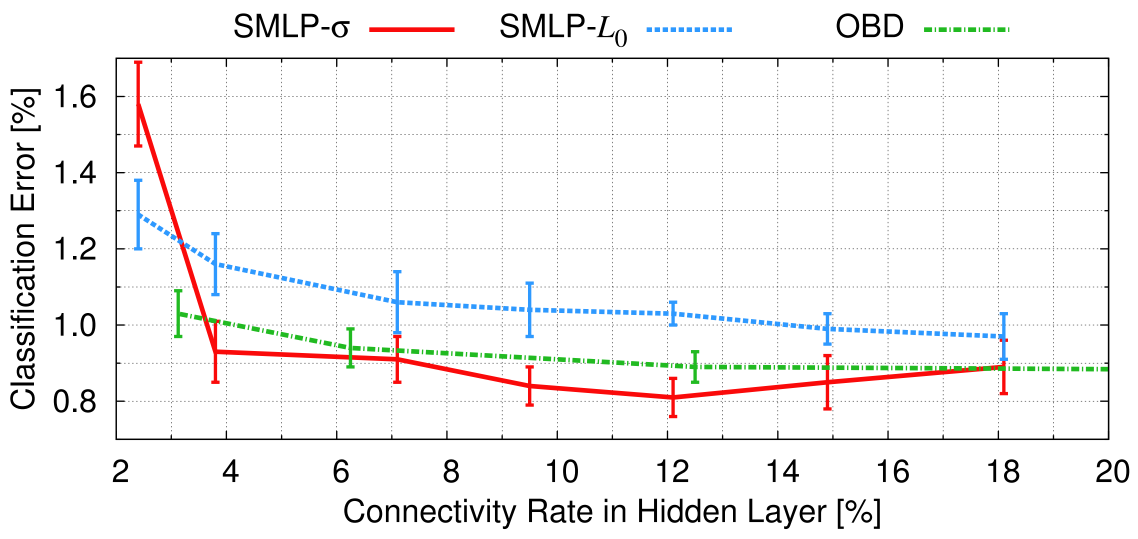


Figure 6.8: Comparison of the achieved classification error on the evaluation set for three different approaches trained on the jittered MNIST learning set. The results for variant SMLP-σ were achieved using the algorithm proposed in Section 6.3, for variant SMLP-L_0 the σ projection was replaced with an L_0 projection as described in the end of Section 6.4.2, and OBD denotes the results using the Optimal Brain Damage technique. The results of SMLP-σ and OBD are about comparable, while both are substantially better than SMLP-L_0.

when the sparseness degree is adjusted correctly. An overview of the achieved evaluation error in dependence of the connectivity rate is shown in Figure 6.8 for the sparsely connected MLPs using a projection onto either σ or L_0 sparseness constraints and the Optimal Brain Damage method. Clearly, SMLPs feature an improved classification performance over OBD for connectivity rates around 12.5%, and the performance is about equal for more sparse connectivity. Further, both methods are superior compared to when an L_0 projection is used to enforce sparse connectivity.

Optimal Brain Damage is quite effective in that it induces generalization capabilities comparable to those of standard MLPs for a variety of connectivity rates. Unfortunately, it has the severe drawback of increased training duration. SMLPs do not share this disadvantage, the duration of fine-tuning is about equal to the time needed for training conventional MLPs.

Because the time needed for pre-training and post-processing was definitely less than that of the fine-tuning stage, SMLPs could also be applied to the very large learning set augmented with elastically distorted samples. Using these artificial samples may slow down the training process, but with their help the classification error could be reduced by about 30% without any change to the architecture. The number of active synaptic connections increased only by a small amount compared to when the jittered learning set was used.

SMLPs are very appealing for use in scenarios where the computational complexity of classification is of utmost importance, as it has been demonstrated that only about one eighth of the synaptic connections in the hidden layer need to be taken into account for class membership prediction. It will be shown in Chapter 8 that by exploiting the sparse structure of the hidden layer's weight matrix, substantial speed-ups over conventional MLPs can be achieved. If merely the number of hidden units is reduced, only mild speed-ups can be yielded which are bought at the expense of an increased number of misclassifications. Therefore the proposed training algorithm can be considered superior to the classical backpropagation procedure.

7 Supervised Online Auto-Encoder

The approaches proposed so far in this work include SCFC from Chapter 5, which featured both sparse activity and sparse connectivity but which could not be applied to large learning sets due to considerable memory consumption, and sparsely connected MLPs from Chapter 6. The latter do not support sparse activity but could be applied to large learning sets, for example the elastically distorted MNIST learning set with 13.5 million distinct samples introduced in Chapter 3. This chapter proposes a new approach called *Supervised Online Auto-Encoder (SOAE)* that extends sparsely connected MLPs with the sparse activity property by implementing the sparseness-enforcing projection operator discussed in Chapter 4 as neural transfer function. In doing so, the internal representation is guaranteed to be sparsely populated, where the sparseness degree is a user-controllable parameter. Since the sparseness projection can be cast almost everywhere as vector-valued function π, which is differentiable almost everywhere, the SOAE network parameters can be optimized efficiently using gradient-based methods. Thus population sparseness is controlled rather than lifetime sparseness as in SCFC. As already mentioned in Chapter 5, and revisited in the present chapter, both notions are closely related.

In technical terms, SOAE can be considered a hybrid of an auto-encoder network and a two-layer neural network. As the transfer function ensures sparse activity, a latent variable that models sparse code words as in SCFC is no longer necessary. This further eliminates the inference error term and thus SOAE's objective function becomes simpler than SCFC's. Moreover, SOAE works in an online fashion on individual samples rather than on the entire learning set at once. Hence, its memory requirements are very low and large data sets can be processed easily. One drawback of this approach, however, is a more complex gradient computation. This is manifested in the complicated gradient of the projection operator π derived in Section 4.5 and because no latent code word matrix is used, SOAE's objective function contains a coupled term in its auto-encoder part which renders its gradient more difficult to compute.

These technical difficulties are nonetheless accompanied by superior generalization performance due to sparse activity, which will be demonstrated in this chapter. Since the sparseness projection can be grasped as non-local, adaptive shrinkage operation, see Chapter 4 and [78], its implementation helps to de-noise the internal representations [106]. This is in fact a superior approach than just using non-adaptive shrinkage-like transfer functions as in [93, 94], since these simple functions do not offer sufficient flexibility.

This chapter is organized as follows. First, the SOAE architecture is proposed and compared with previous approaches. Then, an optimization algorithm is proposed which extends and simplifies the respective algorithms of SCFC and SMLPs. Comprehensive experiments on different data sets assess the benefits of the proposed approach and conclude with the superiority of sparse information processing compared to conventional techniques. The chapter concludes with a discussion of the approach and the results of the experiments. Portions of the included material were previously published in [66].

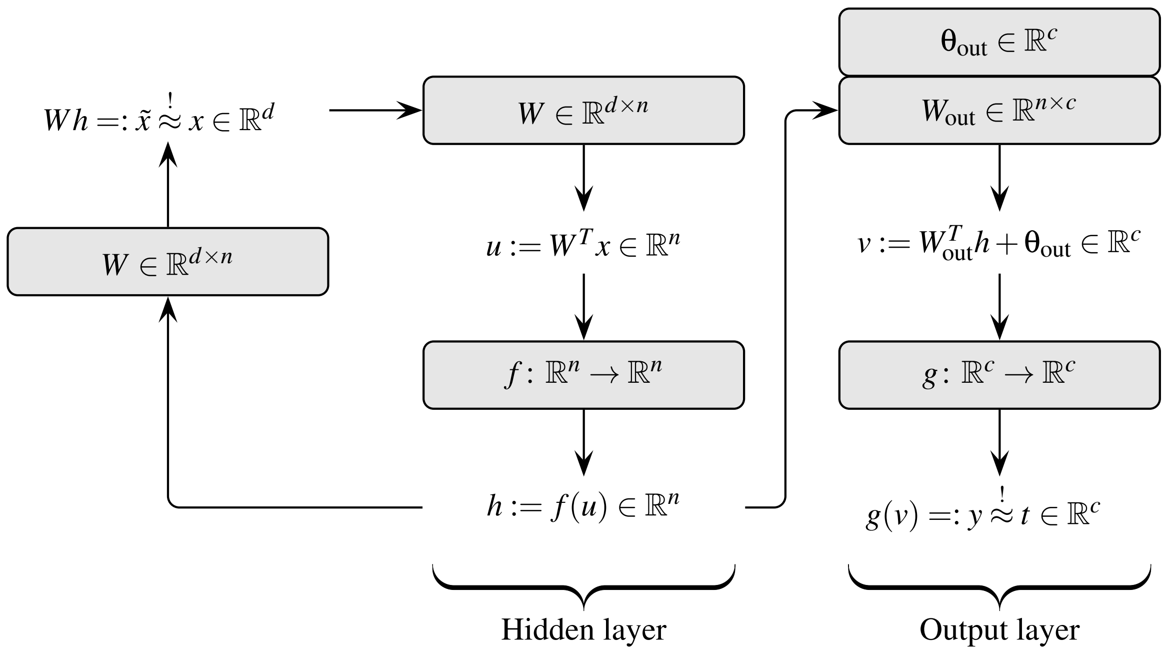
7.1 Architecture and Objective Function

The Supervised Online Auto-Encoder consists of the *reproduction module* and the *classification module*. The data flow in this architecture is shown in Figure 7.1. A comparison with SCFC's architecture, depicted in Figure 5.1, shows that SOAE is simpler in that no approximation of a latent code word with a feedforward code word is necessary. Instead, the reproduction path which is depicted on the left of Figure 7.1 operates by converting an input sample $x \in \mathbb{R}^d$ into an *internal representation* $h \in \mathbb{R}^n$, and using it directly to compute an approximation $\tilde{x} \in \mathbb{R}^d$ to the original input sample. In doing so, the product $u \in \mathbb{R}^n$ of the input sample with a *matrix of bases* $W \in \mathbb{R}^{d \times n}$ is computed, and a transfer function $f\colon \mathbb{R}^n \to \mathbb{R}^n$ is applied.

The vector-valued function f is chosen here as the projection onto sparseness constraints. This is realized by either using Hoyer's sparseness measure σ and the corresponding projection π, or as a simple alternative using the L_0 pseudo-norm where the projection was provided as the introductory example in Section 4.2. Both choices will be compared with each other in the experimental section. Using such a projection ensures that the internal representation is sparsely populated, thus accounting for the sparse activity property. Additionally, as much information as possible is retained because of the best approximation property of projections.

The reproduction is performed in the same manner as in a linear generative model, by multiplication of the matrix of bases with the internal representation. Hence the same matrix W is used for both encoding and decoding, rendering the reproduction module symmetric like in SCFC, which in turn is similar to other related architectures [28, 55, 33]. During development of this architecture, experiments have shown that when two separate matrices were used, they were highly correlated after a few optimization steps. Therefore the same variable was used for the purposes of encoding and decoding to simplify the architecture.

The right-hand side of Figure 7.1 shows the classification path of SOAE. Here, a *classification decision* $y \in \mathbb{R}^c$ is computed by feeding the internal representation h through a one-layer neural network. This layer is characterized by a matrix of weights $W_{\text{out}} \in \mathbb{R}^{n \times c}$, a threshold vector $\theta_{\text{out}} \in \mathbb{R}^c$ and a transfer function $g\colon \mathbb{R}^c \to \mathbb{R}^c$. As inference of the internal representation is also achieved through a one-layer neural network, the data flow from the input sample to the classification decision can be considered a two-layer MLP [89], analogous to SCFC. It should



$\theta_{\text{out}} \in \mathbb{R}^c$
$Wh =: \tilde{x} \overset{!}{\approx} x \in \mathbb{R}^d$
$W \in \mathbb{R}^{d\times n}$
$W_{\text{out}} \in \mathbb{R}^{n\times c}$
$W \in \mathbb{R}^{d\times n}$
$u := W^T x \in \mathbb{R}^n$
$v := W_{\text{out}}^T h + \theta_{\text{out}} \in \mathbb{R}^c$
$f\colon \mathbb{R}^n \to \mathbb{R}^n$
$g\colon \mathbb{R}^c \to \mathbb{R}^c$
$h := f(u) \in \mathbb{R}^n$
$g(v) =: y \overset{!}{\approx} t \in \mathbb{R}^c$
Hidden layer
Output layer

Figure 7.1: SOAE architecture consisting of a reproduction module ($x \mapsto h \mapsto \tilde{x}$) and a classification module ($x \mapsto h \mapsto y$). The latter can also be considered a two-layer neural network, with the hidden layer determined by weight matrix W and the output layer determined by weight matrix W_{out} and threshold vector θ_{out}. The matrix of bases W is enforced to be sparsely populated for sparse connectivity in the hidden layer. The transfer function f can be set to the sparseness-enforcing projection operator for sparse activity in the hidden layer. Adapted from [66].

be noted that choosing the hidden layer's transfer function to be a sparseness projection is a unique feature of SOAE, which facilitates the elegantly mathematical use of sparse code words for classification tasks.

Similar to SCFC, the reproduction module should approximate the input samples, and the classification module should produce a classification decision equal to the labels of the samples. For this, let $s_R\colon M_R \times M_R \to \mathbb{R}$ and $s_C\colon M_C \times M_C \to \mathbb{R}$ be similarity measures defined on sets $M_R \subseteq \mathbb{R}^d$ and $M_C \subseteq \mathbb{R}^c$ for the reproduction module and the classification module, respectively. See Section C.6 for an overview of suitable similarity measures. Let $\tilde{x} := Wf\left(W^T x\right) \in \mathbb{R}^d$ denote the result of the input sample being encoded and decoded again, then the *reproduction error* is defined as $F_R := s_R(\tilde{x}, x)$. Here, the correlation coefficient is chosen for s_R because it is normed, invariant to affine-linear transformations and differentiable, see Lemma 82 in Appendix C and [107]. Because the sparseness projection π is invariant to affine-linear transformations where the scaling is positive, see Corollary 21, it is an ideal supplement to achieve a model invariant to those transformations, rendering pre-processing techniques such as mean-variance normalization of the input samples obsolete. Let $y := g\left(W_{\text{out}}^T f\left(W^T x\right) + \theta_{\text{out}}\right) \in \mathbb{R}^c$ be the classification decision as in Figure 7.1 and let $t \in \mathbb{R}^c$ be the teacher signal associated with the input sample x. Then $F_C := s_C(y,t)$ denotes the *classification error*. Here s_C is chosen to be the cross entropy similarity measure and g to be the softmax transfer function since they form a beneficial pairing for multi-class classification problems, see also Chapter 6.

Because both error functions are to be optimized simultaneously, the overall error function for an individual sample is defined as $F_{\text{SOAE}} := (1-\alpha)F_R + \alpha F_C$, where $\alpha \in [0,\ 1]$ is a trade-off variable. The overall objective function of SOAE is then defined as the mean value of F_{SOAE} over all samples of the learning set. To incorporate sparse connectivity, feasible solutions are restricted to fulfill $\sigma(We_i) = \sigma_W$ for all $i \in \{1,\dots,n\}$. This is the very same sparseness constraint encountered in SCFC and has been taken over from Non-Negative Matrix Factorization with Sparseness Constraints [26] as discussed in Section 2.4.3. The two extreme values for α are zero and one. In the case of $\alpha = 0$, SOAE is put in its purely unsupervised operating mode where the classification module has no impact on the objective function. This way, SOAE can also be applied to data sets not intended for classification tasks. In the other extreme case $\alpha = 1$, only the classification module is active and SOAE is essentially a two-layer MLP with sparse activity and sparse connectivity. Similarly to the SCFC optimization strategy proposed in Section 5.2.3, α can be used to blend continuously between these two extremes.

7.2 Training Algorithm

A stochastic projected gradient descent is proposed to tune the SOAE network parameters W, W_{out} and θ_{out}. As explained in Section C.2, an online learning procedure results in faster convergence and improved generalization capabilities over batch learning. The classification module is essentially a two-layer neural network, thus Theorem 49 from Chapter 6 can be applied to compute its gradients. When the transfer function g is the softmax transfer function, the similarity measure for classification s_C is the cross entropy function and the teacher signals are given as one-of-c codes, then Theorem 50 can be used to optimize the gradient computations. The reproduction module demands special attention as it is quite different to SCFC's. In SCFC, the feedforward code words and the latent code words were mathematically decoupled and consequently the gradient for the matrix of bases could be yielded by addition of two independent gradients. As $\tilde{x} = Wf\left(W^T x\right)$, that is the matrix of bases W emerges twice in multiplicative form in the expression, the chain rule of matrix calculus has to be used to derive the gradient for W:

Theorem 51. The gradient of the reproduction error with respect to the matrix of bases is

$$\left(\frac{\partial F_R}{\partial W}\right)^T = xgWf'(u) + g^T h^T \in \mathbb{R}^{d\times n} \text{ where } g := \frac{\partial s_R(\tilde{x},x)}{\partial \tilde{x}} \in \mathbb{R}^{1\times d}.$$

Since F_R does not depend on W_{out} or θ_{out}, the gradients for these quantities vanish.

Proof. Let $\tau \in S_{dn}$ according to Proposition 53 from Appendix B. Let $\bar{h} := h(W)$ be constant and independent of W, for clarity in the following equation. The chain rule yields

$$\begin{aligned}
\frac{\partial \tilde{x}}{\partial \operatorname{vec}(W^T)} &= W\frac{\partial h}{\partial \operatorname{vec}(W^T)} + \frac{\partial W\bar{h}}{\partial \operatorname{vec}(W^T)} = Wf'(u)\frac{\partial W^T x}{\partial \operatorname{vec}(W^T)} + \frac{\partial \left(\bar{h}^T \otimes E_d\right)\operatorname{vec}(W)}{\partial \operatorname{vec}(W^T)} \\
&= Wf'(u)\frac{\partial \left(x^T \otimes E_n\right)\operatorname{vec}\left(W^T\right)}{\partial \operatorname{vec}(W^T)} + \frac{\partial \left(\bar{h}^T \otimes E_d\right) P_\tau \operatorname{vec}\left(W^T\right)}{\partial \operatorname{vec}(W^T)} \\
&= Wf'(u)\left(x^T \otimes E_n\right) + \left(\bar{h}^T \otimes E_d\right) P_\tau.
\end{aligned}$$

Therefore, writing h instead of $\bar{h}$ and defining $g := {}^{\partial s_R(\tilde{x},x)}/_{\partial \tilde{x}} \in \mathbb{R}^{1\times d}$ as abbreviation, another application of the chain rule yields

$$\begin{aligned}
\frac{\partial F_R}{\partial \operatorname{vec}(W^T)} &= \frac{\partial s_R(\tilde{x},x)}{\partial \tilde{x}} \cdot \frac{\partial \tilde{x}}{\partial \operatorname{vec}(W^T)} = g \cdot \left[Wf'(u)\left(x^T \otimes E_n\right) + \left(h^T \otimes E_d\right) P_\tau\right] \\
&= \left[(x \otimes E_n) f'(u)^T W^T g^T\right]^T + \left[(h \otimes E_d) g^T\right]^T P_\tau \\
&= \operatorname{vec}\left(f'(u)^T W^T g^T x^T\right)^T + \operatorname{vec}(hg)^T.
\end{aligned}$$

In the last step $P_\tau^T = P_{\tau^{-1}}$ was used for transposing the argument of the vectorization operator. Application of Theorem 67 yields the claim. □

If f is the projection onto an L_0 pseudo-norm constraint, the computation of f' is straightforward as discussed in Section 4.5. When the sparseness-enforcing projection operator π has been chosen, one can exploit the fact that f' only emerges in multiplicative form in the statements of Theorem 49 and Theorem 51. It is then more efficient to use Corollary 43 rather than compute the entire gradient of π with the algorithm induced by Theorem 42, see Chapter 4.

Now that the theoretical prerequisites for SOAE optimization have been clarified, the comprehensive training algorithm can be introduced. Its parameters are given in the following. Since the input dimensionality and the output dimensionality of the network are already determined by the choice of a concrete data set, only the number of hidden units n can be used to control the network's architectural complexity. The target degree of sparse connectivity of the hidden layer $\sigma_W \in (0,\ 1)$ has the very same meaning as for SCFC and the sparsely connected MLPs. Likewise, if $f = \pi$ a target degree of the activity sparseness $\sigma_H \in (0,\ 1)$ is introduced to parameterize the projection. Here, internal representation sparseness increases with σ_H. The parameters $(\omega,\ \eta)$ of π are set to (arbitrary, unit_norm) to determine the target set with respect to Table 4.1. If instead the projection onto an L_0 pseudo-norm constraint is chosen, a parameter $\kappa \in \{1,\dots,n\}$ has to be set which controls the exact number of hidden units that are allowed to be active at any one time. Smaller values of κ lead to sparser activity when the L_0 pseudo-norm is used as sparseness measure.

Before the actual training, the step size for gradient descent is determined using two-fold cross-validation. Five runs of cross-validation are carried out for each candidate step size to account for random effects, and then the one providing the best median evaluation error is used. The matrix of bases W is initialized by replication of a subset of the learning samples and storing them in the columns of W. This guarantees that the hidden layer features non-random activity upon presentation of the very first samples to the network, further leading to significant gradients and hence to reduced training time, see also Section C.4. It is not necessary to rescale W due to the network's invariance to scaling; it is merely important that the initial step size was determined using the same scaling for reliable results. The degrees of freedom of the output layer W_{out} and θ_{out} are initialized by sampling from a zero-mean Gaussian distribution with a small standard deviation of $1/100$. Other values in the same domain have led to statistically equivalent results in preliminary experiments, showing that the precise value of the standard deviation is not crucial.

The trade-off variable α, which controls the impact of the reproduction error and the classification error on the objective function, is adjusted according to $\alpha(\nu) := 1 - \exp(-\nu/100)$, where $\nu \in \mathbb{N}$ denotes the number of the current epoch. Hence α starts at zero, which means that SOAE is run in its unsupervised operating mode, then α increases slowly and asymptotically reaches one, where only the classification module is active. In every epoch of learning, samples and their associated target vectors are drawn randomly from the learning set and used to update the network parameters with stochastic gradient descent. To ensure that the sparse connectivity property is fulfilled, the sparseness-enforcing projection operator is applied to the columns of W so as to maintain $\sigma(We_i) = \sigma_W$ for all $i \in \{1, \ldots, n\}$. Subsequently, the value for α is updated using the function described above, and the next epoch is started. Training is terminated when no more significant progress of objective function minimization can be observed.

The choice of the function for α was validated in preliminary experiments by comparing the results of SOAE training with the trivial choice $\alpha \equiv 1$. Figure 7.2 shows the progress of the objective function during training for both notions of α. The curves were averaged over eight runs on the jittered MNIST learning set with parameters set to $n := 1000$, $\sigma_W := 0.75$ and $\sigma_H := 0.80$. It is evident that using a smooth function from zero to one outperforms the function identical to one in terms of convergence speed and achieved objective function value. Further, using a smooth α yielded an evaluation error of $0.74\% \pm 0.03\%$, whereas a constant α was more than 6% worse on a relative scale with an evaluation error of $0.79\% \pm 0.04\%$. Hence the strategy of having α follow an inverted exponential function was selected for the experiments on classification tasks.

7.3 Experiments

Since SOAE possesses an unsupervised operating mode as well as a supervised operating mode, the algorithm was evaluated on the natural image patches as employed in the SCFC experiments from Section 5.3.1 and on the MNIST database of handwritten digits, which was

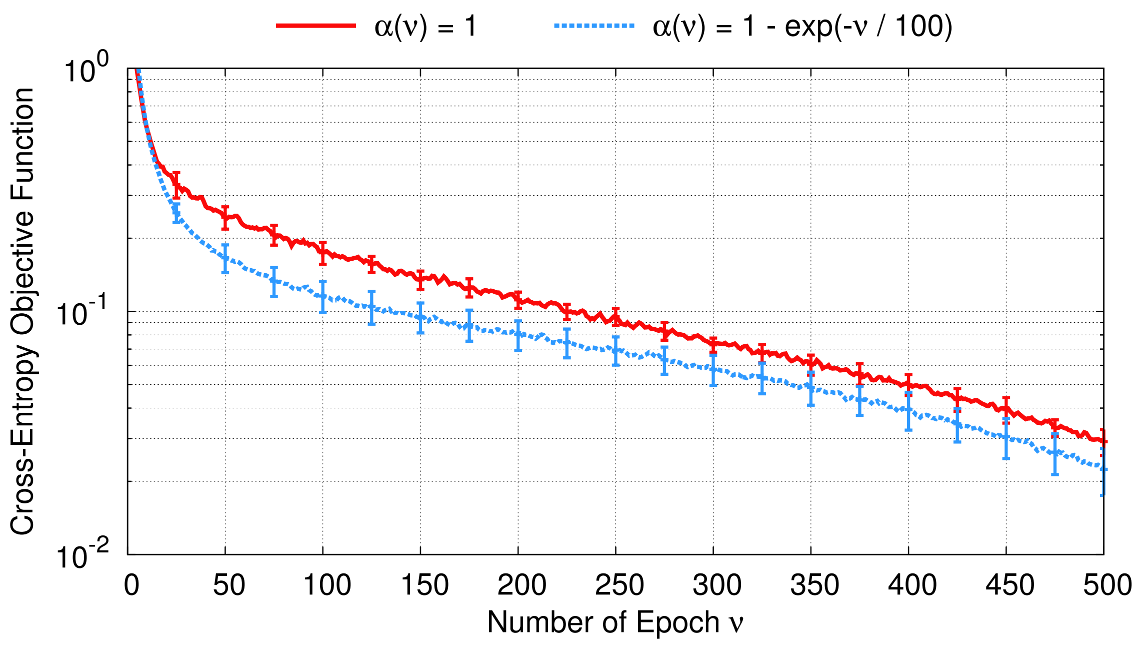
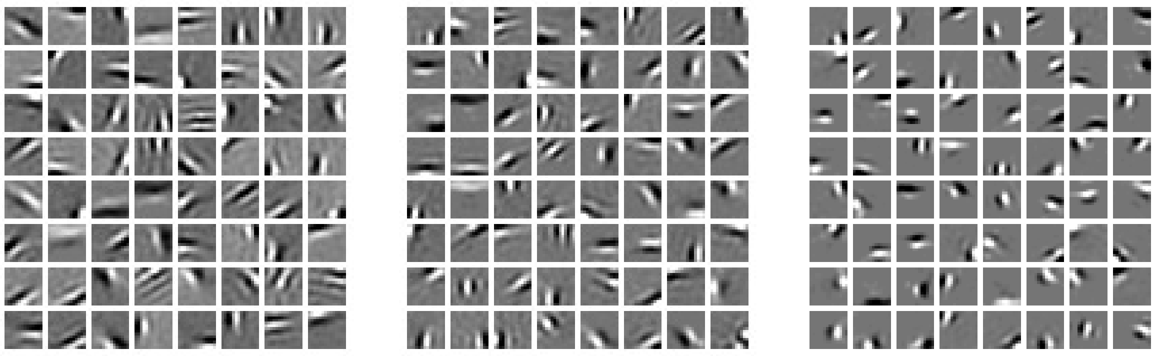


Figure 7.2: Comparison of two different strategies of choosing α as a function of the current epoch ν on a classification task. By forcing α to start from zero and only slowly reaching one, a continuous variant of unsupervised pre-training is carried out. This is superior in terms of objective function value and generalization error to the naive choice $\alpha \equiv 1$ which completely neglects SOAE's reproduction module.

introduced in Chapter 3. SOAE is able to handle very large data sets because it operates directly on individual samples rather than on entire batches. Therefore the MNIST experiments involve both the jittered and the elastically distorted learning sets. The original learning set with only 60 000 samples was not used as it had not led to reasonable results with conventional MLPs in Chapter 6.

7.3.1 Natural Image Patches

In a first series of experiments, SOAE was put in its unsupervised operating mode by setting $\alpha \equiv 0$ throughout optimization, and applied to the 30 000 patches of size 14×14 pixels extracted from the whitened natural images of [14]. Figure 7.3 shows results obtained by setting $n := 64$, $\sigma_H := 0.85$ and by choosing σ_W from $\{0.35,\ 0.50,\ 0.65\}$, which is the same parameter set used for the SCFC experiments on these learning samples. The resulting bases still resemble Gabor-like filters, though their morphology differs from the filters obtained through SCFC, see also Figure 5.5. At a first glance, there are more high-frequency filters produced by SOAE, whereas the SCFC filters almost exclusively consist of exactly two adjacent regions



(a) $\sigma_W = 0.35$. (b) $\sigma_W = 0.50$. (c) $\sigma_W = 0.65$.

Figure 7.3: Matrices of bases W yielded by running SOAE on natural image patches for three different levels of the connectivity sparseness σ_W. The target degree of sparse activity was set to $\sigma_H := 0.85$ throughout the experiments. SOAE has the ability to develop gratings, that is high-frequency filters with large circumference.

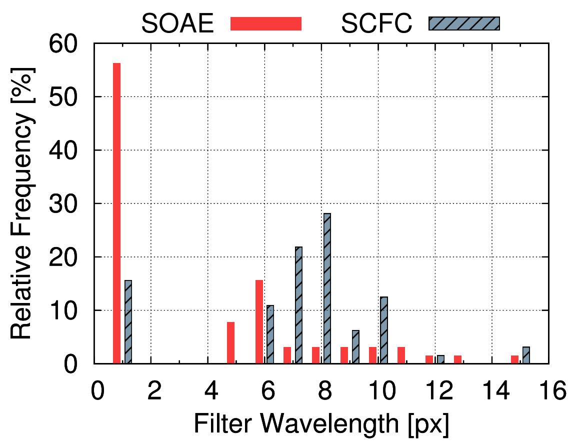
with opposite polarity. This change in appearance can be explained because SOAE by design features sparseness explicitly in space and only implicitly over time and the filters tend to adapt more to individual samples rather than to a smooth mean of samples. SOAE filters therefore exhibit a greater variety.

To analyze this in greater detail, the method for Gabor hyperparameter inversion described in Section 5.3.1 was applied to the SOAE filters as well. The following pairs of mean $\pm$ standard deviation of pixel-wise correlation coefficients ρ were obtained between the original filters and their Gabor approximation: $\rho = 0.978 \pm 0.012$ for $\sigma_W = 0.65$, $\rho = 0.967 \pm 0.019$ for $\sigma_W = 0.50$ and $\rho = 0.963 \pm 0.020$ for $\sigma_W = 0.35$. The Gabor inversion hence worked as well as in the SCFC experiments. An inspection of the main Gabor parameters for $\sigma_W = 0.50$ is given in Figure 7.4 for comparison between the SOAE filters and the SCFC filters. It is remarkable that most of the SOAE filters have a very low wavelength, which is equivalent to having a high frequency. Further, the Gaussian envelopes of the SOAE filters have larger standard deviations in the mean than the SCFC filters. This empirically fortifies the aforementioned observation that SOAE produces a richer set of filters with an increased level of expressiveness.

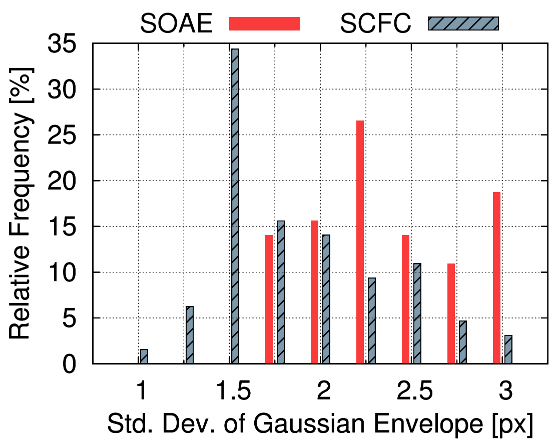
7.3.2 MNIST Database of Handwritten Digits

The next series of experiments was carried out on the MNIST data set. Here, the jittered learning set and the elastically distorted learning set were used and the original learning set was neglected since it previously resulted in poor results in Chapter 6.

In a first experiment, SOAE was applied in its unsupervised operating mode to the jittered learning set using parameters $n := 1000$, $\sigma_W := 0.75$ and $\sigma_H := 0.80$. A subset of the resulting



(a) Wavelength of the fitted Gabor filters. While more than fifty percent of the SOAE filters have a low wavelength and thus a high frequency, SCFC produces primarily filters with a medium wavelength.



(b) Standard deviation of the Gaussian envelope of the fitted Gabor filters. The SOAE filters have higher standard deviations in the mean and hence have a larger circumference than the SCFC filters.

Figure 7.4: Histograms of the main parameters of the Gabor filters that were fitted to the filters resulting from applying SOAE and SCFC to natural image patches. Clearly, both approaches generate morphologically different filters.

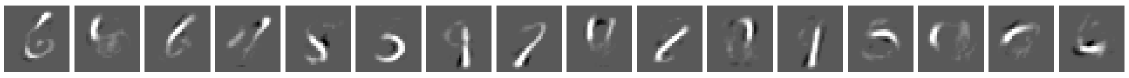


Figure 7.5: A subset of the resulting bases when SOAE is applied to the jittered MNIST learning set in unsupervised operating mode.

bases is shown in Figure 7.5. Similar to the experiments on the natural image patches, SOAE produces morphologically different filters than SCFC in the unsupervised setting. While the SCFC filters resembled entire digits, see Figure 5.7a, where all entries were either non-negative or non-positive, SOAE generates filters which possess non-vanishing entries that are both positive and negative. Again, the explanation for this phenomenon lies in the online nature of SOAE, which simplifies filter adaptation to individual samples of the learning set.

After these preliminary observations on SOAE's capabilities when no class labels are involved, SOAE was used to train classifiers for handwritten digit recognition using the optimization strategy proposed in Section 7.2. A comprehensive series of experiments described in the following was carried out using the jittered MNIST learning set with 540 000 samples. At the end of this section, results obtained with the elastically distorted learning set based on the outcome of the experiments on the jittered samples are presented.

Three variants of the SOAE architecture were evaluated, denoted by SOAE-σ-SC, SOAE-σ-DC and SOAE-L_0-SC, respectively. The number of hidden units was set to one thousand for all variants since this number was also used for the SCFC and SMLP experiments. For

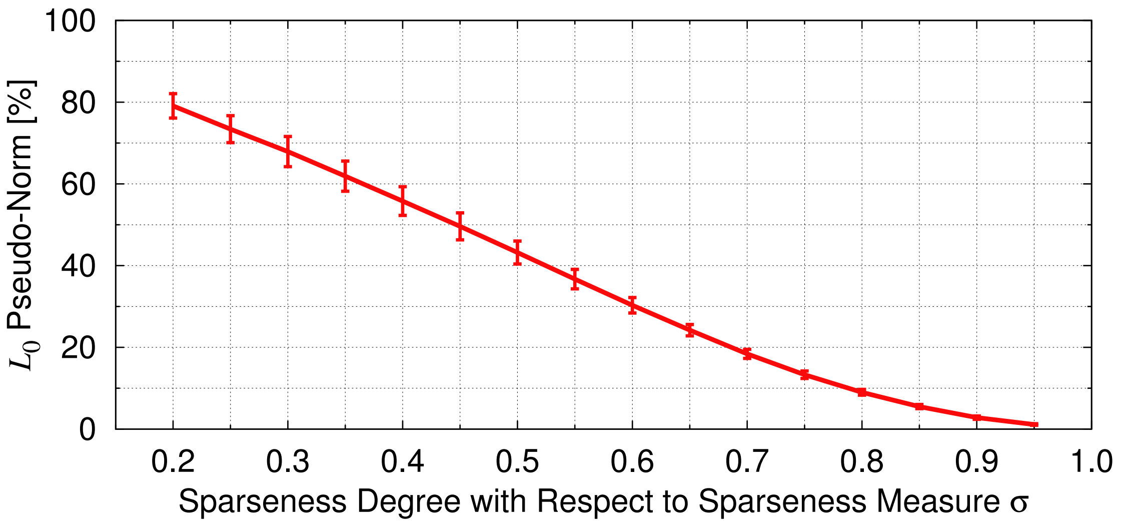


Figure 7.6: Resulting amount of nonzero entries in an internal representation h with 1000 entries of variant SOAE-σ-SC depending on the activity sparseness σ_H. For low values of σ_H, about 80% of the entries are nonzero, whereas for very high sparseness degrees only 1% of the entries do not vanish. Error bars indicate ± one standard deviation distance from the mean. The results of SOAE-σ-DC deviated by less than 0.3% from the results of SOAE-σ-SC and have been omitted in the plot to prevent clutter. Adapted from [66].

SOAE-σ-SC and SOAE-σ-DC, the sparseness projection with respect to Hoyer's sparseness measure σ was used as transfer function in the hidden layer. The activity sparseness σ_H was varied from 0.20 to 0.95 in steps of size 0.05 for both approaches. Sparseness of connectivity with a sparseness degree of $\sigma_W := 0.75$ was incorporated for SOAE-σ-SC, while SOAE-σ-DC only featured dense connectivity. This is reflected by the suffixes of the names of these variants. The concrete value for σ_W was chosen because it led to the best results for SCFC and SMLPs. The third variant, SOAE-L_0-SC, also featured sparse connectivity with $\sigma_W := 0.75$, and, more importantly, the projection onto an L_0 pseudo-norm constraint as transfer function in the hidden layer. The parameter that controls the exact number of hidden units allowed to be active at any one time was set to match the mean sparseness of activity SOAE-σ-SC achieved, as will be discussed below. The step size for gradient descent was determined through cross-validation as 0.1, which is higher than for the other approaches since here no normalization of the learning samples was carried out as this is not necessary for the SOAE architecture.

SOAE-σ-SC and SOAE-σ-DC were evaluated first. To compensate for the effect of random selection of samples for initialization and training, eight optimization runs were performed for each value of σ_H in the feasible range $\{0.20, 0.25, \ldots, 0.90, 0.95\}$. The resulting classifiers were applied to the evaluation set to compute the classification error, and the best and worst classifier were discarded to yield a trimmed sample set with six classifiers for each sparseness

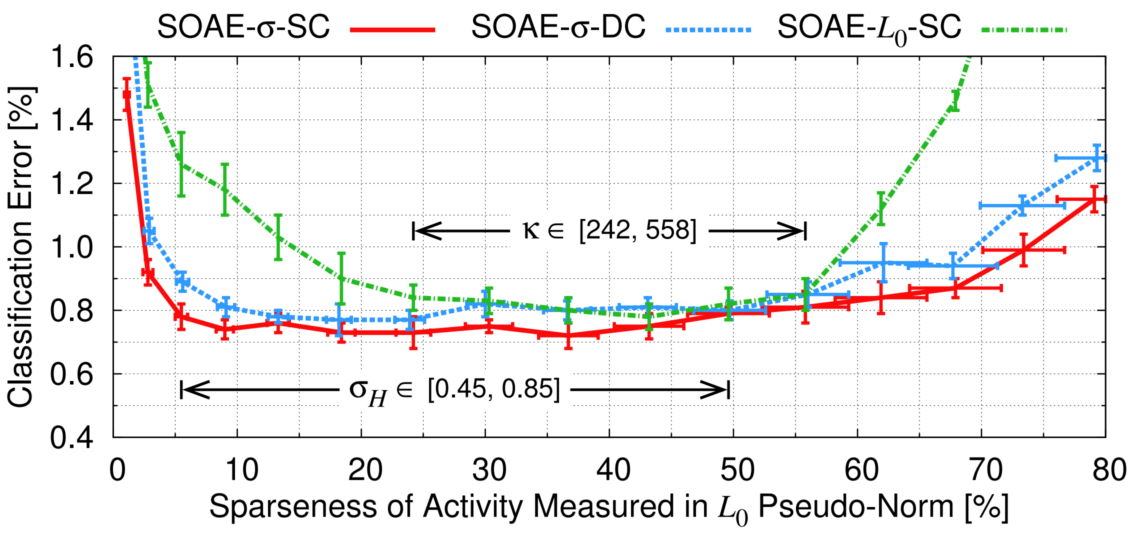


Figure 7.7: Resulting classification error on the MNIST evaluation set of the three variants SOAE-σ-SC, SOAE-σ-DC and SOAE-L_0-SC, depending on the sparseness of activity in the hidden layer. The error bars mark $\pm$ one standard deviation from the mean, both for the classification error and for the activity sparseness measured in L_0 pseudo-norm for variants SOAE-σ-SC and SOAE-σ-DC. SOAE-σ-SC clearly outperforms the other approaches and also achieves competitive classification results even for very sparse activity. The two marked intervals correspond to sparseness degrees where SOAE-σ-SC and SOAE-L_0-SC achieve almost constant generalization performance, respectively. Modified from [66].

degree. To determine reasonable values of κ for SOAE-L_0-SC, the number of active hidden units was counted when SOAE-σ-SC was applied to the jittered learning set. The measured dependence of this number on σ_H is depicted in Figure 7.6. The numbers for SOAE-σ-DC were very similar; the maximum deviation was less than 0.3%, which shows that the shown relationship is quite deterministic and accurate. The relationship between the sparseness degree and the number of active hidden units is clearly reciprocal and almost linear for $\sigma_H \in [0.20,\ 0.70]$. Approximately 80% of all hidden units are active upon sample presentation for the low setting of $\sigma_H = 0.20$. For $\sigma_H = 0.80$, this reduces to 10%, and 1% of activity is achieved for $\sigma_H = 0.95$. Further, the standard deviation of the number of active units decreases when σ_H increases, such that there is less variability in activity sparseness for large values of σ_H.

The mean values given in Figure 7.6 were used to parameterize κ for SOAE-L_0-SC, which here denotes the exact number of active units in the hidden layer. Again, eight classifiers were trained for each value of κ, and the ones with maximum and minimum error on the evaluation set were trimmed away. The resulting median classification errors and their standard deviation for the three variants are shown in Figure 7.7, depending on the percentage of active units in the hidden layer. Detailed results are also given in Table 7.1.

Table 7.1: Raw data achieved by carrying out three variants of SOAE on the jittered MNIST learning set. The infixes "σ" and "L_0" indicate that a projection onto a sparseness constraint given by either Hoyer's sparseness measure σ or the L_0 pseudo-norm was implemented as neural transfer function, respectively. The suffixes "SC" and "DC" denote sparse connectivity and dense connectivity, respectively. The numbers in the second column give the mean value $\pm$ one standard deviation of the activity rate of the SOAE-σ-SC experiments. These mean values were used to parameterize the target sparseness of SOAE-L_0-SC. The activity rates of SOAE-σ-DC were very close to those of SOAE-L_0-SC, the deviation was always less than 0.3%, so they were omitted in the table.

Target degree of activity sparseness σ_H	Number of active units in hidden layer κ [%]	Evaluation error SOAE-σ-SC [%]	Evaluation error SOAE-σ-DC [%]	Evaluation error SOAE-L_0-SC [%]
0.20	79.1 ± 3.0	1.15 ± 0.04	1.28 ± 0.04	4.30 ± 0.10
0.25	73.4 ± 3.3	0.99 ± 0.05	1.13 ± 0.03	2.14 ± 0.07
0.30	67.9 ± 3.7	0.87 ± 0.03	0.94 ± 0.04	1.46 ± 0.03
0.35	61.9 ± 3.7	0.84 ± 0.05	0.95 ± 0.06	1.12 ± 0.05
0.40	55.8 ± 3.5	0.81 ± 0.05	0.85 ± 0.04	0.85 ± 0.05
0.45	49.6 ± 3.3	0.79 ± 0.02	0.80 ± 0.03	0.82 ± 0.05
0.50	43.2 ± 2.8	0.75 ± 0.04	0.81 ± 0.03	0.78 ± 0.04
0.55	36.7 ± 2.4	0.72 ± 0.04	0.80 ± 0.03	0.80 ± 0.04
0.60	30.3 ± 1.9	0.75 ± 0.02	0.82 ± 0.04	0.83 ± 0.04
0.65	24.2 ± 1.4	0.73 ± 0.05	0.77 ± 0.03	0.84 ± 0.04
0.70	18.4 ± 1.1	0.73 ± 0.03	0.77 ± 0.05	0.90 ± 0.08
0.75	13.3 ± 0.9	0.76 ± 0.03	0.78 ± 0.02	1.03 ± 0.07
0.80	9.0 ± 0.7	0.74 ± 0.03	0.81 ± 0.03	1.18 ± 0.08
0.85	5.5 ± 0.5	0.78 ± 0.04	0.89 ± 0.03	1.26 ± 0.10
0.90	2.8 ± 0.4	0.92 ± 0.04	1.05 ± 0.04	1.51 ± 0.07
0.95	1.1 ± 0.2	1.48 ± 0.05	1.89 ± 0.05	2.18 ± 0.15

The abscissa in Figure 7.7 was transformed for SOAE-σ-SC and SOAE-σ-DC from Hoyer's sparseness measure to the L_0 pseudo-norm to facilitate comparison based on a unified sparseness measure. Clearly, SOAE-σ-SC performs best on the entire range of activity sparseness. The generalization performance is about equal for $\sigma_H \in [0.45,\ 0.85]$, translating roughly to between 5% and 50% of active hidden units. Classification capabilities degrade quickly for higher sparseness degrees, and they degrade gradually for lower sparseness degrees. To assess the mean performance of SOAE-σ-SC and SOAE-σ-DC, their results for $\sigma_H \in [0.45,\ 0.85]$ were pooled and then analyzed further. Since six classifiers were available for each distinct value of σ_H, this yielded a sample of size 54 for each variant. Subsequently, the best four and worst three classifiers were trimmed away to yield 47 classification results.

By means of these statistical series, it was deduced that SOAE-σ-SC yields an error rate of $0.75\% \pm 0.04\%$, and SOAE-σ-DC achieved an error rate of $0.81\% \pm 0.04\%$. Comparing this with a sparsely connected MLP with $\sigma_W = 0.75$ which achieved an error of $0.81\% \pm 0.05\%$, see Table 6.3, variant SOAE-σ-DC performs as well and SOAE-σ-SC performs considerably better, the improvement being greater than one standard deviation. The mean connectivity rate in the hidden layer of SOAE-σ-SC was 13.6%, which is higher than the 12.1% of the sparsely connected MLPs with $\sigma_W = 0.75$ since none of the weights in the hidden layer were pruned. When sparse connectivity is not enforced by a sparseness projection of the weights in the hidden layer, classification results are worse. Though, SOAE-σ-DC almost attained the performance of SOAE-σ-SC.

The third variant, SOAE-L_0-SC, performed worst of all three approaches. The optimum range of the sparseness degree κ is between 242 and 558 active hidden units, where nearly equivalent performance is achieved. Since the generalization performance degrades significantly when less than approximately 25% of all units are active, the approach is not well suited to an activity which actually is sparse. As the interval $\kappa \in [242,\ 558]$ corresponds to only six different sparseness degrees, additional optimization runs were carried out to achieve meaningful pooled results. Eight distinct classifiers were already available for each value of κ, so one extra training was sufficient to yield 54 error rates, of which seven were trimmed away to yield a sample of size 47.

This data reveals that SOAE-L_0-SC achieved an error rate of $0.82\% \pm 0.05\%$, comparable to the performance of SOAE-σ-DC and the sparsely connected MLPs. In this setting, there is hence no advantage in using a projection onto L_0 constraints rather than an ordinary local transfer function. Since SOAE-σ-SC achieved considerably better results, the sparseness-enforcing projection operator can be considered a better transfer function than the simple projection onto L_0 constraints. Because constraints on the exact number of active hidden units constitute a very hard optimization problem, using Hoyer's sparseness measure simplifies finding suitable optima of the objective function since it can be considered a smooth relaxation. This is in line with the properties of both sparseness measures and their impact on problem relaxation discussed in Section 2.2.

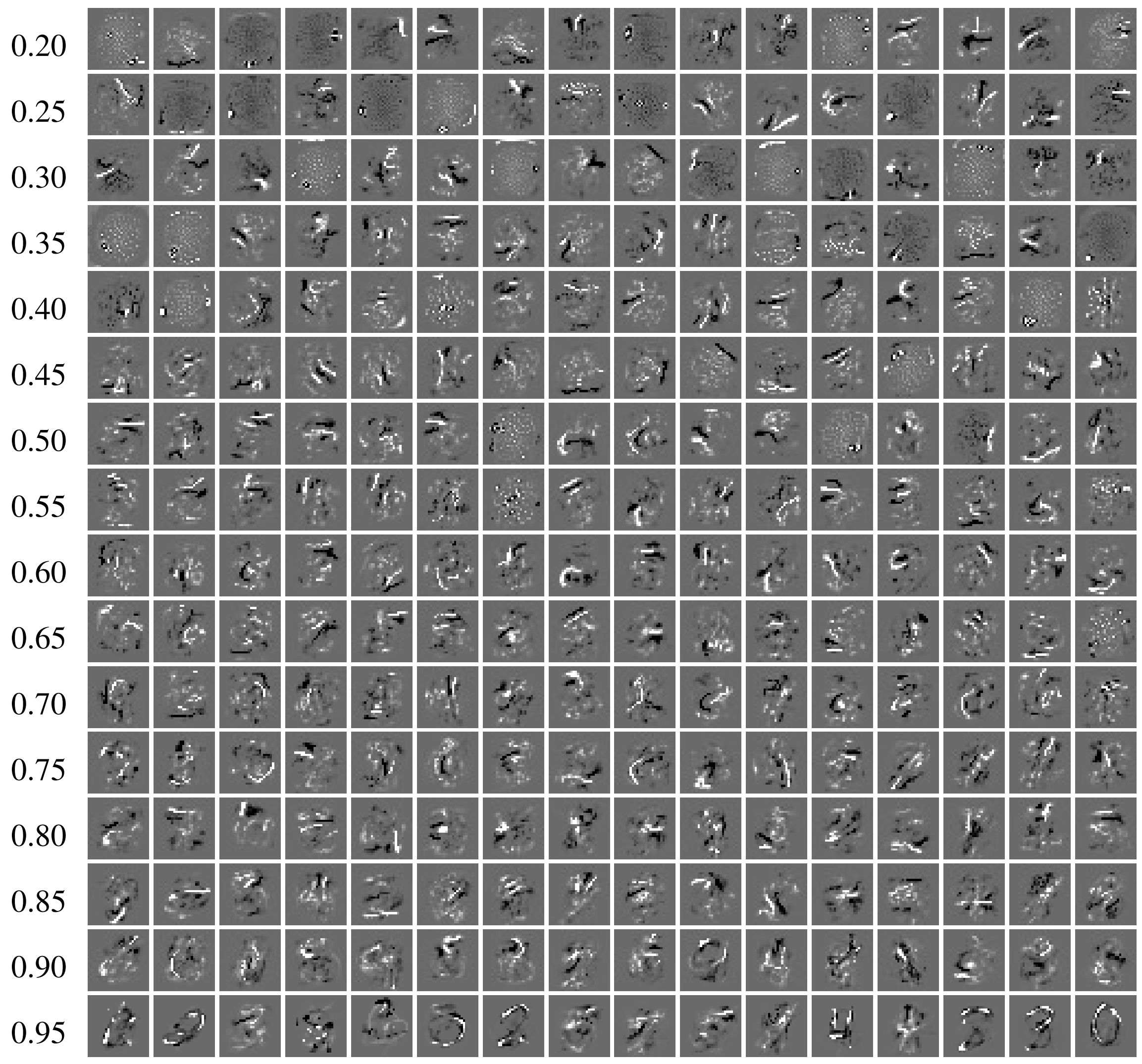


Figure 7.8: Subsets of the resulting bases of variant SOAE-σ-SC applied to the jittered MNIST learning set. The numbers next to the rows denote the target degree of sparse activity σ_H, achieved using the projection onto a constraint determined by Hoyer's sparseness measure σ as neural transfer function. The target degree of sparse connectivity was set to $\sigma_W := 0.75$ for all experiments.

Table 7.2: Results for SOAE-σ-SC applied to the elastically distorted MNIST learning set.

Sparseness degree σ_H	0.50	0.60	0.70	0.80
Evaluation error [%]	0.63 ± 0.02	0.59 ± 0.03	0.58 ± 0.03	0.55 ± 0.03
Active hidden units [%]	43.7 ± 3.4	30.2 ± 2.0	18.2 ± 1.0	9.2 ± 0.7

A subset of the resulting bases of variant SOAE-σ-SC is shown in Figure 7.8 for each individual value of σ_H. Note that the target degree of sparse connectivity σ_W was kept fixed at 0.75 throughout the experiments. At first glance, the filters look much more distorted than those obtained with sparsely connected MLPs, see Figure 6.5. When small values for σ_H were chosen, a large number of filters exhibited no particular structure. On the other hand, for $\sigma_H := 0.95$ there are filters that look like distorted prototypes of the samples from the learning set. However, a large percentage of the filters show the same phenomenon observed in the SCFC and SMLP experiments, namely the emergence of local contrast fields, albeit overlaid with random noise. Since the filters that stem from SOAE's unsupervised operating mode, depicted in Figure 7.5, are not this noisy, it can be concluded that teacher signal incorporation is responsible for the random noise in the bases of the supervised trainings. Because the mapping to a classification decision is much more complicated than just reproducing the input, it is not surprising that random noise is present in the bases of the classifiers.

A statistical analysis in Chapter 8 will discuss the benefit of sparse activity in greater detail. In the remainder of this section, results of SOAE-σ-SC applied to the elastically distorted MNIST learning set with 13.5 million samples are reported. For this, the number of hidden units and the degree of connectivity sparseness were chosen as before, and the activity sparseness was selected from the set $\{0.50, 0.60, 0.70, 0.80\}$ which lies within the interval where the best results were obtained on the jittered learning set. For each value of σ_H, eight classifiers were trained and the best and worst classifiers were removed to yield six distinct classifiers. This procedure led to the results given in Table 7.2. Here, generalization performance increases with the degree of sparse activity, the lowest classification error of 0.55% is obtained for $\sigma_H = 0.80$. This is a slight improvement over the 0.58% error rate achieved with sparsely connected MLPs on the same data set. When the results of all twenty-four classifiers are pooled, a mean error of $0.59\% \pm 0.04\%$ is achieved, which is statistically equivalent to the results of the SMLP algorithm. Further, 16.8% of all synaptic connections in the hidden layer were still active, which is more than the 14.3% connectivity rate of the sparsely connected MLPs. Hence on the elastically distorted samples no benefit of SOAE over SMLPs can be observed, which may be due to the number of hidden units being too small for so many training examples when sparse activity is involved. The generalization capabilities do not degrade either, so SOAE can cope easily with very large data sets. It should be noted, though, that the evaluation speed is slightly slower compared to SMLPs because carrying out a projection during the recall phase is slower than simply applying a local, nonlinear transfer function. A precise analysis of the computational complexity is given in Chapter 8.

Table 7.3: Measured mean lifetime sparseness $\mathbb{E}(\sigma(H))$ and its approximation $\widehat{\mathbb{E}}(\sigma(H))$ for different values of the target activity sparseness σ_H. The absolute deviation Δ is very small, hence the approximation is accurate.

σ_H	0.200	0.300	0.400	0.500	0.600	0.700	0.800	0.900
$\mathbb{E}(\sigma(H))$	0.198	0.295	0.391	0.488	0.585	0.682	0.783	0.887
$\widehat{\mathbb{E}}(\sigma(H))$	0.194	0.291	0.388	0.485	0.582	0.679	0.776	0.873
Δ	0.004	0.004	0.003	0.003	0.003	0.004	0.007	0.014

7.3.3 Lifetime Sparseness of the Neurons

The major difference between SCFC and SOAE lies in the notion of sparse activity. While in SCFC the lifetime sparseness of individual neurons is controlled explicitly, in SOAE the population sparseness is controlled. This measures the sparseness of the entire neuronal population at any one time [80]. It was demonstrated both theoretically and empirically in Section 5.1.1 that both concepts are closely related in the SCFC framework. This was deduced from the observation that the mean L_1 norm and the mean squared L_2 norm over all code words can be computed by changing the order of summation when summing up the entries of the matrix that contains all the code words. Since the L_1 norm and the L_2 norm of the lifetime activation of each neuron is already determined by the target sparseness degree, this allows linking lifetime sparseness to population sparseness. By approximation of occurring nonlinearities, it was further possible to infer a simple mathematical expression which was shown experimentally to be highly accurate on the MNIST data set.

In the SOAE architecture, the sparseness-enforcing projection operator is used to gain internal representations with an explicit sparseness degree with respect to Hoyer's sparseness measure, which depends primarily on the L_1 norm and the L_2 norm of its argument. Therefore, the theoretical results of Section 5.1.1 are applicable to SOAE as well, albeit from another point of view. Given a data set with M samples, the mean value of lifetime sparseness of each of the n neuronal units can be approximated by

$$\widehat{\mathbb{E}}(\sigma(H)) := \sigma_H \frac{\sqrt{n}-1}{\sqrt{n}-\sqrt{n/M}}.$$

This expression should not be confused with $\widehat{\mathbb{E}}(\sigma(h))$ which denoted the mean population sparseness in Section 5.1.1. To verify this approximation, the samples of the jittered MNIST learning set where $M = 540\,000$ were fed to the classifiers with $n = 1000$ from the SOAE-σ-SC experiments described in Section 7.3.2, the internal representations were inferred and written in a matrix next to each other. Then the empirical lifetime sparseness $\mathbb{E}(\sigma(H))$ was measured directly from this matrix, compared with the approximation $\widehat{\mathbb{E}}(\sigma(H))$ and the absolute error $\Delta := \left|\mathbb{E}(\sigma(H)) - \widehat{\mathbb{E}}(\sigma(H))\right|$ was computed. The numerical results of this experiment are given in Table 7.3 for various values of the target sparseness degree σ_H. The deviation Δ is always less than 0.014, therefore the approximation is very accurate. Further, for the values for M and n described above, it is $\widehat{\mathbb{E}}(\sigma(H)) \approx 0.97 \cdot \sigma_H$, which shows that just by selecting the population sparseness σ_H already a lifetime sparseness is determined which numerically does not deviate much from σ_H.

7.4 Discussion

A new architecture, called Supervised Online Auto-Encoder, which combines sparse activity and sparse connectivity in a mathematically elegant manner was proposed in this chapter. Sparseness of the internal representation can be effectively achieved by implementing the projection onto sparseness constraints as neural transfer function. This projection should be differentiable to facilitate parameter tuning through gradient-based methods. By projection of the synaptic weights onto another set of sparseness constraints during learning, sparse connectivity can be yielded as by-product. Because inference of sparsely populated activation vectors requires only a matrix-vector product followed by a vector-valued transfer function, memory and time requirements are very low and at most dominated by the notion of the concrete sparseness constraints.

Here, the L_0 pseudo-norm and Hoyer's σ were used as formal sparseness measures. Projections onto constraints induced by them have been discussed at length in Chapter 4. Since the latter can be considered a smooth approximation to the former sparseness measure, it was intuitively plausible that the sparseness-enforcing projection operator would yield better results than the simple L_0 projection. This was ultimately confirmed by experiments on the MNIST database of handwritten digits, which was used to benchmark generalization capabilities in a comprehensive series of experiments. It was further demonstrated that the combination of sparse activity and sparse connectivity, both assessed with respect to Hoyer's sparseness measure σ, is superior to omitting any of these key properties on the jittered MNIST samples. The classification performance when training on the very large data set with elastically distorted samples was slightly improved for a high degree of activity sparseness in comparison with the results obtained with sparsely connected MLPs. A comprehensive analysis is included in the next chapter, which irrevocably shows that using sparseness in neural networks results in a reduction of both the number of misclassifications and the computational resources needed for the computation of classification decisions.

8 Evaluation and Comparison

New models for sparse information processing in Artificial Neural Networks were proposed in the previous chapters. They featured the two key properties sparse activity and sparse connectivity. Sparse activity means that only few neurons are active at any one time. Sparse connectivity means that each neuronal unit receives input from only a limited number of other units. It could be observed that both sparseness notions lead to a boost in generalization performance compared to non-sparse methods. When sparse connectivity is involved, the states of only a fraction of all other neurons need to be considered for the computation of the output state of a neuron. Intuitively, inference and in particular the estimation of class membership in classification tasks are more efficient when sparse connectivity is exploited.

This chapter analyzes the boost in generalization performance and run-time efficiency in greater detail using the MNIST database of handwritten digits as the benchmark data set, see Chapter 3 for an introduction. A statistical analysis of the results of the proposed classification algorithms is performed to identify individual factors of the improved classification capabilities. Further, numerous classification methods from the literature that have been applied to handwritten digit recognition are evaluated in terms of the achieved classification accuracy and the computational cost required during the recall phase. As the MNIST data set is very popular, there are many different approaches available for comparison, including Convolutional Neural Networks, Support Vector Machines, Polynomial Classifiers and boosting algorithms, and additionally the algorithms proposed in this work.

The evaluation itself enhances the state of the art, since in the literature algorithms are only compared with each other based on the classification accuracy they achieve. By also considering the computational demands, a second dimension is added which facilitates the assessment of the properties of the respective algorithms, in particular the hardware cost associated with porting classifiers to consumer hardware under the constraint of a real-time capable recall phase.

The remainder of this chapter is organized as follows. First, a comprehensive statistical analysis is performed to demonstrate the significance of sparseness in neural networks. Then, a model of computation is defined which enables the comparison of the computational complexity of different algorithms. The determination of the cost of a variety of key operations such as matrix-vector multiplications simplifies the composition of the total cost of more complex algorithms. Next, selected approaches for classification are discussed and put in context with one another and the approaches proposed in this work. The chapter is concluded with a summary of the analysis and a discussion. This chapter contains material previously published in [101, 66].

8.1 Statistical Analysis

In this section a statistical analysis is performed to evaluate the significance of performance differences among selected approaches. The goal is to analyze the impact of the two key properties of Sparse Neural Networks, sparse activity and sparse connectivity, on the generalization performance. The raw data of the analysis consists of the results of the experiments on the jittered MNIST learning set carried out in Chapter 6 and Chapter 7.

In total, eight different approaches are available, where each approach has a sample of size 47 representing the distribution of the achieved evaluation errors. First, the individual approaches are briefly recapitulated. Then, hypothesis testing procedures proposed by [108, 109] are applied to gain a ranking of the approaches subject to statistical significance. Ultimately, an effect size estimation as proposed by [110, 111] is carried out to conclude the analysis.

The eight approaches analyzed here are described in the following and given a short name. From Chapter 6, there is the conventional MLP trained using randomly initialized weights which achieved an error of $0.88\% \pm 0.03\%$, here denoted as MLP-random. When the weights are initialized by replicating training samples rather than using random weights, an error rate of $0.91\% \pm 0.05\%$ results. This variant is called MLP-samples. If the SCFC algorithm is used instead, the results are almost equal, with an error of $0.91\% \pm 0.06\%$. This result is denoted by MLP-SCFC in the following.

The generalization performance was improved by incorporation of sparse connectivity. The sparsely connected MLP, pre-trained using SCFC, fine-tuned using projected gradient descent and post-processed to yield maximum connectivity sparseness, achieved an error of $0.81\% \pm 0.05\%$ on the MNIST evaluation set when the target degree of sparse connectivity was set to $\sigma_W = 0.75$. This approach is hereafter identified as SMLP-SCFC.

Another way of incorporating sparse connectivity is the Optimal Brain Damage method, where training and pruning of irrelevant synaptic connections are alternated until a target connectivity rate is reached. The connectivity rate that reflects $\sigma_W = 0.75$ best is 12.5%, where the approach denoted by MLP-OBD achieved an evaluation error of $0.89\% \pm 0.04\%$, which is comparable to MLP-random.

In Chapter 7, the MLP model was enriched by sparse activity using a sparseness projection as transfer function. Variant SOAE-σ-SC also featured sparse connectivity and achieved the best error of $0.75\% \pm 0.04\%$ among all approaches discussed here. Worse results were obtained when sparse connectivity was not incorporated. This is reflected by variant SOAE-σ-DC,

Table 8.1: Overview of the eight approaches subject to the statistical analysis. See the text for a description of each approach. The fourth column gives the result of the Shapiro-Wilk test for normality. Since all p-values are relatively large, it cannot be rejected that the algorithm outcomes follow a normal distribution.

Approach	Sparse Connectivity	Sparse Activity	Result $(W,\ p)$ of Shapiro-Wilk Test	Evaluation Error [%]
SOAE-σ-SC	yes	yes	(0.9802, 0.60)	0.75 ± 0.04
SOAE-σ-DC	no	yes	(0.9747, 0.40)	0.81 ± 0.04
SOAE-L_0-SC	yes	yes	(0.9786, 0.53)	0.82 ± 0.05
SMLP-SCFC	yes	no	(0.9770, 0.47)	0.81 ± 0.05
MLP-OBD	yes	no	(0.9807, 0.62)	0.89 ± 0.04
MLP-random	no	no	(0.9798, 0.58)	0.88 ± 0.03
MLP-samples	no	no	(0.9773, 0.49)	0.91 ± 0.05
MLP-SCFC	no	no	(0.9794, 0.57)	0.91 ± 0.06

which yielded an error rate of $0.81\% \pm 0.04\%$. It was further evaluated whether a simple L_0 pseudo-norm projection would perform competitively. The answer was negative, since variant SOAE-L_0-SC only achieved an error of $0.82\% \pm 0.05\%$, which is worse than what could be achieved through the σ projection.

An overview of the eight approaches is given in Table 8.1, together with their evaluation error, and the information whether they feature sparse activity or sparse connectivity. At first, all algorithm outcomes were tested for normality using the Shapiro-Wilk test [112]. The test statistic W and the p-value of this test for each algorithm is included in Table 8.1. Since the minimum p-value of all approaches is 0.40, it cannot be rejected that the algorithm outcomes follow a normal distribution.

It was subsequently tested whether the variances of the results of the eight algorithms were equal, or in statistical terms homoscedastic. The Levene test [113] was used, which resulted in a test statistic $F = 2.7979$ with 7 and 368 degrees of freedom, implying a p-value of 0.0075. Homoscedasticity can hence be rejected with very high probability, and parametric omnibus tests and post-hoc tests that require the groups to be normally distributed with identical variance are not applicable [109].

Instead, nonparametric tests based on the ranking of the algorithm outcomes can be used. The Kruskal-Wallis test [114] was used as an omnibus test to determine whether all algorithms had the same mean performance. Since the test statistic was $H = 214.44$ with 7 degrees of freedom, indicating a p-value less than 10^{-15}, it was found that there is a statistically significant deviation in the mean performance of the eight approaches.

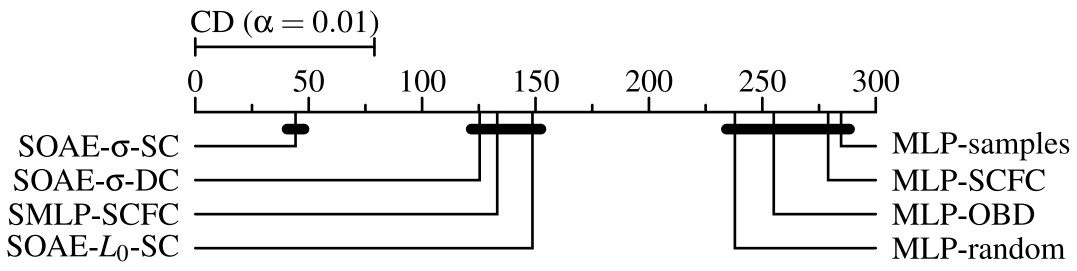


Figure 8.1: Demšar diagram to facilitate visual inspection of the results of a post-hoc test. Each algorithm has a mark at the mean rank of its results compared to all other algorithms. Two algorithms are considered to produce statistically different results if the difference between their mean ranks is greater than the critical difference (CD). The black bars mark three groups of algorithms which could be partitioned through the critical difference obtained at the $\alpha = 0.01$ significance level. Adapted from [66].

This deviation was then located using a post-hoc test suitable for the results of the Kruskal-Wallis test, which yielded the mean ranks of all eight approaches as a by-product. Using a Tukey-Kramer type modification of Dunn's procedure yields a critical difference at a given significance level [115]. When the difference between the mean ranks of two approaches is greater than this critical difference, these two approaches cannot result in equal performance. This can be visualized using Demšar diagrams [108], which were adapted here to the use case of unpaired observations. Such a diagram is depicted in Figure 8.1 for the eight approaches analyzed here, including the critical difference at the $\alpha = 0.01$ significance level. At this significance level, the eight approaches were partitioned into three groups A, B and C given by $A := \{$SOAE-σ-SC$\}$, $B := \{$SOAE-σ-DC, SOAE-L_0-SC, SMLP-SCFC$\}$, and $C := \{$MLP-OBD, MLP-random, MLP-samples, MLP-SCFC$\}$. The p-value for this partition is 0.007. If the significance level α would have been chosen lower than this, the groups A and B would blend together until there would no longer be a statistically significant difference between SOAE-σ-SC and SOAE-σ-DC, although the difference between their means is 1.5 standard deviations.

The effect size was assessed after this partition was obtained. This number describes the probability of superior outcome. In the context discussed here, this is the probability that a classifier trained using an algorithm from one of the groups has fewer misclassifications on the evaluation set than another classifier trained using an algorithm from another group. This method has applications in the medical sciences where the purpose is, for example, to determine if one substance is superior to another in the treatment of diseases [110, 111]. For this final analysis, the results from the three groups were pooled and tested for normality. Group A has already been confirmed to be normal for it is a singleton. Groups B and C achieved pairs of the Shapiro-Wilk test statistic W and p-value of (0.9845, 0.11) and (0.9882, 0.12), respectively.

Table 8.2: Summary of the statistical analysis at the $\alpha = 0.01$ significance level in terms of classification accuracy for trainings on the jittered MNIST learning set. Sparseness is always achieved here through a σ projection. Other approaches like using an L_0 projection or the OBD method do not contribute to performance boosts.

Sparse Connectivity	Sparse Activity	Generalization Performance
yes	yes	best
no	yes	medium
yes	no	medium
no	no	worst

Since it cannot be rejected that all three groups stem from a normal distribution, the parametric minimum variance unbiased estimator $\hat{R}_2$ of [116] could be employed to determine the effect size. In formal terms, if E_G models the distribution of the evaluation errors of an algorithm from a group $G \in \{A, B, C\}$, then $\hat{R}_2$ estimates the probability $P(E_G < E_{\tilde{G}})$ that an algorithm from group G achieves better generalization performance than an algorithm from group $\tilde{G}$. Evaluation of the estimator $\hat{R}_2$ shows that $P(E_A < E_B)$, $P(E_B < E_C)$ and $P(E_A < E_C)$ can be estimated by 0.87, 0.88 and 0.99, respectively. A simple nonparametric alternative for effect size estimation is normalizing the Mann-Whitney U statistic by the number of samples [110, 111], which deviated less than 1% from $\hat{R}_2$ on the concrete data.

These results can be interpreted, for example, as SOAE-σ-SC producing a better classifier than MLP-random in 99% of all cases. Analogously, the SMLP training algorithm produces a better classifier than the conventional MLP training algorithm in 88% of all cases. In both cases, the effect size can be considered more than just large [110, 111].

With respect to the two key properties of sparse activity and sparse connectivity, the results of the statistical analysis show that the combination of both performs best when the smooth sparseness measure σ of Hoyer is used, see Table 8.2. Midrange generalization capabilities can be achieved when either sparse activity or sparse connectivity is incorporated.

One exception is Optimal Brain Damage, which provides results statistically equivalent to the situation when no sparseness whatsoever is involved, which are the worst results in the entire analysis. Because OBD is a destructive algorithm that cannot recover from inadvertently removed synaptic connections, it is inherently inferior to projected gradient descent methods which do not suffer from this shortcoming. The other exception is SOAE-L_0-SC, which featured sparse connectivity through an L_0 pseudo-norm projection. The latter is a highly non-smooth function, and it is questionable whether good solutions to the concrete learning problem with hard constraints on the number of non-vanishing entries actually exist.

8.2 Model of Computation and Complexity of Elementary Operations

In pure complexity theory, the computational complexity of an algorithm is given using Landau symbols for Turing-complete models of computation [23]. Landau symbols have the drawback that constant factors are lost since only asymptotic estimates are reproduced. The constant factors are crucial in practice however since, for example, an algorithm that needs only a hundredth of the computational resources of another algorithm is preferable, even though both algorithms have the same asymptotic complexity. In the context of classification algorithms it is moreover important that the size of the classifier is taken into consideration. For example, when the number of hidden units in a Multi-Layer Perceptron is doubled, then the enlarged neural network takes about twice the time to classify one single sample. This cannot be heeded when Landau symbols are used.

Another measure of complexity would be to implement all algorithms on a suitable hardware and determine their run-time. The drawback of this approach is that conclusions based on it cannot be easily transferred to another type of hardware, as certain operations may or may not be more efficient depending on the machine characteristics. For example, some machines do not feature a dedicated multiplication unit, such that shift and add operations need to be used for the evaluation of elementary functions [117]. To be as independent as possible from such low-level details, it is hence proposed to use an abstract and Turing-complete model of computation and merely count the number of operations required to run certain algorithms.

In this work, an abstract model based on register machines is used. Register machines provide elementary arithmetic operations, control flow facilities and random access to an infinite amount of registers [23]. In order to better reflect the design of modern computers, this model is altered as follows. A multiply-accumulate (MAC) operation is introduced which computes the expression $x := x + y \cdot z$, where x, y and z are real numbers. This is a common operation in signal processing and can be carried out in one cycle on most modern machines. The cost of all other computations is measured relative to the cost for carrying out one MAC operation. This includes individual addition and multiplication operations, which are not faster than MAC operations provided the machine has a dedicated MAC unit.

In addition, the read and write accesses to main memory are counted and included in the analysis. The use of a very limited number of registers not involving additional memory accesses is allowed to provide, for example, accumulation registers which are useful in matrix-vector computations. This is a modification of the original register machines encountered in complexity theory where infinitely many registers are available, but this property is not realistic in real-world hardware. Mechanisms such as cache hierarchies, out-of-order execution and pipelining are neglected in the analysis as they are too machine-specific to provide general conclusions.

Algorithm 8.1: Canonical computation of the general matrix-vector product.

Input: $A \in \mathbb{R}^{a \times b}$, $x \in \mathbb{R}^a$ and $y \in \mathbb{R}^b$.
Output: $y := y + A^T x \in \mathbb{R}^b$.

```
for $i := 1$ to $b$ do
    $D := y_i$;
    for $j := 1$ to $a$ do $D := D + A_{j,i} \cdot x_j$;
    $y_i := D$;
end
```

Algorithm 8.2: Canonical computation of the sparse matrix-vector product using the modified ELLPACK storage format for sparsely populated matrices.

Input: $A \in \mathbb{R}^{a \times b}$ given in the modified ELLPACK format via value matrix $V \in \mathbb{R}^{k \times b}$ and position matrix $P \in \mathbb{N}^{k \times b}$, $x \in \mathbb{R}^a$ and $y \in \mathbb{R}^b$.
Output: $y := y + A^T x \in \mathbb{R}^b$.

```
for $i := 1$ to $b$ do
    $D := y_i$;
    for $j := 1$ to $k$ do $D := D + V_{j,i} \cdot x_{P_{j,i}}$;
    $y_i := D$;
end
```

In the following, the complexity of a variety of elementary operations with respect to this model of computation is deduced. The results of this analysis can then be used to infer the computational complexity of more complex algorithms.

8.2.1 Operations from Linear Algebra

Simple operations involving vectors and matrices are considered first. This includes general matrix-vector products, the multiplication of a sparsely populated matrix with an arbitrary vector, and the computation of Euclidean distances.

General Matrix-Vector Product

This operation is frequently encountered in Multi-Layer Perceptrons, where a layer's input is multiplied with a weight matrix to compute dendritic potentials which in turn are used to derive the layer output through a transfer function. More formally, given a matrix $A \in \mathbb{R}^{a \times b}$ and a vector $x \in \mathbb{R}^a$, the vector $y := A^T x \in \mathbb{R}^b$ should be computed. A slight generalization thereof is additively updating a vector with the product, that is computation of $y := y + A^T x$. The

Algorithm 8.3: Canonical computation of the column-wise Euclidean distance between a matrix and a vector.

Input: $A \in \mathbb{R}^{a \times b}$ and $x \in \mathbb{R}^a$.

Output: $y := \left(\|Ae_1 - x\|_2^2,\ \ldots,\ \|Ae_b - x\|_2^2 \right)^T \in \mathbb{R}^b$.

```
1 for $i := 1$ to $b$ do
2     $D := 0$;
3     for $j := 1$ to $a$ do $E := A_{j,i} - x_j$;  $D := D + E \cdot E$;
4     $y_i := D$;
5 end
```

canonical algorithm for this operation is given in Algorithm 8.1. Here, D is a register which is used to avoid having to store back and re-fetch the result of every MAC operation. For the initialization of D, b read accesses to memory are necessary, and another b write accesses are needed to write back the final values of D. In addition, ab MACs are carried out which need $2ab$ memory accesses for fetching the arguments. Therefore, there are ab MAC operations and $2b(a+1)$ memory accesses in total involved with this operation. Asymptotically more efficient algorithms than the canonical method are known, but they are not relevant for the problem size encountered here [23].

Sparse Matrix-Vector Product

Sparsely connected MLPs as described in Chapter 6 feature sparsely populated weight matrices, that is most of their entries are zero and thus irrelevant in the computation of matrix-vector products. As in Section 6.3.3, let $A \in \mathbb{R}^{a \times b}$ be sparsely populated with at most $k \in \mathbb{N}$ nonzero entries in each column, and let $V \in \mathbb{R}^{k \times b}$ be the matrix of values and $P \in \mathbb{N}^{k \times b}$ be the matrix of positions gained from converting A into the modified ELLPACK format. The canonical algorithm for the sparse matrix-vector product using this storage format is given in Algorithm 8.2. There are $2b$ memory access for reading and writing y. The matrix P is used to index the entries of the input vector x. The number of MAC operations reduces to kb, but for their arguments the entries of P have to be read as well, totaling in $3kb$ memory accesses. Summing up, kb MAC operations and $3b(k + 2/3)$ memory accesses are required for carrying out Algorithm 8.2.

Euclidean Distance between the Columns of a Matrix and a Vector

Matrix-vector products are employed in neural networks that use the dot product as similarity measure. In Radial Basis Function Networks, this is replaced with a Euclidean distance operation [53]. As can be seen from Proposition 1 in Chapter 4, there is a close relationship between both operations. In formal terms, given a matrix $A \in \mathbb{R}^{a \times b}$ and a vector $x \in \mathbb{R}^a$, the vector

$$y := \left(\|Ae_1 - x\|_2^2, \ \dots, \ \|Ae_b - x\|_2^2 \right)^T \in \mathbb{R}^b$$

should be computed. Algorithm 8.3 describes the canonical implementation which uses two registers D and E to reduce the number of memory accesses. The innermost computational kernel computes the squared difference between two values and adds the result to an accumulator. The complexity of this operation is comparable to an ordinary MAC operation, depending on the concrete machine's instruction set. Although the initialization of register D with zero does not necessarily involve a memory access on some machines, one is charged nevertheless. This way, the computational complexity is the same as that of a general matrix-vector product.

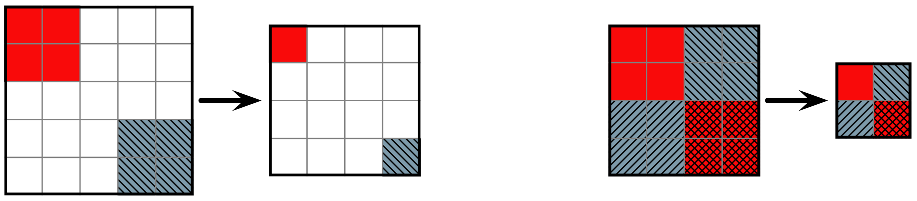
8.2.2 Convolutional Operations for Image Processing

The key feature of *Convolutional Neural Networks* is the use of small receptive fields which are applied in sliding window fashion to two-dimensional inputs [54]. This also includes the weight-sharing property, which states that the entries of one receptive field are fixed regardless of the concrete position in which the field is used to scan the input image. In other words, the central operation is a convolution [118], see Figure 8.2 for an illustration.

This operation also emerges in one-dimensional form in the computation of the product of two polynomials [23]. The analysis of the convolution operation is restricted here for simplicity to the two-dimensional form, as this is the only occurrence needed for classification of images. While naive evaluation of the convolution has increased asymptotic computational complexity compared to using integral transforms such as the Fourier transform [118, 23], direct evaluation may be preferred when the filter kernels are small for efficiency reasons on real hardware. Here, the different types of convolutions typically used for classification tasks are discussed.

Unit-Stride Convolution

Let $B \in \mathbb{R}^{p_h \times p_w}$ be an input image and let $w \in \mathbb{R}^{r_h \times r_w}$ be a filter kernel where $r_h \leq p_h$ and $r_w \leq p_w$. Let $C := B * w \in \mathbb{R}^{(p_h - r_h + 1) \times (p_w - r_w + 1)}$ denote the result of the convolution when only the valid region of the output is considered, that is when no boundary handling is performed. For each entry of C, a dot product of dimensionality $r_h r_w$ has to be carried out, see Figure 8.2. Neglecting that w may be stored in memory cache due to its size which may be small in real applications, the computational complexity is equal to that of a general matrix-vector product. Hence, $r_h r_w$ MAC operations and $2(r_h r_w + 1)$ memory accesses are required for one entry of C. The cost for computing the entire matrix C is then simply the cost for one entry times the number of entries.



(a) Unit-stride convolution.

(b) Convolution with stride two.

Figure 8.2: Convolution of an input image with a convolutional kernel of size 2×2 pixels. The shaded regions in the input image indicate different positions where a dot product is evaluated. The same shading is used in the output image to show where the corresponding scalar value is stored.

Arbitrary Stride Convolution

In this modification of the original convolution, the filter kernel is shifted by $s_y \in \mathbb{N}$ pixels down and $s_x \in \mathbb{N}$ pixels to the right after each evaluation, where $s_y, s_x \geq 1$. The common unit-stride convolution can be obtained by setting $s_y = s_x = 1$. To avoid special boundary handling, s_y must divide $p_h - r_h$ and s_x must divide $p_w - r_w$. The result of the convolution is then a matrix with $(p_h - r_h)/s_y + 1$ rows and $(p_w - r_w)/s_x + 1$ columns, where the cost for each entry is equal to that discussed in the unit-stride case.

Tensor Convolution

This operation is a generalization of the conventional convolution, where $d \in \mathbb{N}$ input images are convolved with the same number of filter kernels, and the resulting images are combined additively to yield the overall result. The tensor convolution is used for example to process color images which are separated into a fixed number of color channels, or to combine the output of several convolutions carried out with a multitude of kernels in an earlier layer of a Convolutional Neural Network. Let $B_1, \ldots, B_d \in \mathbb{R}^{p_h \times p_w}$ be the input images and let $w_1, \ldots, w_d \in \mathbb{R}^{r_h \times r_w}$ be the filter kernels. The output is then given by the matrix $C := \sum_{i=1}^{d} (B_i * w_i) \in \mathbb{R}^{(p_h - r_h + 1) \times (p_w - r_w + 1)}$.

For each entry of C, d dot products of size $r_h r_w$ have to be evaluated. The overall sum in the definition of C does not incur any additional costs when this operation is cast as the computation of a dot product of dimensionality $d r_h r_w$, as the results of previous dot products then do not need to be fetched from memory again compared to when d individual dot products are performed. Therefore, each entry of C takes $d r_h r_w$ MACs and $2(d r_h r_w + 1)$ memory accesses. The generalization of this operation to arbitrary strides is straightforward and hence omitted in this discussion.

Pooling and Subsampling

In order to reduce resolution and therefore enhance invariance against small translations and distortions, Convolutional Neural Networks also feature so-called pooling layers. Here, a single input image is transformed into an output image with a reduced number of rows and columns. The dimensionality reduction is achieved by computation of the empirical mean value of entries in small non-overlapping spatial neighborhoods, usually 2×2 pixels in size. Another notion uses the maximum entry within these neighborhoods, which is superior in terms of generalization capabilities to taking the average value on some data sets [119]. Average pooling is identical to a two-stride convolution with the kernel $\frac{1}{4}\left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$, which halves the dimensionality along both axes, see Figure 8.2.

Although this filter kernel is separable, the special 2×2 pixels filter size means it is not more efficient to perform successive horizontal and vertical convolutions using the factors of the kernel. As all entries of the filter kernel are equal, one might argue that three additions and one division are sufficient to compute one entry for the result image. However, performing the operation this way is as efficient as carrying out four MAC operations on machines with a dedicated MAC unit. In spite of this, the number of read memory accesses can be reduced to the four values that have to be fetched from the input image, exploiting the homogeneity of the filter kernel. Additionally, one write access to memory for each entry in the output image has to be accounted for. Since 2×2 pooling results in an image with ${}^{p_h p_w}\!/\!{}_4$ pixels, the overall cost comes to a total of $p_h p_w$ MAC operations and ${}^5\!/\!{}_4 \cdot p_h p_w$ memory accesses. For simplicity, pooling using the max operator is assumed to incur the same cost. The generalization to the case where $s_h \times s_w$ neighborhoods are used is straightforward, and concludes that the operation needs $p_h p_w$ MACs and ${}^{p_h p_w (s_h s_w + 1)}\!/\!{}_{s_h s_w}$ memory accesses.

8.2.3 Computation of Monomials up to a Specified Degree

The idea behind a *Polynomial Classifier* is to compute all possible monomials up to a target degree $g \in \mathbb{N}$, evaluate an input vector $x \in \mathbb{R}^n$ with respect to the monomials yielding a vector $h \in \mathbb{R}^p$, and then to linearly predict class membership using h as feature vector [120, 121], see also Figure 8.3. Here, the evaluation of all monomials is analyzed while the entire Polynomial Classifier is handled in Section 8.3.5. For example, the only monomial of degree zero is unity, all monomials of degree one are $x_1, \ldots, x_n$, and the monomials of degree two are given by $x_1^2, x_1x_2, x_1x_3, \ldots, x_1x_n, x_2^2, x_2x_3, \ldots, x_2x_n, \ldots, x_n^2$, that is the upper or lower triangle of the dyadic product xx^T. The number of all possible monomials in n variables where the degree is exactly $q \in \mathbb{N}$ is given by the binomial coefficient $\binom{n+q-1}{q}$. Hence, $p := \sum_{q=0}^{g} \binom{n+q-1}{q} = \binom{n+g}{g}$ is the number of all monomials in n variables with a degree less than or equal to g [120].

Given an input vector x, an arbitrary monomial with degree q can be computed using at most $\max(0,\ q-1)$ MAC operations. If $q = 0$, then the only monomial is unity, which can be

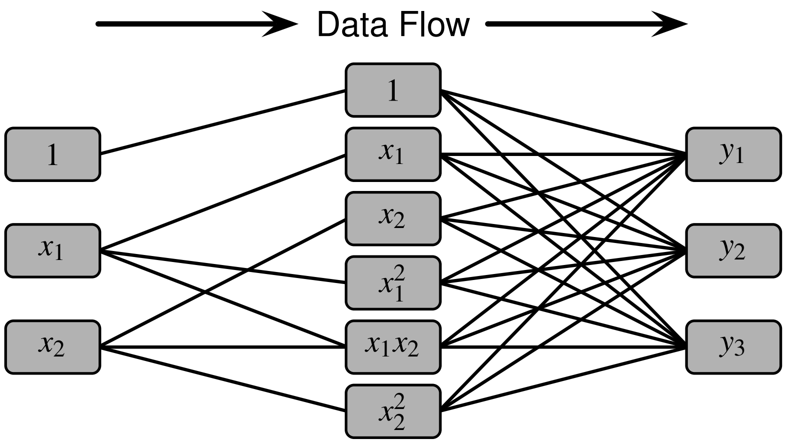


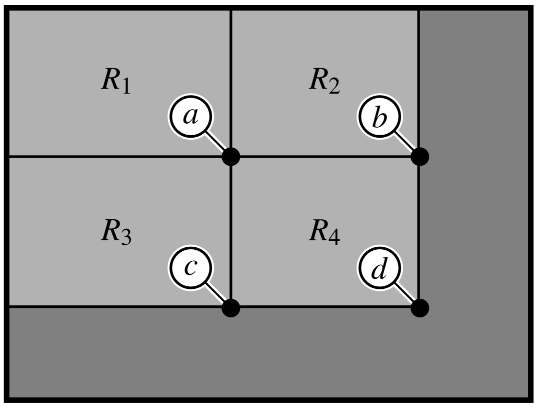
Figure 8.3: Principle of a quadratic Polynomial Classifier with two-dimensional input vectors and three-dimensional output vectors [121]. All possible monomials up to degree two are evaluated and then linearly combined to form the classification decision.

yielded without computations by simply loading a number from memory. In the case of $q = 1$ the monomials are already given by the entries of x, which can simply be copied into the output vector y. In the non-trivial case of $q > 1$, a product of q numbers has to be computed, which can be achieved with $q - 1$ MACs. In total, computation of the entire vector with the evaluated monomials then requires $\sum_{q=0}^{g} \max(0,\ q-1) \cdot \binom{n+q-1}{q} = \sum_{q=2}^{g} (q-1) \cdot \binom{n+q-1}{q}$ MAC operations.

The analysis of memory accesses uses a similar argument. For all monomials where the degree is exactly q, the total number of memory accesses is $\min(2,\ q+1) \cdot \binom{n+q-1}{q}$. When $q = 0$, one value has to be read and written to another location in memory. In the case of $q = 1$, the entire input vector has to be copied to another location, resulting in $2n$ memory accesses. For a monomial of degree $q > 1$, at most q values have to be read, the monomial evaluated, and the resulting scalar written back into memory. The monomials where not exactly q distinct entries have to be combined, for example x_1^2 and x_n^2, are few and for simplicity not treated individually. Therefore, there are $(q+1) \cdot \binom{n+q-1}{q}$ memory accesses in this case for all monomials. Summing up, $\sum_{q=0}^{g} \min(2,\ q+1) \cdot \binom{n+q-1}{q} = 2 + \sum_{q=1}^{g} (q+1) \cdot \binom{n+q-1}{q}$ memory operations are necessary for computation of the output vector.

8.2.4 Evaluation of Haar-Like Filters using an Integral Image

Given an arbitrary image, it is possible to compute the sum over all pixels lying in a rectangle of arbitrary size with a complexity independent of the actual rectangle size using the intermediate data structure known as *Integral Image* [122]. Each pixel in the Integral Image equals



$$
\begin{aligned}
a &= \textstyle\sum_{(u,v)\in R_1} A_{u,v} \\
b &= \textstyle\sum_{(u,v)\in R_1\cup R_2} A_{u,v} \\
c &= \textstyle\sum_{(u,v)\in R_1\cup R_3} A_{u,v} \\
d &= \textstyle\sum_{(u,v)\in R_1\cup R_2\cup R_3\cup R_4} A_{u,v} \\
&\Rightarrow \textstyle\sum_{(u,v)\in R_4} A_{u,v} = a+d-b-c
\end{aligned}
$$

Figure 8.4: Computation of the sum of pixel values in an arbitrary rectangle R_4 using an Integral Image [122]. The numbers a, b, c and d can be read directly from the Integral Image; they denote the sums over the rectangles as given in the formulas above. The sum with respect to R_4 can be deduced with arithmetic operations using these four numbers. The computational complexity is independent of the area of R_4.

the sum over all pixel values from the original image which are located to the top left of the Integral Image pixel position, see Figure 8.4. In formal terms, if $A \in \mathbb{R}^{a\times b}$ is an arbitrary image, then the Integral Image of A is a matrix $B \in \mathbb{R}^{(a+1)\times(b+1)}$ where $B_{i,j} := \sum_{u<i \text{ and } v<j} A_{u,v}$ for $i \in \{1,\dots,a+1\}$ and $j \in \{1,\dots,b+1\}$ [122]. Note that the dimensions of B have increased to produce a row and column that entirely vanish, which alleviates the need for special case handling when computing sums of pixels in rectangles that contain the topmost row or the leftmost column of A, respectively. Since B can be computed efficiently in one pass using simple recurrence relations [122], the complexity for computing the Integral Image is negligible.

Instead, the evaluation of *Haar-like filters* is analyzed in detail. These are rectangular filters which can be used to detect contrasts in images, for example those that occur at edges or junctions. Figure 8.5 shows a repertoire of five different types of Haar-like filters. They are commonly used in all possible scalings, including those that do not maintain the filter aspect ratio. Each filter response is defined to equal the sum of the pixel values corresponding to the white areas, minus the sum of pixel values belonging to the black areas. To determine the computational complexity of filter evaluation, first consider the procedure to determine pixel sums using an Integral Image. As seen in Figure 8.4, one rectangular sum can be computed using four read accesses to the Integral Image, and combining these four values in additive or subtractive operations, respectively.

Further, note that the actual Haar-like filters consist of adjacent regions, that is the white and black areas share edges. Because there are two numbers from the Integral Image associated with each edge, these two numbers do not have to be processed independently. Instead, the values can be directly kept and used to compute the sum of pixel values where the pixels

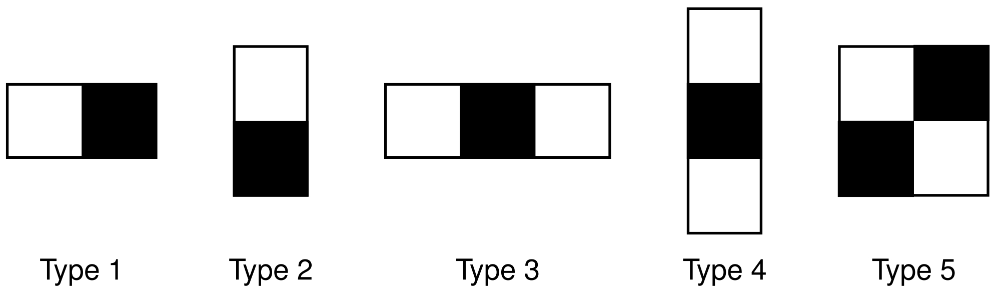
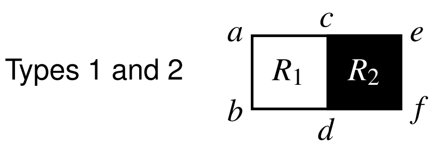
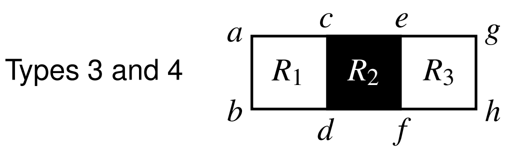


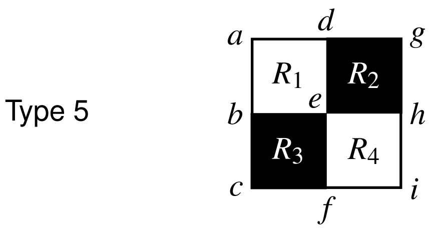
Figure 8.5: Five types of Haar-like filters for image processing [123]. The filter response is computed by subtracting the sum of the pixel values corresponding to the black areas from the sum of the pixel values corresponding to the white areas. The filters are used in all possible scales; the complexity of the filter evaluation is independent of the scaling when an Integral Image is used.



$$
\begin{aligned}
R &= \textstyle\sum_{(u,v)\in R_1} A_{u,v} - \sum_{(u,v)\in R_2} A_{u,v} \\
&= (a+d-b-c)-(c+f-d-e) \\
&= a-b-2(c-d)+e-f
\end{aligned}
$$



$$
\begin{aligned}
R &= \textstyle\sum_{(u,v)\in R_1\cup R_3} A_{u,v} - \sum_{(u,v)\in R_2} A_{u,v} \\
&= (a+d-b-c)+(e+h-f-g)-(c+f-d-e) \\
&= a-b-2(c-d)+2(e-f)-(g-h)
\end{aligned}
$$



$$
\begin{aligned}
R &= \textstyle\sum_{(u,v)\in R_1\cup R_4} A_{u,v} - \sum_{(u,v)\in R_2\cup R_3} A_{u,v} \\
&= (a+e-b-d)+(e+i-f-h) \\
&\quad -(d+h-e-g)-(b+f-c-e) \\
&= a+c+g+i-2(b+d+f+h)+4e
\end{aligned}
$$

Figure 8.6: Computational cost for Haar-like filter evaluation using edge sharing. The filter response R is given by the sum over the white regions minus the sum over the black regions. Using Integral Images, the values at the corners of the regions can be combined arithmetically to efficiently yield the filter response. For example, filters of type 1 or 2 need six values to be combined additively, which costs five MACs, six read accesses to memory, and one write access to store the filter response.

belong to adjacent areas. This is carried out formally in Figure 8.6, which shows that when n corner values have to be combined to yield the filter response, n read accesses to the Integral Image are necessary, followed by $n-1$ MAC operations and one memory write access to compute and store the filter response. According to Figure 8.6 it follows that $n=6$ for filters of type 1 and 2, $n=8$ for filters of type 3 and 4, and $n=9$ for filters of type 5.

8.3 Approaches for Handwritten Digit Recognition

This section reviews numerous learning algorithms whose performance results on the MNIST database of handwritten digits have been reported in the literature. Additionally, the methods developed in this work are evaluated considering their computational complexity during recall phase and compared to the literature.

The training phase is explicitly not included in the analysis, as its duration is mostly irrelevant for real-world applications. Here, the task is merely to provide recognition or detection systems for end users. These systems are implemented on embedded devices for consumer electronics, such as automotive electronic control units or smart phones, which are produced in large quantities using industrial manufacturing processes. For such systems, a minimum performance in terms of classification accuracy is defined and the goal is to find a classifier that meets this requirement while minimizing the cost per produced unit. This cost mainly depends on the performance of the employed processor. Nowadays, the price-performance ratio is approximately linear because modern processors feature superscalar arithmetical units whose number is proportional to the chip surface. Note that this is contrary to Grosch's law which postulated a squared relationship, that is doubled costs imply a four times faster system [124]. This law has however been formulated long before modern microprocessors emerged and is not applicable today due to improved manufacturing processes.

Therefore, only the computational complexity of the recall phase is relevant and not that of the training phase. The only requirement for learning algorithms is that they are asymptotically efficient, that is at most linear in the number of training samples such that no limitation on the size of the learning set is necessary. In the case where both classification accuracy and complexity are about equal, as encountered in the comparison of sparsely connected MLPs with the Optimal Brain Damage technique in Chapter 6, a smaller learning time is however decisive.

A few simplifications are made for the following comparison. The overhead of data pre-processing operations, which include performing a mean-variance normalization, padding the input samples with zero values to obtain images of a certain dimensionality and computation of Integral Images, are neglected. The comparison is still equitable, as all classification systems need some kind of pre-processing to produce robust classification results, and these routines are usually trivial such that their impact on overall computational complexity is negligible. Another simplification is ignoring the cost of transfer function evaluations, for example hyperbolic tangents. Depending on the concrete hardware the classification algorithms are run on, the evaluation of nonlinear functions can be achieved by a multitude of different algorithms such as polynomial approximations, lookup tables or shift-and-add algorithms that do not need a hardware multiplier, to name just a few [117]. Another possibility would be to use Elliot's sigmoid transfer function, see Section C.5, which possesses a sigmoid shape and can be computed exactly with a small number of simple algebraic operations, or the rectifier transfer function which was discussed in Section 5.2.4 and which can be computed simply by setting negative entries to zero. The exception to this rule is the sparseness-enforcing projection operator, which is used as transfer function in the Supervised Online Auto-Encoder. Because its computation involves solving an optimization problem with the problem dimensionality equal to the number of hidden units, its run-time overhead cannot be ignored. Instead, the upper complexity bound derived in Section 4.6.2 is incorporated into SOAE's complexity.

The comparison between competing classification algorithms is here based on two criteria. The first one is the achieved classification error on the MNIST evaluation set. As explained in Chapter 3, the error denotes the number of wrongly classified samples relative to all 10 000 samples, such that one misclassified sample corresponds to 0.01%. The second criterion is the computational cost of classification, as quantified by the model of computation introduced in Section 8.2. This model can be used to reduce entire algorithms to two scalar values, namely the number of multiply-accumulate operations and the number of memory accesses that are carried out during run-time. These two numbers should be reduced further to a single number on a ratio scale to allow the quantitative comparison of two approaches. An additive model is clearly appropriate for the considered model of computation.

Hence a value $Z \in [0, \infty] \subseteq \mathbb{R}$ is introduced which denotes the latency of memory accesses relative to the cost of a MAC operation, so that the overall cost becomes the number of MACs plus Z times the number of memory accesses. Large values of Z penalize a large number of memory accesses. Since Z is machine-specific, it has to be estimated using experiments once a concrete target hardware is known. Section 8.3.1 analyzes the impact of Z in greater detail and determines a value $Z_{\text{mid}} = 0.34$ that approximates the average effect of the two extreme choices $Z \in \{0, \infty\}$ best.

To ultimately yield values that are easy to interpret, the computational cost of individual approaches is given relative to that of a densely connected two-layer MLP with one thousand hidden units, called the *reference MLP*. This facilitates rendering comprehensive two-dimensional diagrams for comparison, see for example Figure 8.9. Here, the abscissa gives the generalization performance on a linear scale and the ordinate indicates the relative computational complexity on a logarithmic scale. Regions shaded in green, yellow and red simplify visual identification of the quality of different classification algorithms. The boundary on the ordinate was chosen to exactly match the complexity of the reference MLP, and an error of 1.0%

was chosen as the boundary on the abscissa because this is a classification performance easily satisfied by the reference MLP. Approaches that exceed both boundaries in the positive sense are located in a green region. When either the computational complexity or the number of misclassifications of an approach falls below the boundary values, it is located in the yellow regions. The red region encloses approaches where both the classification performance and the computational cost are suboptimal. This scheme is used to graphically illustrate all comparisons being made in the remainder of this chapter.

8.3.1 Sparsely Connected Multi-Layer Perceptrons

In each layer of an MLP, the input vector is multiplied with a weight matrix, a vector of thresholds is added to the result and a nonlinear transfer function is evaluated. Because adding the threshold vector is equivalent to proper initialization of the accumulation register in the matrix-vector product and transfer function evaluations are neglected, the computational cost of a layer equals that of an ordinary matrix-vector product.

For reference, consider a two-layer MLP with d inputs, n hidden units and c outputs as described in Chapter 6. Using the results from Section 8.2.1, inference in the hidden layer constitutes a cost of $dn + 2n(d+1)Z$, and computation of the output costs $nc + 2c(n+1)Z$, totaling $n(d+c) + 2Z(n(d+c+1)+c)$. Next, consider a sparsely connected two-layer MLP with a connectivity rate of $\rho \in (0,\ 1]$ in the hidden layer. When the training algorithm proposed in Section 6.3 has been used, it can be assumed that the connectivity matrix of the hidden layer is stored in the modified ELLPACK format, and ρd equals the number of nonzero entries of each column in the weight matrix. Then inference of the internal representation can be achieved at a cost of $\rho dn + 3n(\rho d + 2/3)Z$, and computation of the classification decision involves a cost equal to the case of a conventional two-layer MLP. Hence classification with an SMLP costs $n(\rho d + c) + 2Z(n(3/2\rho d + c + 1) + c)$.

By taking the ratio of the cost associated with a conventional MLP to the cost of a sparsely connected MLP, the speed-up as a function of the connectivity rate can be defined as

$$S_Z\colon\ (0,\ 1] \to \mathbb{R}, \quad \rho \mapsto \frac{n(d+c) + 2Z(n(d+c+1)+c)}{n(\rho d + c) + 2Z(n(3/2\rho d + c + 1) + c)}.$$

Figure 8.7 shows plots of S_Z where the memory latency Z was chosen from $\{0, 1, \infty\}$. Clearly, S_Z is decreasing when Z increases such that the limit $S_\infty := \lim_{Z\to\infty} S_Z$ is a lower bound to S_Z for any value of Z. Moreover, speed-ups are higher for a sparser connectivity in the hidden layer. The break-even point, that is the connectivity rate where both the conventional and the sparse MLP incur the same cost, can be determined by solving $S_Z(\rho^*) = 1$ for ρ^*, which yields $\rho^* = \frac{2Z+1}{3Z+1}$. Because the sparse storage format involves additional memory accesses to the matrix of positions, this expression depends on the memory latency. If $Z = 0$, that is when memory accesses are neglected and only the MAC operations are counted, then clearly

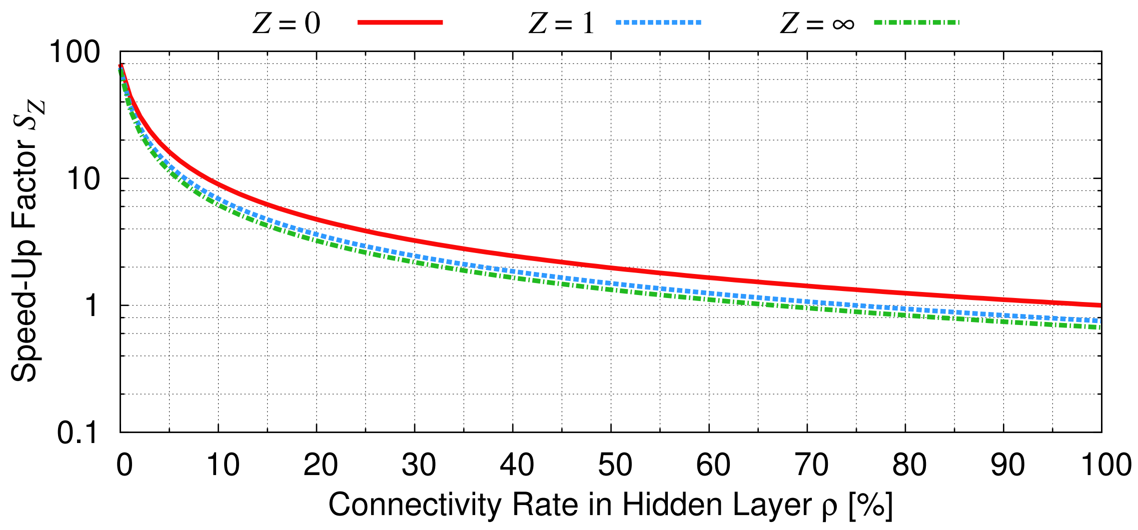


Figure 8.7: Speed-up S_Z of a sparsely connected two-layer MLP over a conventional MLP dependent on the connectivity rate in the hidden layer ρ and the cost Z of a memory access relative to the cost of a MAC operation. As the distance of the curves for the extreme values $Z = 0$ and $Z = \infty$ is small, the concrete choice of Z does not involve significant differences.

$\rho^* = 1$ and the sparse MLP is faster than the normal MLP for every connectivity rate smaller than a hundred percent. The other extreme case of $Z = \infty$ implies $\rho^* = 2/3$, which is also a lower bound for all feasible Z. This means that in this situation every sparsely connected MLP with a connectivity rate higher than $2/3$ is slower than its dense counterpart. Conversely, if the connectivity rate in the hidden layer drops below $2/3$, then the sparse MLP is faster than the dense MLP.

The memory latency Z is a parameter that penalizes frequent memory accesses. Its concrete choice depends in practice on the machine being used and its value has to be estimated experimentally. To allow for as general an analysis as possible, a value Z_{mid} for Z is chosen such that $S_{Z_{\text{mid}}}$ averages the extreme functions S_0 and S_∞. Solving $S_{Z^*} = 1/2 \cdot (S_0 + S_\infty)$ yields $Z^* = \frac{n(\rho d + c)}{3n\rho d + 2(nc + n + c)}$, which depends on the connectivity rate and the dimensionalities of the MLPs. Taking the average over all feasible connectivity rates yields

$$Z_{\text{mid}} := \int_0^1 Z^* \, \mathrm{d}\rho = \frac{1}{3} + \frac{nc - 2n - 2c}{9nd} \cdot \log\left(1 + \frac{3nd}{2(nc + n + c)}\right),$$

where the second term is small for reasonable values for d, n and c. By choosing these dimensionalities to match those of an MLP with 1000 hidden units that should process the MNIST database, it follows that $Z_{\text{mid}} = 0.34$. This can be interpreted as three memory accesses being approximately as expensive as one multiply-accumulate operation.

Table 8.3: Possible speed-ups of sparsely connected MLPs over conventional, densely connected MLPs. Here, the results from Table 6.3 of Chapter 6 for the trainings on the jittered and elastically distorted learning sets were used.

jitter	Sparseness degree σ_W	0.65	0.70	0.75	0.80	0.85	0.90
	Evaluation error [%]	0.89	0.85	0.81	0.84	0.91	0.93
	Minimum speed-up S_∞	3.6	4.3	5.2	6.5	8.4	14.3
	Average speed-up $S_{Z_{\text{mid}}}$	4.4	5.3	6.4	7.9	10.3	17.1
	Maximum speed-up S_0	5.2	6.3	7.6	9.4	12.1	20.0
elastic	Sparseness degree σ_W	0.65	0.70	0.75	0.80	0.85	0.90
	Evaluation error [%]	0.63	0.59	0.58	0.62	0.66	0.70
	Minimum speed-up S_∞	3.4	3.9	4.4	6.1	7.2	12.2
	Average speed-up $S_{Z_{\text{mid}}}$	4.2	4.8	5.5	7.5	8.9	14.7
	Maximum speed-up S_0	4.9	5.7	6.5	8.9	10.5	17.2

Quantitative estimates for the sparsely connected MLPs discussed in Chapter 6 are given in Table 8.3, for the minimum speed-up S_∞, the average speed-up $S_{Z_{\text{mid}}}$ and the maximum possible speed-up S_0. These numbers show that significant speed-ups are possible for very sparse connectivity, and that the concrete choice of Z does not involve very large differences. Hence for the remainder of this chapter, $Z_{\text{mid}} = 0.34$ is used to compare different approaches, as this is the best compromise between the two extrema where either only MACs or only memory accesses are counted to determine computational complexity.

It was moreover evaluated whether these considerations also hold in practice when a real computing machine is employed. For this, an Intel Core i7-980X processor was used in conjunction with three different implementations of the matrix-vector product from the Basic Linear Algebra Subprograms (BLAS). The "Canonical" variant is a simple implementation of Algorithm 8.1 which was neither optimized for the concrete processor nor does it benefit from sparsely populated matrices. A highly optimized alternative is the "GotoBLAS2" library [75, 76], which was tuned for the employed machine but does not support sparse matrices either. The simple Algorithm 8.2 was implemented for the "Sparse BLAS" variant, which can use structured sparseness but which lacks the machine-specific optimizations of GotoBLAS2. The classification decision of all MNIST evaluation samples was computed for each variant and each feasible connectivity rate, which was simulated by pruning a fully populated weight matrix until the desired sparseness degree was attained, and the elapsed run-time was measured. Then, by taking the ratio of the run-time of the two dense variants to the run-time of Sparse BLAS, the effective speed-ups could be obtained. They are depicted in Figure 8.8, together with the mean theoretical speed-up as determined above. The "Canonical" implementation closely follows the projected speed-ups, a small discrepancy occurs since transfer function evaluations are neglected in the model of computation but were carried out

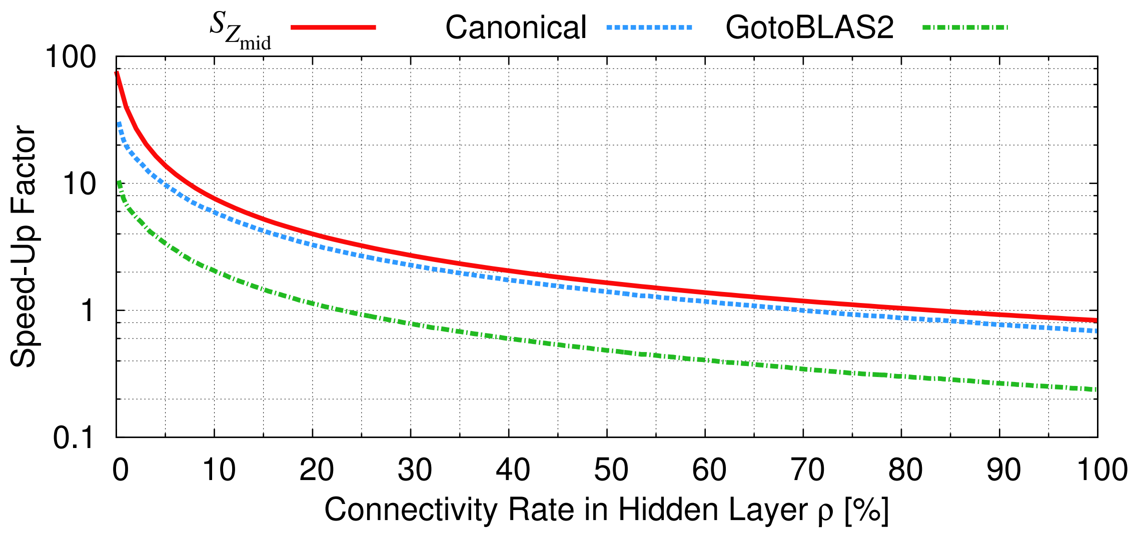


Figure 8.8: Measured speed-up when the class memberships of the samples of the MNIST evaluation set are predicted on a real computing machine. The $S_{Z_{\text{mid}}}$ curve denotes the theoretical speed-up based on the proposed model of computation. Further, "Canonical" indicates the speed-up when a sparse implementation of the matrix-vector product as described in Algorithm 8.2 is used relative to a canonical implementation of Algorithm 8.1, and for the "GotoBLAS2" curve an optimized and tuned library for the dense matrix-vector product was used. The "Canonical" curve closely follows the projected speed-up, whereas speed-ups are smaller when the GotoBLAS2 library is used since the sparse matrix-vector product was neither tuned nor optimized for the concrete processor.

for the measurements. The speed-up relative to the GotoBLAS2 library is not equally distinctive since that library was excessively tuned for the concrete hardware. If a comparable effort was spent in reducing the run-time for the sparse matrix-vector product, or if another processor was used, the performance boost due to sparseness would be even more significant.

8.3.2 Multi-Layer Perceptrons

Since the Sparse Coding for Fast Classification algorithm from Chapter 5 only achieved an error of 1.4% and could not be applied to the augmented MNIST learning sets due to memory constraints, it is not included here. All relevant results from Chapter 6 and Chapter 7 are shown in Figure 8.9, dependent on the achieved classification error on the evaluation set and the computational complexity relative to a conventional two-layer MLP with 1000 hidden units. The entry "MLP Jitter" denotes standard MLPs trained on the jittered MNIST learning set. Each point corresponds to a different number of hidden units, namely 1000, 900, 600,

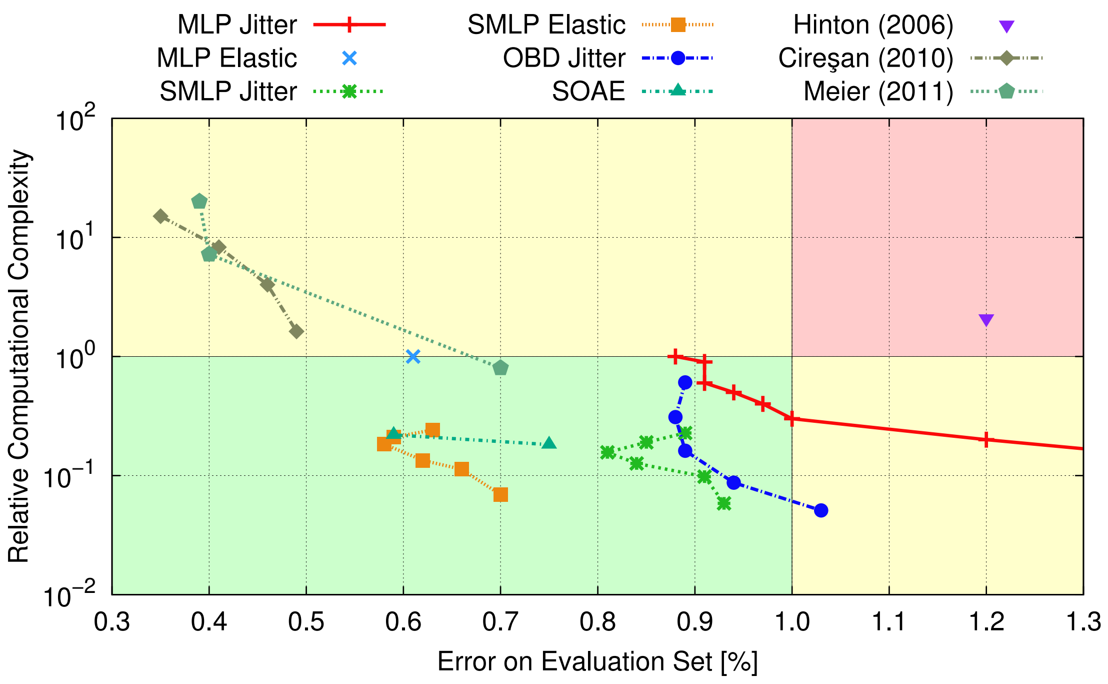


Figure 8.9: Results from Section 8.3.2, "Multi-Layer Perceptrons".

500, 400, 300, 200 and 100 from the left to the right where the last point is not entirely visible. The points for 800 and 700 hidden units were omitted to avoid clutter in the graphics. The raw data for this curve was given in Table 6.2. The single point of "MLP Elastic" corresponds to densely connected MLPs trained on the elastically distorted MNIST learning set. The relative computational complexities of "MLP Elastic" and the leftmost point of "MLP Jitter" equal unity, since they have their architecture with the reference MLP in common.

The entries "SMLP Jitter" and "SMLP Elastic" denote sparsely connected MLPs, trained on both augmented variants of the learning set. The points in the individual curves correspond to different connectivity rates in the hidden layer and thus different speed-ups, see Table 8.3 for the raw data. The densely connected MLPs from the "MLP Jitter" trajectory are clearly outperformed by the SMLPs, which shows that simply reducing the number of hidden units is not very effective. While the classification accuracy is about equal to the densely connected counterparts, and in some cases better, computational complexity is reduced by up to one order of magnitude.

The trajectory for "OBD Jitter" belongs to the Optimal Brain Damage experiments from Chapter 6, each point belonging to a different connectivity rate. While OBD does help to improve computational complexity, the classification performance does not improve with sparser connectivity as opposed to sparsely connected MLPs. Hence, considering OBD's overhead during training due to alternating optimization and pruning, sparse MLPs are favorable.

The results of the Supervised Online Auto-Encoder from Chapter 7 are shown in the "SOAE" curve, the point on the left for training on the elastically distorted samples and the point on the right for training on the jittered samples. Both points correspond to the pooled results of the SOAE-σ-SC variant. If the individual results of the best-performing sparseness degrees from Table 7.1 and Table 7.2 were used, the evaluation errors would be slightly lower. The computational complexity was determined based on the sparse matrix-vector product given the concrete connectivity rates in the hidden layer and the overhead of the sparseness-enforcing projection operator for inferring the internal representations. SOAE is slightly slower than the SMLPs due to the sparseness projection. The generalization performance is however better for the jittered training samples, and statistically equivalent for the elastically distorted samples.

The other results from Figure 8.9 have been taken over from the literature. A deep restricted Boltzmann machine trained greedily in a layer-by-layer fashion was proposed by [125], see also Section C.4 for a discussion. Their result is denoted by "Hinton (2006)" in Figure 8.9. As this is essentially a Multi-Layer Perceptron with more than two layers, computational complexity can be determined easily. This approach used the original MNIST learning set without any artificially generated samples, and achieved an error of 1.2% using three hidden layers with 500, 500 and 2000 hidden units, respectively. This error is surprisingly low for a permutation-invariant approach; for example, the two-layered SCFC architecture from Chapter 5 achieved an error of 1.4%, which already significantly outperformed an ordinary MLP that achieved an error of 1.8% using only the original learning set, see Chapter 6.

Another surprising result was reported by [59]. They trained very deep conventional MLPs with up to six layers and up to twelve million synaptic connections. To reliably train such large neural networks, virtual samples were generated continuously during training using elastic distortions. The displacement fields were permanently re-sampled so that every artificial sample was unique. At the time of its publication, [59] took the lead on MNIST with a classification error of 0.35% using an MLP with five hidden layers with 2500, 2000, 1500, 1000 and 500 hidden units, respectively. This result is the leftmost point on the "Cireşan (2010)" curve in Figure 8.9. The other points correspond to the smaller neural networks reported in Table 1 in [59]. Although the classification error is very low, in particular on an absolute scale, this is paid for with an increased demand for computational resources: The largest network that achieved the 0.35% error rate is about 15.1 times slower than the reference MLP.

While this is a success for deep architectures applied to handwritten digit recognition, a similar performance can also be achieved with a very broad architecture consisting of ordinary two-layer MLPs. The same technique was also applied to Convolutional Neural Networks to yield the current record, see Section 8.3.3. For the experimental results reported in [126], a number of different variants of the MNIST learning set were generated using combinations of affine and elastic distortions, and width normalization by compressing pixels in the horizontal direction. For each experiment, nine two-layer MLPs with 800 hidden units were then trained on nine distinct learning sets. These nine classifiers were used to form a committee with different strategies for fusing the classification decision. The strategy that averaged the outputs of the classifiers and subsequently took the maximum entry for the classification decision worked best. The best committee of nine MLPs achieved an error of 0.40% on the MNIST evaluation set; this corresponds to the middle point of the "Meier (2011)" trajectory in Figure 8.9. By pooling the results from three different experiments, which resulted in twenty-five distinct classifiers without redundancy, the error could be improved to 0.39%, which is the leftmost point of the respective curve in Figure 8.9. The rightmost point of that curve belongs to the median error of the individual neural networks, whose error rates are given in Table 2 in [126]. While the committee strategy does not improve the results from the same group using deep MLPs described above, it provides a simple and effective scheme to improve single classifiers. The striking disadvantage, though, is that computational complexity increases linearly with the number of classifiers used to form the committee. While the small committee with nine networks is about 7.2 times slower than the reference MLP with 1000 hidden units, the large committee with 25 classifiers is already twenty times slower. Compared to the small committee with nine classifiers, only one sample less is misclassified. It is therefore questionable whether this approach is able to produce well-performing classification systems with both a small number of classification errors and low running times during the recall phase.

Few of the approaches discussed in Section 2.4 and incorporating sparse information processing are suited for classification tasks. The Optimal Brain Damage method and the three models proposed in this work were already discussed here. Supervised NMFSC with Second Order Cone Programs [29] only features a mild form of teacher signal incorporation. No results of this approach on MNIST were reported, instead only a very small subset of another handwritten digit recognition data set was used. The Supervised Dictionary Learning approach of [47] achieved an evaluation error of 1.05% on the MNIST database using a permutation-invariant classifier trained on the original learning set with 60 000 samples. This is a substantial improvement to the "Hinton (2006)" result [125], which is also permutation-invariant but yielded an error of 1.2%, discussed earlier. The final approach that was discussed in Section 2.4 is the permutation-invariant Differentiable Sparse Coding by [48]. Teacher signal incorporation is here achieved through a separate backpropagation step carried out after unsupervised pre-training. This resulted in an evaluation error of 1.30%, worse than for Supervised Dictionary Learning. The latter two approaches were not included in Figure 8.9 since no details on the number of used bases have been published by [47, 48], which renders an estimation of their computational complexity impossible.

8.3.3 Convolutional Neural Networks

Typical Convolutional Neural Networks consist of several alternating convolutional and pooling layers, followed by fully connected layers that compute the classification decision from the features extracted by the earlier layers [54]. The cost for carrying out the convolutional and pooling layers can be inferred using the resulting numbers from Section 8.2.2 and multiplying them with the appropriate number of hidden units. It is merely important to accurately

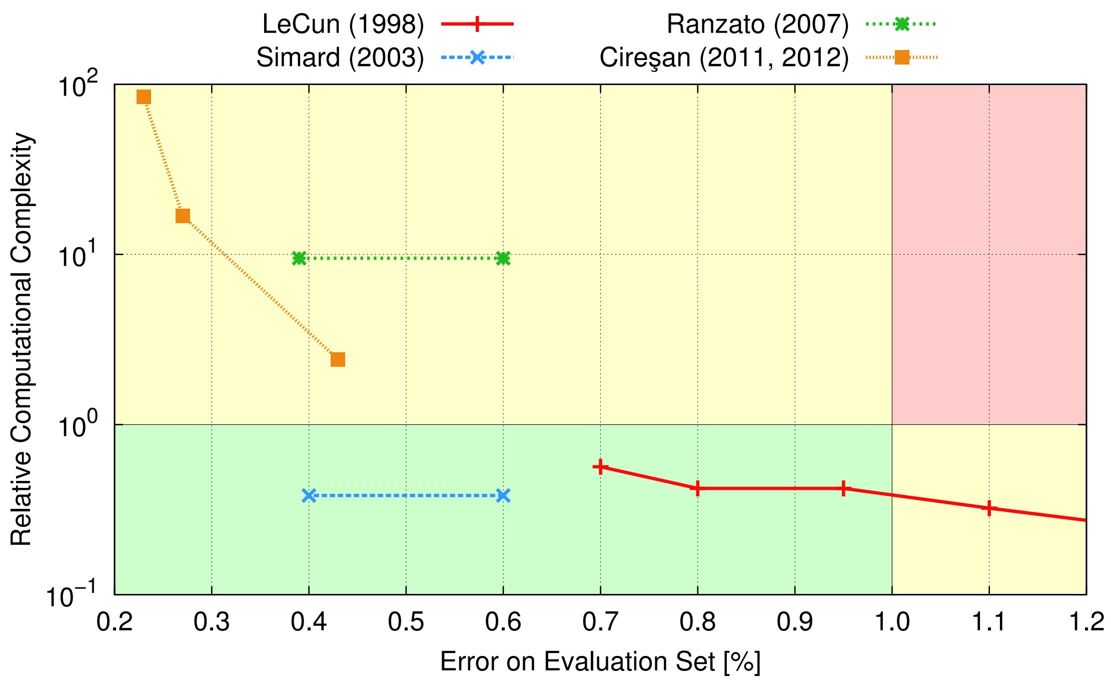


Figure 8.10: Results from Section 8.3.3, "Convolutional Neural Networks".

track the output dimensionality of each layer for a precise overall cost computation. The fully connected layers at the end of the processing chain can be modeled as done in the previous section on Multi-Layer Perceptrons. Because computation of Euclidean distances is as efficient as performing an ordinary matrix-vector product, see Section 8.2.1, RBF network layers require no special treatment. Evaluation of Gaussian functions is neglected since they can be cast as ordinary nonlinear transfer functions which are neglected in the analysis.

The LeNet family of Convolutional Neural Networks was proposed by the same group that released the MNIST database. The "LeCun (1998)" curve in Figure 8.10 shows results from different variations of these CNNs [127, 54]. The first incarnation LeNet-1 works on input images with only 16×16 pixels. The network consists of two pairs of convolutional and pooling layers followed by one fully connected layer, thus totaling in five layers. The exact architecture parameters are given in Section 4 of [127]. This very simple and very small network yields an evaluation error of 1.7% on MNIST [54]; the corresponding point is not visible in Figure 8.10 due to the small range depicted there.

The successor of this architecture, LeNet-4, already works on fully padded MNIST samples and has more hidden units than LeNet-1. LeNet-4 achieves an evaluation error of 1.1% [54], resulting in the rightmost still visible point in Figure 8.10. Its computational complexity was derived based on the architecture description given in Section III.C.8 in [54]. The next point belongs to LeNet-5, which yielded an error rate of 0.95%. This network processes input images with 32×32 pixels, thus padding the original samples that had 28×28 pixels, first feeding them through two pairs of convolutional and pooling layers followed by three fully-connected layers with the final one implementing an RBF network. The precise architectural details needed to infer computational cost are given in Figure 2 and Table 1 of [54]. It is noteworthy that LeNet-5 is 2.4 times faster than the reference MLP, mainly due to all network parameters, such as the number of layers and hidden units and the size of filter kernels, having been tuned exclusively for MNIST. When the learning set is augmented with artificial samples created using affine distortions, the evaluation error drops to 0.8% using LeNet-5; this corresponds to the second point from the left on the "LeCun (1998)" curve in Figure 8.10.

The leftmost point of that curve belongs to a boosted LeNet-4, where three classifiers were combined to form a rejection cascade such that in the mean the computational cost is increased by a factor of 1.75 compared to a single instance of LeNet-4. This yielded the lowest error of 0.7% within the LeNet family, and is still 1.8 times faster than the reference MLP. These results show that an architecture tuned exclusively for a given classification task is quite effective, the main drawback being that a large number of experiments is required for optimal tuning of the numerous architecture parameters.

By introducing elastic distortions to handwritten digit recognition, [58] set a benchmark that took several years to be improved. While previous approaches only used affine distortions, elastic distortions can be used to create huge numbers of virtual samples with an unparalleled variety, see also Chapter 3. Further, pooling layers were omitted in [58] and instead convolutions with a stride of two were used to achieve downsampling. Note that without a low-pass filter such as averaging over small neighborhoods, this approach introduces aliasing artifacts that may degrade performance significantly on other data sets. The CNN of [58] thus only has two convolutional layers followed by two fully connected layers, see Figure 3 in [58] for details. When trained using elastically distorted samples, the network achieves an error of 0.4%, and using only affine distortions it yields a classification error of 0.6%. This corresponds to the "Simard (2003)" curve in Figure 8.10. Regarding computational complexity, this optimized CNN is 2.6 times faster than the reference MLP, and still 1.5 times faster than the boosted LeNet-4 while almost halving the number of misclassifications. Because of this good performance in both evaluation criteria, the approach will later be compared with other approaches in more detail.

As shown by [125] and discussed in Section C.4, unsupervised pre-training can help to improve generalization capabilities compared to when random numbers are used to initialize the degrees of freedom. The trajectory titled "Ranzato (2007)" represents results from [128], where a modified LeNet-5 architecture was used in conjunction with a sparse auto-encoder. To account for sparseness of activity during unsupervised pre-training, the original LeNet-5 network was enlarged by adding hidden units, so as the enlarged network involves a cost increased by factor 22.5 compared to its original variant. When trained on the original learning set without any artificial samples, this yields an error of 0.60%, which belongs to the point on the right of the corresponding curve in Figure 8.10. When elastic distortions are used, the error decreases to 0.39%, that is one digit more out of ten thousand has been correctly classified compared to the result of [58]. Further, classification is 9.5 times slower than the reference MLP. Since such a classification performance can also be achieved with a classifier that needs only a twentieth of the computational resources, the results of [128] are merely of academic interest.

The final results using CNNs discussed here are associated with the curve named "Cireşan (2011, 2012)" from Figure 8.10. Here, a network with about a quarter the size of the enlarged CNN from [128] was used and large committees of several instances of classifiers trained on different variants of the learning set were formed. The detailed network characteristics are given in Section 2 of [129]. Analogously to the approach of [126] discussed in Section 8.3.2, different distortion parameters for augmenting the learning set were used, and the final classification decision was inferred by averaging over the output of individual networks. The rightmost point of the relevant curve in Figure 8.10 with a classification error of 0.43% represents the median performance of 35 single CNNs, see Table 8 in [129]. The middle point of that curve corresponds to a committee of seven such classifiers, which achieved an error of only 0.27% on the MNIST evaluation set.

In the subsequent publication [60] of the same authors, all 35 individual CNNs were fused into one large committee, yielding the record error of 0.23%, given by the leftmost point in Figure 8.10. Although this is the best result ever obtained on the MNIST database, combination of that many classifiers results in proliferation of computational complexity. Computation of the classification decision of these 35 CNNs is about 84.3 times slower than using the reference MLP that achieved an error of 0.61%. That relates to almost two orders of magnitude, and compared to sparse MLPs that achieved an error of 0.58%, complexity is raised by a factor of 460. Porting such a large classifier onto consumer hardware is not realistic due a disproportionate price-performance ratio in terms of hardware cost, power consumption and heat dissipation.

8.3.4 Support Vector Machines

Support Vector Machines (SVMs) are mathematical models that infer classification decisions based on the location of samples in learned half-spaces of the feature space [56]. These half-spaces are divided by an optimal margin hyperplane, that is a hyperplane found by maximizing its distance to all the training points in dependence of their class label, trading off classification accuracy on the learning set for margin size in the usual case where training samples are not linearly separable. The normal of the separating hyperplane is given as linear combination of a subset of the training samples, the so-called *support vectors*. Nonlinear decision surfaces are possible by mapping the input samples implicitly into a higher-dimensional kernel space $\mathcal{H}$ and using a kernel function $\kappa\colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ that maps from the feature space to $\mathcal{H}$ and then computes the dot product in $\mathcal{H}$ [56]. The classification decision for an input sample $x \in \mathbb{R}^d$ is the sign of $\sum_{i=1}^{n} \alpha_i \kappa\big(x, s^{(i)}\big) + \theta$, where $s^{(i)} \in \mathbb{R}^d$ are the support vectors with associated weight $\alpha_i \in \mathbb{R} \setminus \{0\}$ for $i \in \{1, \ldots, n\}$, and $\theta \in \mathbb{R}$ is a threshold [56]. Typical examples for κ are a Gaussian kernel $\kappa(x,s) := \exp\big(-\|x-s\|_2^2/\sigma\big)$ where $\sigma > 0$ controls its shape, or a polynomial kernel $\kappa(x,s) := (x^T s)^g$ where $g \in \mathbb{N}$ is the degree of the polynomial.

The computational complexity of classifying an individual sample equals the cost of evaluating the kernel function between the input sample and all support vectors, and subsequently computing the weighted sum of the result. Moreover, kernel function evaluation reduces to computation of a dot product or a Euclidean distance in $\mathbb{R}^d$; possible nonlinearities are neglected here as they are essentially transfer functions. Therefore, as dot products and Euclidean distance are treated equally, the overall complexity is assembled by the kernel function evaluations which correspond to multiplying the transpose of a matrix from $\mathbb{R}^{d \times n}$ with a vector from $\mathbb{R}^d$, plus weighting the kernel products with the entries of a vector $\alpha \in \mathbb{R}^n$, which corresponds to the dot product of two vectors from $\mathbb{R}^n$. Therefore the total cost is $n(d+1) + 2Z\big(n(d+2)+1\big)$. Classifiers are trained in a one-vs.-all fashion when there are more than two classes. This produces c classifiers when c denotes the number of classes, where each classifier may have its own disjoint set of support vectors. In this case the complexity of each of the c classifiers has to be combined additively to yield the overall complexity of the classification system.

The tendency of SVMs to incur high computational cost due to large sets of support vectors on real-world data sets was recognized early. Proposals were made to reduce the complexity by shrinking the set of vectors that determine the classification decision. One such approach was proposed by [130], where the normal vector of the SVM decision surface was approximated with a linear combination of a smaller number of vectors. In their MNIST experiments, they used an SVM with a polynomial kernel of degree five, trained using the virtual support vector method by jittering the support vectors in each of four principal directions and starting training anew. This yielded an error of 1.0%, depicted in Figure 8.11 in the "Burges (1997)" curve as top point. Computational complexity was determined using the number of support vectors per class as given in Table 1 of [130] and found to be 43.9 times higher than that of the reference MLP. Subsequently, the number of support vectors was reduced by factor 50 using the aforementioned approximation to the decision surface, resulting in a classification error of 1.1% on the MNIST evaluation set [130]. This result belongs to the bottom point of the "Burges (1997)" curve in Figure 8.11, whose computational complexity was computed using the number of reduced support vectors given in Table 3 of [130].

These experiments show that a method similar to virtual sample generation can lead to competitive results in terms of generalization performance when SVMs are used, and that support vector reduction can be used to improve classification speed without significant loss in generalization capabilities. The overall performance, though, is worse than what can be achieved with a conventional two-layer MLP using jittered samples, which resulted in an error of 0.88% as demonstrated in Chapter 6.

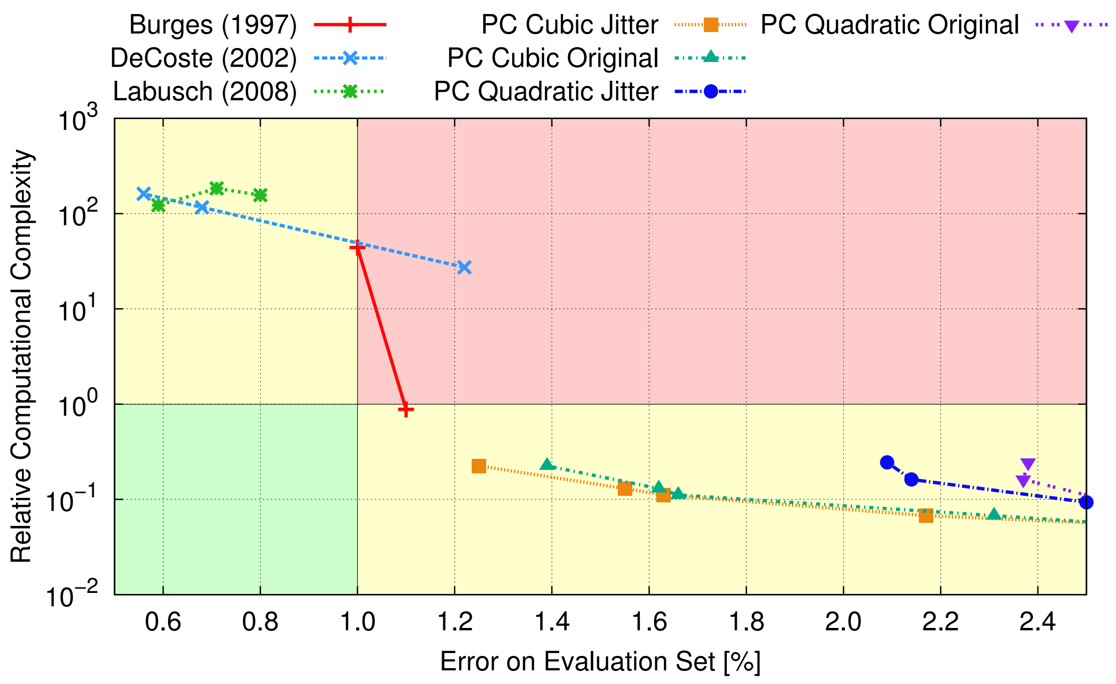


Figure 8.11: Results from Section 8.3.4 and Section 8.3.5, "Support Vector Machines" and "Polynomial Classifiers", respectively.

An incremental result was reported by [57], where only the virtual support vector method was evaluated but no support vector reduction was carried out. They used a polynomial kernel of degree nine and de-slanted all samples of the learning set by rotating the original samples into the upright position. Using only 60 000 de-slanted samples, an error of 1.22% was obtained [57], represented by the rightmost point of the "DeCoste (2002)" curve in Figure 8.11. The middle and leftmost point are associated with virtual support vectors jittered in all possible eight directions by one or two pixels, respectively. The largest SVM is more than 160 times slower than the reference MLP as determined from its number of support vectors per class given in Table 9 in [57]. This is the second-slowest of all approaches analyzed in this chapter. Unfortunately, no attempt to lower the computational complexity was performed in [57].

A hybrid of unsupervised feature extraction, shallow Convolutional Neural Networks and Support Vector Machines with Gaussian kernel was proposed by [131], which although called a simple method by its authors, involves the largest computational complexity of all methods investigated here, being two times slower than the committee of 35 CNNs proposed by [60]. The model proposed by [131] is first trained in an unsupervised manner on 13×13 pixels image patches from the MNIST learning set using either Principal Component Analysis or the sparse coding model of [10]. Additionally, a bank of Gabor filters [15] was used as a third feature set. Feature values are generated for each feature set by unit-stride convolution of the input samples and subsequent pooling over 9×9 pixels neighborhoods. Here, the pooling is determined to find both the maximum and minimum entry of each neighborhood, effectively doubling the effort involved in an ordinary pooling operation. Because either 160 or 169 distinct elements from the feature set are used, feature dimensionality equals 2880 or 3042, respectively. These large feature vectors are eventually classified by an SVM with several thousand support vectors per class, the concrete numbers are given in Table 1 in [131]. It should be noted that when using the algorithm of [10] for sparse coding, an optimization problem would have to be solved for every position in which the feature set is applied to the input samples, compare also with Section 5.1.2. This is neglected here for simplicity, and hence the complexity for this variant is underestimated.

The results are shown as the "Labusch (2008)" curve in Figure 8.11, from right to left using PCA, Gabor and sparse coding features, respectively. The best feature set is that using sparse coding, which achieved an error of 0.59% in this architecture and is at least 120 times slower than the reference MLP, not accounting for the cost of sparse code word inference. The feature values extracted using Gabor filters lead to a classification error of 0.71%, with complexity of the recall phase being 184 times higher than the reference MLP due to the large number of support vectors. The only argument for the approach of [131] is that only the distortion-free original learning samples were used and that the feature extractor did not receive any information on class membership during training. However, this approach is not permutation-invariant due to the convolution and pooling operations, and it is hence not clear why this approach should be preferred over others. For example, the approach of [128] using an enlarged LeNet-5 as discussed in Section 8.3.3 achieved a statistically equivalent error of 0.60% also using the original learning set without any distorted samples, but their approach is more than one order of magnitude faster.

Summing up, approaches based on Support Vector Machines may result in good classification performance, but their complexity is very high unless reduced by special techniques such as that of [130]. There is however no result in the literature indicating whether this also works for classifiers already yielding fewer than 1.0% misclassifications on the evaluation set.

8.3.5 Polynomial Classifiers

As explained in Section 8.2.3, a Polynomial Classifier evaluates all monomials up to a specified degree $g \in \mathbb{N}$ and combines the result linearly to predict the class membership of individual samples. In order to reduce the dimensionality of the input samples and thus the number of monomials, the input samples are first projected onto the first n principal axes of the learning set [120]. This yields $p := \binom{n+g}{g}$ monomial terms, see Section 8.2.3. If this dimensionality reduction was not performed, memory requirements of the training algorithm would be intractable [121]: There is one moment matrix for each class used to determine the linear classifier, and each moment matrix is square with the edge length being equal to the number of monomials. Therefore, there are $p^2 \approx n^{2g}$ entries in each moment matrix. If for example $g = 2$ was chosen for quadratic monomials, then the memory requirements quadruple in the number of dimensionalities input to the Polynomial Classifier training algorithm. Hence for $g \geq 2$ only small values for n are feasible.

Input samples of dimensionality d are thus projected onto $n \ll d$ principal components, all monomials are evaluated as in Section 8.2.3 and the result is combined linearly for the class membership estimation. The projection is carried out by multiplying a vector from $\mathbb{R}^d$ with the transpose of a matrix from $\mathbb{R}^{d \times n}$, and linear classification consists of computing the dot product of two vectors from $\mathbb{R}^p$ for each of the c classes. The total cost hence amounts to $dn + pc + \sum_{q=2}^{g} (q-1) \cdot \binom{n+q-1}{q}$ MACs and $2n(d+1) + 2c(p+1) + 2 + \sum_{q=1}^{g} (q+1) \cdot \binom{n+q-1}{q}$ memory accesses.

So far, most of the results of Polynomial Classifiers have been reported using private, undisclosed data sets rather than the publicly available MNIST data sets. Results on MNIST only include that of [54] where 40 principal components and quadratic monomials were used to achieve an error of 3.3%, and the results of [132] who used 50 principal components and also quadratic monomials but achieved an error of 1.63% by handling the multi-class task by a one-vs.-one strategy instead of training one-vs.-all classifiers.

Using software kindly provided by Ulrich Kreßel, the first author of the review paper [121], meaningful results could be obtained using a machine that was equipped with sufficient memory to store the moment matrices required for training. In doing so, classifiers with quadratic and cubic monomial terms were trained using the original and the jittered MNIST learning set for a variety of different numbers of principal components. Since the optimization problem is convex and the training algorithm is deterministic, it was not necessary to carry out each variant more than once. The classification results on the MNIST evaluation set and the possible speed-ups over the reference MLP are given in Table 8.4. Additionally, the results up to an error of 2.5% are depicted in Figure 8.11 as the curves titled "PC Cubic Jitter", "PC Cubic Original", "PC Quadratic Jitter", and "PC Quadratic Original".

Clearly, the classifiers using monomials up to the quadratic degree do not attain errors below 2.0%. Augmenting the learning set with jittered samples does help reduce the amount of misclassifications, although the effect is not as strong as in the MLP and SVM experiments. This may be because only small numbers of principal components could be employed due to memory constraints. This has the effect of a low-pass filter and as a result small changes like jittering by one pixel do not invoke sufficient changes in the projection onto the principal axes. The best classification error of 1.25% was obtained using forty principal components, monomials up to the cubic order and the jittered learning set. Computational complexity is very low, as this classifier is about 4.4 times faster than the reference MLP.

Increasing the number of principal components n or the degree of the monomials g was not possible due to the enormous memory requirements. If double precision floating point numbers with 8 bytes per scalar are used, for example, the moment matrices for $n = 40$ and $g = 3$ require 11 gigabytes. In the case of $n = 48$ and $g = 3$, 32 gigabytes would be needed, and $n = 40$ and $g = 4$ would require 1370 gigabytes of main memory, which demonstrates the

Table 8.4: Results of Polynomial Classifiers obtained by application of the software kindly provided by the first author of [121] on the MNIST data set. The first and second columns denote parameters of the training algorithm. The third and fourth columns show the classification error on the MNIST evaluation set when training on the original and jittered learning set, respectively. The final column shows the speed-up factor of classification relative to a two-layer MLP with 1000 hidden units.

Maximum monomial degree	Number of principal components n	Evaluation error [%] when trained on original samples	Evaluation error [%] when trained on jittered samples	Speed-up factor to reference MLP
quadratic	8	16.78	19.13	117
quadratic	12	11.09	11.89	76.2
quadratic	16	7.86	8.62	55.7
quadratic	20	5.95	6.58	43.5
quadratic	24	5.11	5.24	35.4
quadratic	30	4.04	4.03	27.3
quadratic	32	3.89	3.75	25.3
quadratic	40	3.38	3.20	19.4
quadratic	48	2.93	2.86	15.5
quadratic	60	2.60	2.55	11.6
quadratic	64	2.56	2.50	10.7
quadratic	96	2.37	2.14	6.2
quadratic	128	2.38	2.09	4.1
cubic	8	12.78	14.06	96.7
cubic	12	6.80	7.11	53.7
cubic	16	4.09	4.58	33.0
cubic	20	2.79	2.92	21.6
cubic	24	2.31	2.17	14.8
cubic	30	1.66	1.63	9.0
cubic	32	1.62	1.55	7.7
cubic	40	1.39	1.25	4.4

substantial memory increase due to the large size of the moment matrices. It can hence be concluded that Polynomial Classifiers only have a limited field of application, which already seems exhausted with the MNIST database of handwritten digits.

8.3.6 Boosted Stumps

Boosting is a technique that greedily combines a number of simple classifiers, called *weak learners*, into a more powerful classifier, called a *strong learner* [122]. In each boosting iteration, the weak learner with the lowest error on the learning set is added to the strong learner and the strong learner's weights are adapted. Perhaps the most simple weak learner type is a decision stump, which only compares a feature value with a learned threshold and outputs a bipolar classification decision from $\{\pm 1\}$ based on the comparison. Since the set of all possible Haar-like filters is highly overcomplete, boosting can here be used to simultaneously select features and construct a classifier [122].

A generalization of this classical technique is boosting products of decision stumps for classification tasks [133]. When the number of output classes is c, then each weak learner computes a vector $h := \alpha \prod_{j=1}^{m} \varphi_j v^{(j)} \in \mathbb{R}^c$, where $\alpha \in \mathbb{R}$ is the weight of the weak learner, $\varphi_j \in \{\pm 1\}$ is the bipolar response of a single decision stump and $v^{(j)} \in \{\pm 1\}^c$ is a vote vector for multi-class classification. The number $m \in \mathbb{N}$ defines how many factors each weak learner should have and is a parameter of the learning algorithm. The overall classification decision is formed by evaluating all $n \in \mathbb{N}$ weak learners, adding their output vectors and finally finding the maximum entry in this vector.

First note that $\prod_{j=1}^{m} \varphi_j v^{(j)} \in \{\pm 1\}^c$ can be computed using simple logic operations and does not require floating point computations as soon as the decision stump's output φ_j has been determined based on a simple floating point comparison. Hence, when the number of classes c is smaller than the processor's word size this product takes m MAC operations for the comparisons. Additionally, there are $2m$ numbers that have to be read from memory, namely the feature values and the decision thresholds. Only one additional write access is required because when c is small the bipolar output vector fits into a single word.

To compute the strong learner's output, the bipolar vectors from the last step have to be linearly combined using floating point arithmetic. This is the same as multiplying the transpose of a matrix from $\{\pm 1\}^{c \times n}$ with a vector from $\mathbb{R}^n$, where n is the number of weak learners. Although a multiplication is not mandatory in this setup since the matrix has only bipolar entries, floating point additions which are not faster than multiply-accumulate operations nevertheless have to be carried out. Hence, cn MACs are associated with the strong learner evaluation. Further, even though only one bit of information has to be read to determine the sign of the argument, this is as efficient as reading an entire word on a modern processor. Therefore, the entire $2c(n+1)$ memory accesses associated with a matrix-vector product are necessary.

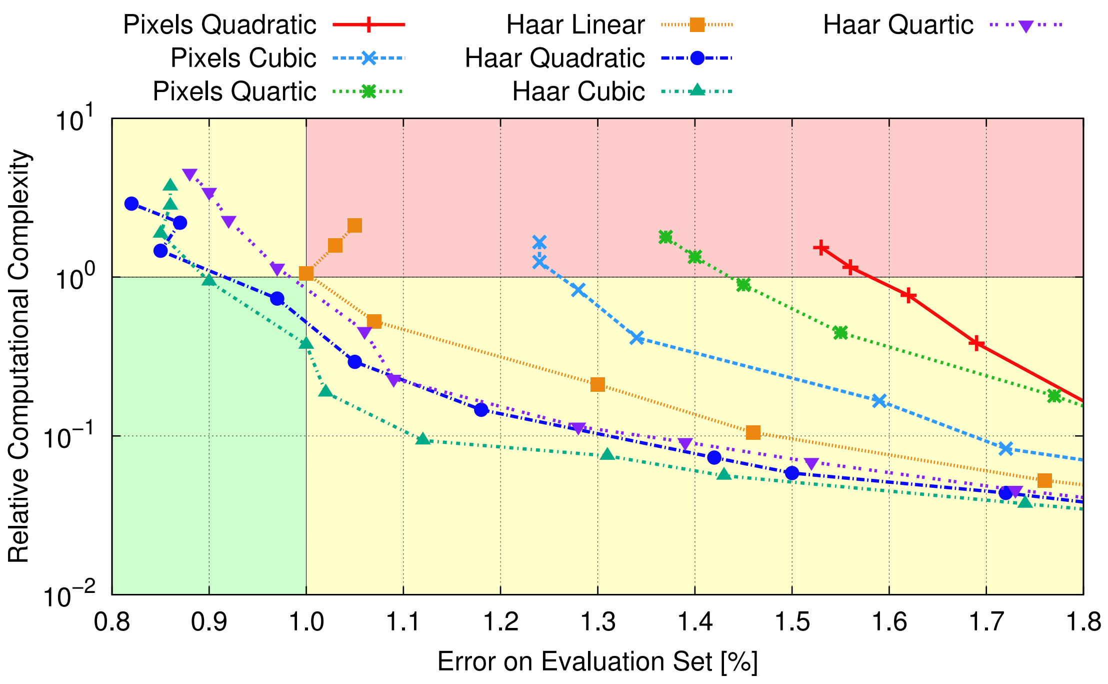


Figure 8.12: Results from Section 8.3.6, "Boosted Stumps".

The first author of [133], Balázs Kégl, kindly provided detailed results from their experiments of boosting products of decision stumps on the MNIST data set, where either the raw pixel values or Haar-like filters were used to generate feature values. The entire history of the training runs was made available, so it was possible to infer classification accuracy and computational complexity for every number of weak learners up to 100 000, at which point training was stopped. An elaborate list of results is reproduced in Table 8.5 since they have not been published in this detail.

The results are also depicted in Figure 8.12, where the number of weak learners was varied to generate the different trajectories. The first word of the curve titles indicates the feature type, and the second word denotes the number of factors used in the products of the stumps. The trivial case of pixel features and products with exactly one factor has been omitted in Figure 8.12 due to the bad classification performance. Further, it can be seen that for both feature types the classifiers with quartic terms produce worse results than the ones with cubic terms, which is due to overfitting [133]. Moreover, usage of Haar-like filters consequently outperforms raw pixel values, which is because Haar filters take the pixel topology into consideration and hence the information that images and not unordered vectors should be processed is available to the classifier [133]. Overall, the classifier using Haar-like filters and cubic terms performs best; for example, it achieves an error of 1.02% using only 5000 weak learners which corresponds to a speed-up factor of 5.3 to the reference MLP. The best single result of an error of 0.82% is obtained for quadratic product terms, which is however 2.9 times slower than the

Table 8.5: Selection of results of boosted product stumps as kindly provided by the first author of [133]. The first column indicates the used feature type and the second column gives the number of weak learners the strong learner consists of. The following columns show the achieved classification error on the MNIST evaluation set and the speed-up relative to the reference MLP, dependent on the maximum degree of the products of stumps.

		linear		quadratic		cubic		quartic	
Feature type	Number of weak learners	Evaluation error [%]	Speed-up factor	Evaluation error [%]	Speed-up factor	Evaluation error [%]	Speed-up factor	Evaluation error [%]	Speed-up factor
Pixels	500	10.38	142	4.34	130	4.13	120	4.20	112
Pixels	1000	9.08	70.9	3.04	65.1	2.96	60.2	3.07	55.9
Pixels	1500	8.27	47.3	2.58	43.4	2.57	40.1	2.56	37.3
Pixels	2000	8.12	35.5	2.35	32.5	2.24	30.1	2.42	28.0
Pixels	2500	7.89	28.4	2.26	26.0	2.06	24.1	2.18	22.4
Pixels	5000	7.61	14.2	1.92	13.0	1.72	12.0	1.91	11.2
Pixels	10 000	7.50	7.1	1.81	6.5	1.59	6.0	1.77	5.6
Pixels	25 000	7.62	2.8	1.69	2.6	1.34	2.4	1.55	2.2
Pixels	50 000	7.66	1.4	1.62	1.3	1.28	1.2	1.45	1.1
Pixels	75 000	7.66	0.95	1.56	0.87	1.24	0.80	1.40	0.75
Pixels	100 000	7.76	0.71	1.53	0.65	1.24	0.60	1.37	0.56
Haar	500	3.47	96.2	2.52	69.2	2.29	53.9	2.17	43.9
Haar	1000	2.53	47.9	1.97	34.4	1.74	26.8	1.73	21.9
Haar	1500	2.13	31.9	1.72	22.9	1.43	17.8	1.52	14.6
Haar	2000	1.90	23.9	1.50	17.2	1.31	13.4	1.39	10.9
Haar	2500	1.76	19.1	1.42	13.7	1.12	10.7	1.28	8.7
Haar	5000	1.46	9.5	1.18	6.8	1.02	5.3	1.09	4.4
Haar	10 000	1.30	4.7	1.05	3.4	1.00	2.7	1.06	2.2
Haar	25 000	1.07	1.9	0.97	1.4	0.90	1.1	0.97	0.87
Haar	50 000	1.00	0.95	0.85	0.68	0.85	0.53	0.92	0.44
Haar	75 000	1.03	0.63	0.87	0.45	0.86	0.35	0.90	0.29
Haar	100 000	1.05	0.47	0.82	0.34	0.86	0.27	0.88	0.22

reference MLP due to the large number of employed weak learners. Summing up, boosted stumps can be used to train classifiers that exhibit low computational complexity of classification while the generalization performance is still in the acceptable range. They are therefore ideal for rejection cascades such as proposed by [122], where the task is to quickly discard the majority of samples that have to be classified so that a more powerful classifier may be applied to only a few samples that are hard to classify accurately.

8.4 Comparison of Best Approaches

This section compares the best algorithms from the previous section with each other for a final analysis. An illustration of the overall comparison is depicted in Figure 8.13 and consists of the sparsely connected MLPs from Chapter 6, the deep MLPs of [59] discussed in Section 8.3.2, the small CNNs of [58] and the CNN committees of [129, 60] from Section 8.3.3, the SVMs trained using the virtual support vector method of [57] as analyzed in Section 8.3.4, and the boosted products of stumps using Haar filters as proposed by [133] and considered in Section 8.3.6.

The results of [57], depicted as the "DeCoste (2002)" curve in Figure 8.13, are clearly the worst from this selection, since their computational complexity is tremendously high and they do not achieve fewer misclassifications than the neural network approaches. The boosted products of stumps of [133], that is the "Kégl Haar Cubic (2009)" trajectory, achieve low computational complexity, but still involve a relatively large number of misclassifications. Their field of application is thus rather in rejection cascades than in classification of individual samples. The deep and big MLPs of [59], depicted as "Cireşan MLP (2010)", are clearly outperformed by the committees of CNNs proposed in [129, 60] from the same group and titled "Cireşan CNN (2011, 2012)" in Figure 8.13. While the latter approach yields the lowest evaluation error of 0.23% reported on MNIST so far, combination of 35 CNNs in a committee comes at the cost of high resource demand for classification. The best sparse MLP, in contrast, achieves a classification error of 0.58%, which is about 2.5 times higher than that of [60], but the sparse MLP is 460 times faster. The sparse MLP with the lowest connectivity rate in the hidden layer, represented by the bottom point of the "SMLP Elastic" curve in Figure 8.13, produces 0.70% misclassifications, roughly three times the error of the CNN committee. Due to its high sparseness of connectivity, however, it is about 1200 times faster than the large CNN committee, that is more than three orders of magnitude.

It was further shown by [58] that architectural optimization can lead to both high classification accuracy and low computational cost with CNNs. They omitted the pooling layers and used convolutions with a stride of two to simulate downsampling, which contributes greatly to complexity reduction. Since they were the first to use elastic distortions, they also obtained decent classification results, as can be seen from the "Simard (2003)" curve in Figure 8.13, where the

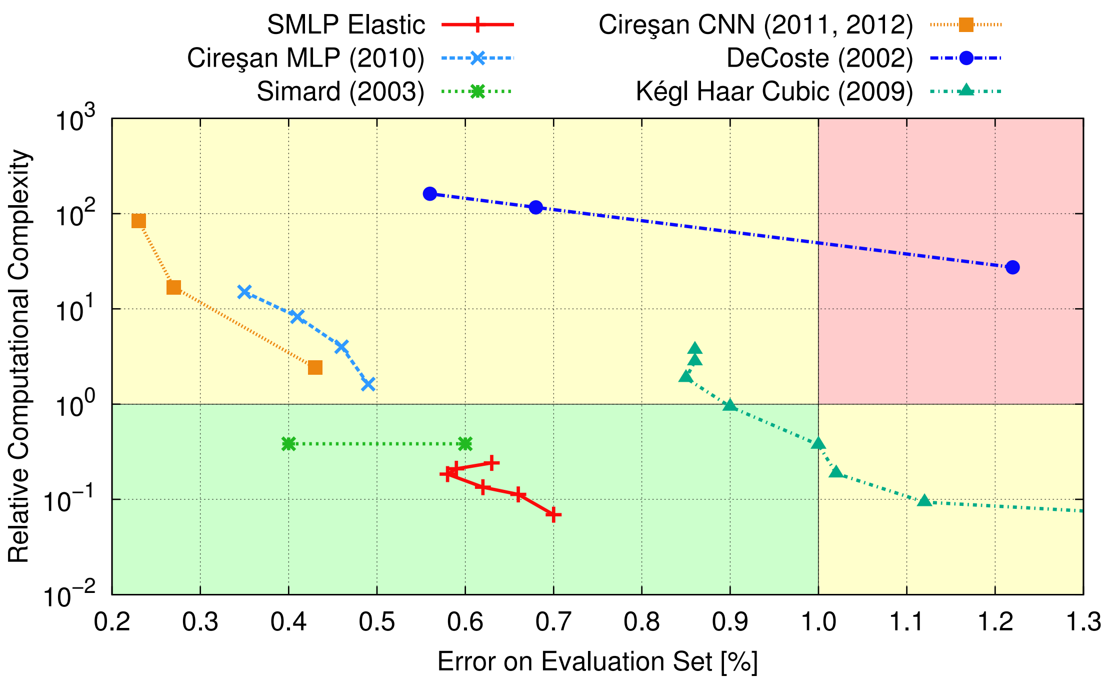


Figure 8.13: Comparison of the best results from the individual techniques.

best error rate is 0.4%. The sparse MLP with the lowest error has about 1.5 times more classification errors, but is 1.9 times faster, and the sparsest MLP has about 1.8 times more errors but is 5.1 times faster. Hence though the CNNs of [58] achieve very good classification performance and have the advantage of being designed for image inputs with a well-defined pixel topology, sparse MLPs are unparalleled in their achieved accuracy-complexity trade-off.

8.5 Discussion

This chapter analyzed the benefits of sparse information processing in Artificial Neural Networks for classification accuracy and run-time efficiency. A comprehensive statistical analysis demonstrated that the positive effect of sparse activity and sparse connectivity on the generalization performance is substantial. When either sparse activity or sparse connectivity are used, superior results compared to classical, non-sparse approaches can be achieved. Subsequently, numerous learning algorithms for handwritten digit recognition were analyzed based on two criteria: Classification accuracy and computational complexity of the recall phase. While the former can simply be estimated by application of a classifier to an evaluation set which is disjoint to the learning set and counting the number of wrongly classified samples, determination of computational complexity is more involved. To realize a reliable comparison, a simple yet

Turing-complete model of computation was defined in this chapter. The model's main characteristics are counting the number of multiply-accumulate operations and memory accesses required for running an algorithm. Reducing the operations of the investigated classification algorithms to a few main building blocks such as matrix-vector multiplications and convolutions facilitated analysis of complex architectures such as deep Multi-Layer Perceptrons and Convolutional Neural Networks.

Building on these results, a great variety of pattern recognition algorithms was evaluated, including Artificial Neural Networks, Support Vector Machines, Polynomial Classifiers, and boosting algorithms. It became evident that SVMs are too resource demanding and that Polynomial Classifiers are too weak classifiers to be competitive. Boosted classifiers were competitive when only very little computational resources are available, which makes them ideal for rejection cascades but not for classification of single samples.

Conventional Artificial Neural Networks are compelling for the investigated task of handwritten digit recognition, because they distinguish themselves from the other methods by their good classification performance in combination with moderate computational complexity. It was demonstrated that sparsely connected two-layer MLPs excel at reducing the cost for classification of individual samples by several orders of magnitude, without adverse effects on the accuracy compared to conventional, densely connected MLPs. Only more complex and specialized architectures, such as deep Convolutional Neural Networks and committees thereof achieved lower errors, but at the expense of highly increased run-time. It is however possible that these architectures might also benefit from a sparse connectivity in the same way it was proven for the perhaps most concise universal approximator, the classical two-layer MLP.

9 Night-Time Pedestrian Detection

The majority of the experiments and comparisons conducted in this work used the MNIST database of handwritten digits which is a benchmark data set and has been used thoroughly in the scientific literature to judge classification capabilities. In this chapter, the methods proposed in this work are applied to the problem of *night-time pedestrian detection from a moving vehicle* to further fortify the empirical evidence of the superiority of sparse methods over classical approaches.

Though the trend of road traffic fatalities is decreasing in high-income countries over the past five decades, road traffic injuries are still among the top ten leading causes of death and are projected to be among the top five in 2030 [134]. Pedestrians are especially vulnerable as they have no inherent protection in the case of a collision with a vehicle. Since scotopic vision is far inferior to photopic vision in terms of spatial and temporal resolution and contrast sensitivity, pedestrians are three to seven times more vulnerable at night as it is more difficult for vehicle drivers to detect them [135]. Statistics also show that the ratio of fatal collisions to total collisions is twice as high after dark, and that absence of street lighting triples this ratio compared to when the road is well illuminated [135]. It is hence desirable to automatically warn vehicle drivers of pedestrians in time such that collisions can be avoided.

This can be achieved by recording the scenery in front of the vehicle with night vision capable camera systems, detection of pedestrians in the camera images through pattern recognition algorithms, and ultimately indicating detected pedestrians through a display or a programmable headlight [136]. Headlight systems that emit aimed light beams are preferable over displays since the gaze of the driver can then remain directed to the street rather than having to alternate between the street and the display with long accommodation processes [136]. While latency times of pattern recognition systems can easily be hidden to some extent when using a monitor by delaying the displayed camera images, this is not possible with an intelligent headlight as high latencies cause pedestrians to be overlooked by concentrated light beams, rendering the entire system useless. Therefore recognition techniques with a low computational complexity are prohibitive.

The results of Chapter 8 narrow down the list of candidates to simple Multi-Layer Perceptrons and boosted Haar-like filters. This chapter proposes a heterogeneous rejection cascade approach to reliably detect pedestrians in a real-time constrained environment. Using sophisticated feature descriptors and learning algorithms, it is possible to extend system availability to complex scenarios which cannot be handled well merely by classical approaches.

The remainder of this chapter is structured as follows. First, an overview of eligible camera systems for recording the scenery in front of the vehicle is given, followed by a discussion on suitable headlight systems for a marking light. Then an approach for detection of pedestrians in camera images using sparse information processing is proposed, combining classical techniques with the optimization techniques developed in this work. The detection procedure is further evaluated in terms of recognition accuracy and processing time, and the benefit of sparse connectivity is analyzed. The chapter is concluded with a discussion of its practical results. This chapter contains material previously published in [136, 137].

9.1 Camera and Headlight Systems

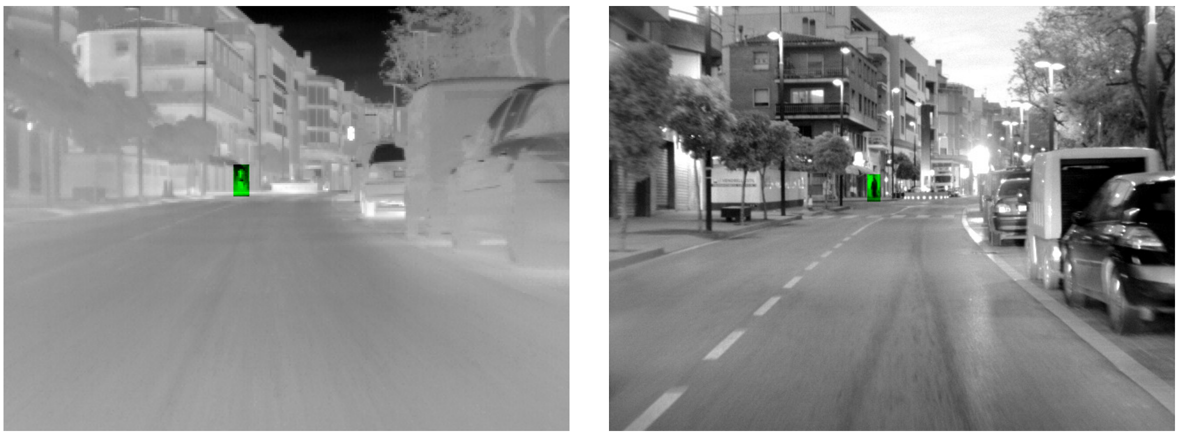
Since the task is night-time pedestrian detection from a moving vehicle, it is obvious that standard cameras which are sensitive in the visible spectrum only would not lead to desirable results. Instead, camera systems designed for the specific task have to be employed. Currently available automotive night vision systems can be divided into *active near-infrared (NIR) systems* and *passive far-infrared (FIR) systems*, where both are sensitive in disjoint regions of the electromagnetic spectrum [138].

Near-infrared cameras are sensitive up to a wavelength of 1000 nanometers, which includes parts of the visible spectrum, and need an active light source in the form of separate lighting modules in the headlights to illuminate the scenery in front of the vehicle [138]. Since the infrared light emitted by these modules is invisible to the human eye, other traffic participants receive no glare while the NIR camera has a considerable visual range.

Far-infrared cameras, on the other hand, are able to detect electromagnetic radiation with wavelengths ranging from 7 micrometers to 12 micrometers and are completely passive in that they need no special lighting [138]. FIR systems produce thermal images and are hence ideal for pedestrian detection since the heat of the human body becomes apparent as high contrast region in the imagery, while the visual range of FIR systems exceeds those of NIR systems by a large margin [139].

Sample photographs of both sensor types are shown in Figure 9.1. Because the similarity of NIR images with the human visual perception is stronger than that of FIR images, a combination of both techniques would facilitate both good detection performance and the display of an intuitive image to the driver [139]. Both sensor types feature different characteristics and hence it is beneficial to combine their sensory data to yield a detection system that features greater accuracy than if only one of the sensors is used [140].

To illuminate detected pedestrians with a narrow light beam, it is possible to either introduce a lighting module exclusively intended for marking purposes, or use a common lighting module which also generates the low beam and high beam light distributions [141]. A separate lighting module is impractical due to the increased space requirements in the headlight and the involved



(a) Far-infrared photography. (b) Near-infrared photography.

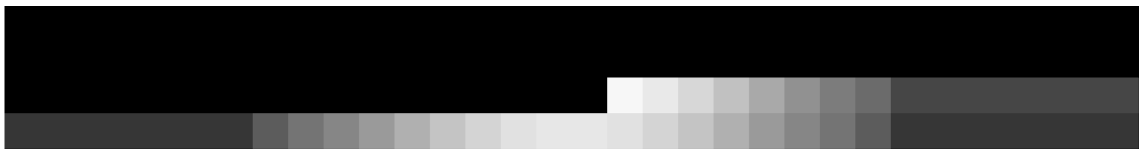
Figure 9.1: Example of pedestrian detection in a challenging scenario. The camera images were recorded in an urban area near Barcelona, Spain, about one hour before sunset on a day with temperatures exceeding 20 °C. These conditions render the detection task difficult due to highly structured backgrounds in both camera images. The pedestrian detected by the proposed system is highlighted with a green box. It is almost impossible for a human driver to recognize a person at that distance while occupied with maneuvering the vehicle.

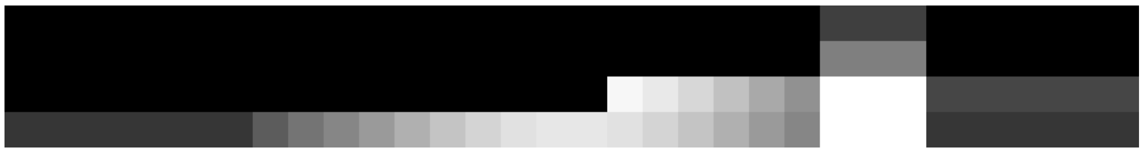
costs [141]. Since the marking light beam has to be aimed for the target to be illuminated, conventional mechanical headlight systems have the disadvantage of a non-negligible aiming speed of only about twenty degrees per second [141].

Freely programmable headlights based on semiconductors constitute a viable alternative because they offer the freedom to arbitrarily configure the light distribution with negligible response time [142]. With such light systems it is possible to implement functions like a dynamic bending light that alters the light distribution on a global level in conjunction with local light distribution changes without adverse effects on lighting homogeneity [136].

Exemplary light distributions of a freely programmable headlight constructed by [142] are depicted in Figure 9.2. Here, more than one hundred pixels can be controlled electronically to produce arbitrary patterns. Because of the short response time it suffices to compensate for driving dynamics effects on a small time scale only, provided the processing time of the pedestrian detection system is small enough [136].

Since every false alarm of the pedestrian detection system is quite noticeable, trading off processing time for detection accuracy is challenging. Needless to say that approaches such as sparsely connected MLPs which are known to reduce the computational cost of the recall phase of classification systems while even improving generalization capabilities are appealing for this real-world application.


(a) Conventional low beam light distribution.


(b) Low beam light distribution combined with a marking light beam.

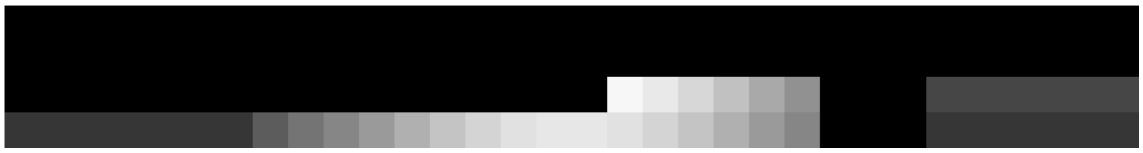

(c) Low beam light distribution where the marking light beam has been cut out.

Figure 9.2: Exemplary light distributions that can be produced by a freely programmable headlight. The main lighting module of such a headlight consists of an array of light emitting diodes (LEDs), arranged in four rows and thirty-two columns, where each element can be controlled independently. The pixels of the depicted images correspond to the individual LEDs, and the pixel intensity is proportional to each element's luminosity. A projection lens in front of the LED array provides an opening angle of about one degree in horizontal and vertical direction for each light element. The light distributions from (b) and (c) can be alternated to produce a flashing marking light beam. Adapted from [136].

9.2 Pedestrian Detection through Sparse Image Processing

As outlined in the previous section, combination of the sensory data from both an FIR camera and an NIR camera promises to yield the best detection performance [140]. Moreover, hard timing constraints require fast but accurate classification techniques. As already discussed in Chapter 8, boosted Haar-like features offer a good compromise between classification accuracy and computational complexity. Since the task considered in this chapter is a detection system rather than a pure classification system, these simple filters can be used in a rejection cascade to quickly thin out regions in the camera images that certainly do not contain instances of the target class [122, 140].

Although there are strong theoretical guarantees on error bounds of boosting approaches [122], they all depend on the discriminative capabilities of the available features. Hence late stages of the rejection cascade require a large amount of weak learners to further reduce candidate regions until only the relevant objects remain since the regions might also contain objects that

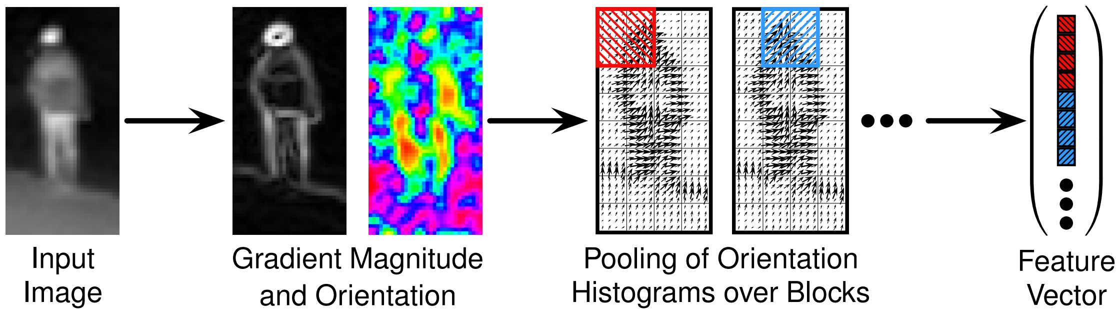

Input Image
Gradient Magnitude and Orientation
Pooling of Orientation Histograms over Blocks
Feature Vector

Figure 9.3: Principle of Histograms of Oriented Gradients (HOG) feature computation. First, the gradient magnitude and orientation are computed from the input image patch. Then, histograms based on the gradient information are computed for each cell, where the gradient orientation is mapped to several bins. The histograms are pooled over blocks, normalized and concatenated for the final feature vector.

are similar to but are not instances of the target class. The combination of a rejection cascade to discard large regions of the input images and more sophisticated classifiers to accurately predict class membership of remaining image regions is therefore a perspicuous approach.

In order to realize a real-time capable pedestrian detection system with high accuracy it is therefore proposed to use a rejection cascade with heterogeneous structure. The early layers should use the well-proven combination of Haar-like features that can be computed efficiently using an Integral Image and a boosted classifier [122], see also Section 8.2.4. Subsequent layers should work directly on the image pixels and classify them with sparsely connected MLPs. The final decision-making should be based on sophisticated feature values developed for pedestrian detection. In this work, the *Histograms of Oriented Gradients (HOG)* descriptor [143] is employed, which was shown empirically to be the state-of-the-art feature type for day-time pedestrian detection [144].

The HOG features are computed by first deriving gradient orientation and magnitude for every pixel in the input image patch, computing histograms over the gradient orientation for each cell spatially combining several pixels, and ultimately normalizing and combining the cell histograms for each block, which spatially combines several cells [143]. The block information is concatenated into a single feature vector. Hence, cell histograms are used more than once when blocks overlap, albeit normalized differently, which results in a certain redundancy of the representation. An illustration of the HOG feature computation is given in Figure 9.3.

Improvements to the detection performance in the day-time situation could only be achieved in combination with other feature types such as color or motion information in an exhaustive evaluation [144]. Although [140] avoided using HOG features due to small window sizes induced by a long-range detection constraint, it should be noted that merely boosting algorithms were used to greedily construct classifiers that use a sparse selection of the available features.

In the situation considered there, it is possible that Haar-like filters are more appropriate due to their simple structure. The experimental results in this chapter will however demonstrate that classifier training with projected gradient descent methods does not suffer from the high redundancy in the HOG feature descriptor and does achieve excellent recognition capabilities even for remote pedestrians.

9.2.1 Description of Data Sets

A number of sequences containing synchronized FIR and NIR footage were recorded using a vehicle equipped with cameras sensitive in the appropriate regions of the electromagnetic spectrum. Disjoint subsets of these sequences were selected to serve as learning set and evaluation set, respectively. To ensure both data sets were challenging, only sequences recorded at moderate to high ambient temperatures and with a high variability of image structure as encountered in urban areas were used. The contrast of pedestrians to the background is tremendous in FIR images recorded at low ambient temperatures or in images of both sensor types without any cluttered background. Since simple Haar-like filters would lead to a reasonable detection performance, such scenarios were excluded to allow for estimates of detection performance when the conditions are far from optimal. It was further ensured that the days on which the sequences for the evaluation set were recorded were not correlated to the recording days of the learning set to facilitate an unbiased estimate of the detection performance.

This resulted in 576 sequences for learning containing about 247 000 pairs of FIR and NIR images, and 165 sequences for evaluation with about 55 000 image pairs. This translates roughly into 137 minutes and 30 minutes of footage for both data sets, respectively. Although the sequences were manually labeled so as to provide a set of rectangles for each image which describe the bounding boxes of pedestrians, a post-processing step was necessary to identify bounding boxes which belonged to only partly visible pedestrians or to those too distant to be recognizable even by a human expert. Those rectangles were marked as *neutral labels*, that is image regions where a detection system is neither rewarded nor penalized if it signals the presence or absence of a pedestrian. All other bounding boxes remained as *positive labels* which denote image regions where the system must emit a positive response. The complement of the positive and neutral regions is the background, where no detection should occur. In total, there were about 143 000 positive and 101 000 neutral labels in the learning set, and the evaluation set contained about 37 000 positive and 50 000 neutral labels.

9.2.2 Sample Generation

The pairs of camera images should be processed using a rejection cascade in a sliding window fashion, or using a more sophisticated technique such as a tree-based coarse-to-fine search strategy [140]. Based on the alignment between the two cameras, pairs of corresponding search windows are generated for each image pair and fed to a chain of classifiers. Constraints on the position and shape of the search windows can be derived from domain knowledge. Since pedestrians are supposed to be encountered in upright position, the aspect ratio of the search windows can be fixed to one to two. With a ground plane assumption and the known alignment of the cameras to the moving platform, a constraint on the vertical position can be augmented.

Upper and lower bounds on the search window size were derived from the expected body height of adult pedestrians, from the maximum distance in which pedestrians should be identified, and from the characteristics of the optical systems of the employed cameras. A maximum detection performance of 120 meters was assumed, which corresponds to 4.3 seconds of time at a traveling velocity of 100 kilometers per hour, which is enough to warn the driver with a large safety margin. The minimum body height was assumed to be 1.60 meters, which covers a great portion of the adult population. Smaller pedestrians can of course be detected, albeit with a decreased detection range. These assumptions led to a minimum search window height of 16 pixels in the FIR images and 28 pixels in the NIR images. These numbers deviate from those given by [140] since other camera optics with a different focal length were employed. Since allowances for the accuracy of the calibration between both cameras have to be considered, the sliding window approach results in about 1.5 million windows that have to be inspected for each image pair. As already mentioned, the number of effectively analyzed windows and hence the processing time for pedestrian detection can be reduced using a coarse-to-fine search strategy.

A rejection cascade is trained greedily in a layer-after-layer fashion [122]. For each new layer, training samples are generated by classifying the search windows of the learning images with the previously learned layers and remembering those that gave rise to a detection. The empty classifier necessary for the very first layer always reports a detection by convention. All detected search windows are then assigned a teacher signal based on the overlap with the positive labels in the images. When a search window shares an image region with a neutral label, it is discarded and not used for classifier training. Since there is a huge number of potential negative samples for each image pair, only a randomly chosen subset is stored to conserve memory. In the experiments described below, at most one hundred negative samples for each image pair were used, such that at most about 25 million negative samples were available. The number of positive samples is much smaller, no subset selection was necessary as a total of about 7 million positive samples could be extracted.

9.2.3 Pre-Classifier Training

In order to generate a learning set that contains only hard negative samples suitable for more complex classifiers, a *pre-classifier* in the form of a few layers of a rejection cascade was trained on the images from the learning set sequences. Here, the decision threshold of each layer was adjusted to achieve a fairly high true positive rate of 99.5%. The false positive rate should be low to quickly discard uninteresting regions of the input images while retaining the majority of the regions that potentially contain pedestrians [122]. In doing so, the number of potential training examples for higher layers is greatly decreased, in turn modifying the distribution of the training samples. This becomes apparent from Figure 9.4, where negative training samples from the original images are shown, together with negative samples that are falsely classified as positive by the pre-classifier. The hard samples clearly feature more structure, especially in the form of vertical bars that resemble blurred versions of actual pedestrians, shown in the bottom row of Figure 9.4.

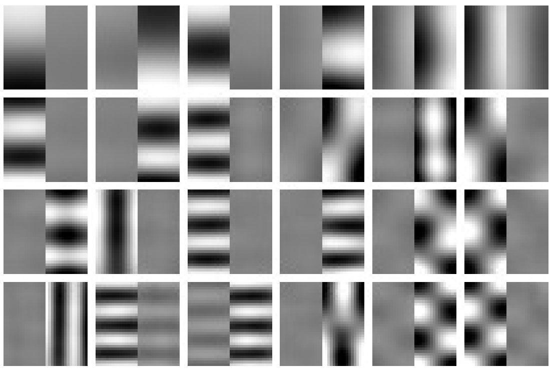
The very first layer of the pre-classifier only compares the ratio of the empirical standard deviation of the search windows in both the FIR and NIR image with a learned threshold to discard homogeneous regions with hardly any structure. The threshold was determined to achieve a 99.5% true positive rate on all the learning samples, which resulted in a false positive rate of 95.7%. Since the standard deviation alone is not a well-discriminating feature, lower false positive rates cannot be expected in highly structured camera images, but discarding regions with low standard deviations prevents learning samples from becoming too noisy when performing a normalization to zero mean and unit variance.

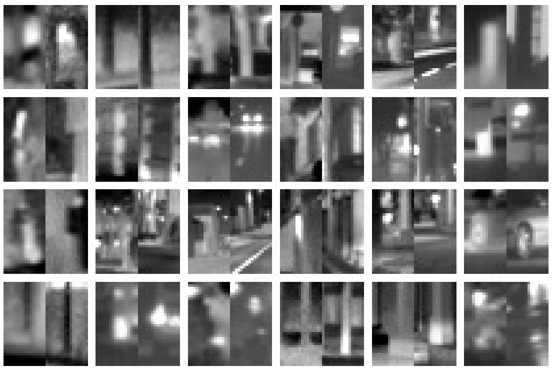
The next five layers of the pre-classifier, indexed by numbers from two through six, were trained with the Adaptive Boosting algorithm on the responses of Haar-like filters [122]. This learning algorithm is impractical for a very large number of training samples, to speed up optimization 2 million positive samples and 4 million negative samples were randomly chosen from the available pool. The set of possible Haar-like filters was created for a base image size of 32×16 pixels. Each box within the filters was ensured to cover at least two pixels, and the total filter size was ensured to be even to reduce the number of overall filters, which helps prevent overfitting. In total, 6950 Haar-like filters were available for each of the two input streams and the boosting algorithm could greedily select among 13 900 filters.

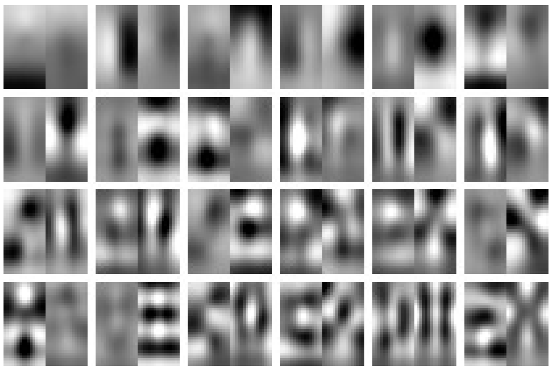
During training of the strong learners, the decision threshold was adjusted after each weak learner training for a minimum true positive rate of 99.5%. Strong learner training was terminated when either the false positive rate on the learning set fell below 50% or a preset number of maximum weak learners was exceeded. Table 9.1 shows the results of the boosting trainings for each layer, where in the first and third layer the maximum number of weak learners was exceeded and in the other layers the target false positive rate was reached. By multiplication of the detection statistics, it can be estimated that after the first six layers a false positive rate of 3.8% at a true positive rate of 97.0% is achieved on the learning set.

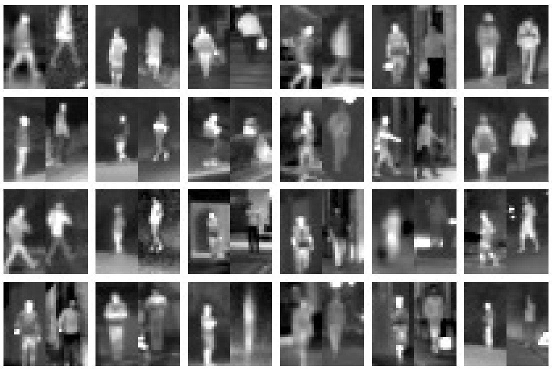
Although using more layers trained with a boosting algorithm could further lower the false positive rate provided the features are discriminative enough [122], this would lead to strong learners with a prohibitively large number of weak learners. This would not only result in a long training time since each weak learner involves one pass through the entire learning set, but also in an increased computational complexity during the recall phase of the classifier. It is therefore expedient to replace the learning algorithm for the rejection cascade layers with

(a) Subset of negative samples.


(b) Principal components of negative samples.


(c) Subset of hard negative samples.


(d) Principal components of hard negative samples.


(e) Subset of positive samples.

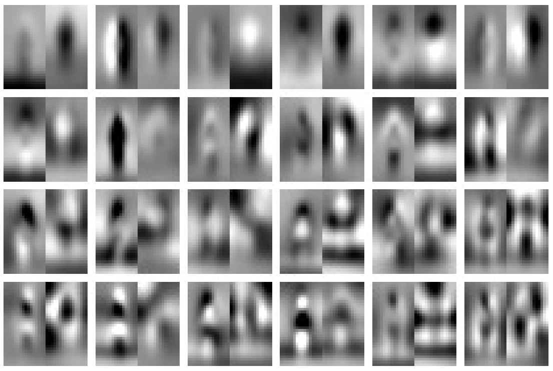

(f) Principal components of positive samples.

Figure 9.4: Visualization of the learning samples for the pedestrian detection system, where the first half of the images corresponds to FIR imagery and the second half corresponds to NIR imagery. The negative samples consist of all search windows in the camera images that do not contain a pedestrian, and the *hard* negative samples belong to the subset of all negative samples which have been incorrectly predicted as positive by the pre-classifier. The positive samples correspond to all windows that exhibit a significant overlap with the positive labels in the camera images. The principal components indicate that the entirety of the negative samples possesses no particular structure, while the hard negative samples follow a distribution similar to that of the positive samples.

Table 9.1: Statistics of the five layers of the pre-classifier that were trained by boosting of Haar-like filter responses. The decision threshold was modified after each weak learner training to achieve a true positive rate of 99.5% on the learning set. The termination criterion was then either achieving a false positive rate of less than 50% on the learning set, or exceeding a maximum number of weak learners.

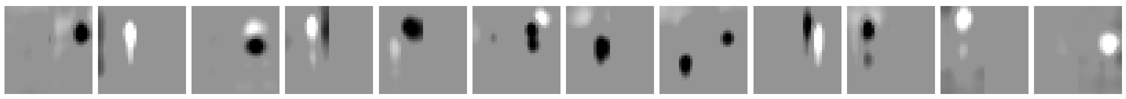
Layer index	2	3	4	5	6
Total number of weak learners	4	13	25	53	74
Weak learners in FIR stream	3	10	18	37	52
Weak learners in NIR stream	1	3	7	16	22
False positive rate [%]	61.4	48.8	53.4	49.9	49.4

one that does not suffer from these shortcomings now that the amount of data was already decreased to $1/25$th of the original volume.

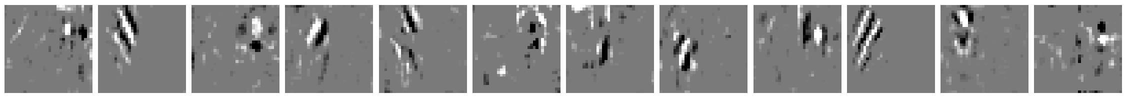
Because of their low computational cost at high classification accuracy, sparsely connected MLPs were used for the remaining layers of the pre-classifier. The features for the SMLPs were created from the raw pixel values as follows. First, from both the FIR and the NIR images the corresponding search windows were cut out and re-sized with bicubic interpolation to yield images with 32×16 pixels. These were then normalized to zero mean and unit variance independently, and then vectorized and concatenated to yield a single feature vector with 1024 entries. The neural networks were optimized using online learning, so no subset selection of the learning samples was necessary.

The parameters of the training algorithm, that is the number of hidden units, the degree of connectivity sparseness, and the step size for gradient descent were determined through two-fold cross-validation. The maximum number of hidden units was forced to be less than a maximum value to prevent exaggerated computational complexity, which could hinder the entire detection system from being real-time capable. This maximum value was set to 64 for the first cascade layer trained with the SMLP optimization algorithm. The cross-validation was repeated five times on each tuple of feasible training parameters to allow for a statistical robust estimate. Instead of simply using the classification error on the validation set to determine the winning parameter set, the area under the ROC curve was used. This number is more appropriate because it does not depend on a concrete decision threshold or the frequency of occurrence of samples with distinct class labels [145].

The best combination of training parameters was using 32 hidden units, a sparseness degree of connectivity of 0.80, and a step size of $5 \cdot 10^{-4}$. After the training on the entire learning set was carried out with these parameters, the decision threshold was altered to obtain a target true positive rate of 99.5%. Subsequently, the synaptic connections in the hidden layer were pruned to increase sparseness, resulting in a connectivity rate of 15.3% and a false positive



(a) Filters after SCFC initialization.



(b) Filters after SMLP training.

Figure 9.5: Visualization of the filters from SMLP training, where the first half corresponds to FIR imagery and the second half corresponds to NIR imagery. After pre-training with SCFC, the filters resemble localized blobs, mostly situated in only one of both channels. After fine-tuning and post-processing, all filters look like contrast fields, similar to the results on handwritten digit recognition.

rate of 22.7% on the learning set. This is equivalent to an overall false positive rate of 0.86% at a true positive rate of 96.6%. Although this false positive rate may already seem very low, this number translates to about 13 000 search window pairs of the total 1.5 million pairs which are still recognized to contain a pedestrian although there is none present.

To further decrease the number of false positives the pre-classifier produces, another sparse MLP was added to the rejection cascade. To account for the increased difficulty of the classification task, the maximum allowed number of hidden units for cross-validation was set to 512. After cross-validation, however, it turned out that using more than 128 hidden units did not improve classification accuracy dramatically, and hence only 128 hidden units were used for the final training. The other parameters that maximized the area under the ROC curve were a connectivity sparseness degree of 0.75 and a step size of 10^{-3}.

A subset of the resulting filters is depicted in Figure 9.5, divided into the starting values after pre-training and the final weights after fine-tuning and post-processing. It can be seen that in most filters the majority of the non-vanishing entries reside in either the FIR half or the NIR half. After pre-training, the filters possess a localized blob-like structure, and in some cases adjacent regions with opposite signs are present. After classifier training, all filters resemble contrast fields, similar to the experiments on handwritten digit recognition. A great portion of the filters contain diagonal gratings, while others are reminiscent of bright, round regions with circumjacent negative regions. Considering the positive samples shown in Figure 9.4, such filters may serve to discriminate humans from the background as the former frequently appear bright in the FIR and NIR imagery.

The new layer of the rejection cascade achieved a false positive rate of 14.6% at a true positive rate of 99.5%, with a connectivity rate of 13.9% in the hidden layer. In terms of search window pairs, this accounts for about 1900 remaining false positives per image pair. This number seemed low enough to make use of the HOG feature descriptor, which is expensive to compute but which promises better discriminatory capabilities than feeding raw pixel values to a neural network.

9.2.4 Final Cascade Layer

The final layer of the rejection cascade should use the Histograms of Oriented Gradients feature descriptor [143] in conjunction with a sparsely connected MLP. Using the previously learned pre-classifier, a total of 6.7 million positive samples and 18.9 million negative samples could be extracted from the images of the learning set for optimization.

The feature values from each search window pair were computed by cutting out the corresponding image regions, rescaling them to 32×16 pixels with bicubic interpolation and then computing the HOG features for FIR and NIR independently and concatenating them into a single feature vector.

The parameters of the HOG descriptor were determined as follows. The cell size was chosen to be 4×4 pixels, hence there were eight cells in the vertical and four cells in the horizontal direction. Smaller cell sizes would have led to extremely sparse orientation histograms, while larger cell sizes would impair the spatial resolution due to the small input image patch size. The block size was set to 2×2 cells, the histograms in each block were normalized to unit L_2 norm, and adjacent blocks were chosen to overlap by one cell. There were thus $7 \cdot 3 = 21$ blocks in total, each block had four histograms with 9 orientation bins, totaling in 756-dimensional feature vectors for each of the two input images, or 1512-dimensional features for both images.

The features computed using this parameterization were then used to train Multi-Layer Perceptrons. First, a conventional, densely connected two-layer MLP was used to obtain baseline results. Using two-fold cross validation, the steps size for gradient descent and the number of hidden units was determined as those that maximized the area under the ROC curve in the median of five repetitions. It was found that 64 hidden units and an initial step size of $5 \cdot 10^{-4}$ yielded the best validation performance. Here, more hidden units degraded the performance, which is likely to result from overfitting since the HOG descriptor is already a well-discriminating feature vector for pedestrian detection. After cross-validation, five MLPs were trained with the backpropagation algorithm for later evaluation.

Moreover, the training methodology for sparsely connected MLPs as proposed in Chapter 6 was carried out using the same number of hidden units. Two-fold cross-validation yielded an initial step size of 10^{-3} optimal for the target degree of sparse connectivity σ_W chosen from the set $\{0.60, 0.65, \ldots, 0.90\}$. Five SMLPs were trained for each value of σ_W from this candidate set. During the post-processing stage of SMLP training, only that amount of synaptic connections was removed to allow the false positive rate to increase by at most 2.5% at a true positive rate of 99% on the learning set.

9.3 Evaluation

This section evaluates the proposed pedestrian detection system in terms of detection accuracy and processing time. It is most intuitive and meaningful to incorporate all the layers of the rejection cascade in the analysis of the detection performance, since a tree-based coarse-to-fine search strategy is employed here to reduce run-time. Here, a node in the tree of candidate search window pairs is only traversed when the rejection cascade recognizes an object of the target class in the search window pair corresponding to the node's parent [140]. Thus, data dependence renders the analysis of the performance of individual layers impossible. Therefore, a search strategy based on the sliding window fashion is used afterwards to assess the discriminatory capabilities and time requirements of each layer independently.

9.3.1 Detection Performance

When the proposed detection algorithm is applied to pairs of FIR and NIR images, it produces a list of *detections* containing bounding boxes in each camera image and associated confidence values. The confidence values are here equal to the output value of the neural network from the last layer of the cascade. They are numbers from the interval $[0,\ 1] \subseteq \mathbb{R}$ that can be interpreted as probabilities of class membership of the input samples [53].

Since there are usually multiple overlapping detection windows in the images, they should be merged for a concise display which can easily be understood by the vehicle driver while maneuvering the vehicle or for the control unit of the headlight which operates the marking light beam. It is moreover beneficial to combine nearby windows such that groups of two or more pedestrians can be more easily recognized. Standard methods for bounding box merging do not support this grouping feature, hence a simple approach based on graph partitioning was implemented [137]. The method accumulates the confidence values of the merged boxes, introducing a new feature which can be used to suppress detections which are likely to be false alarms [137].

Each image pair was analyzed independently of the others. First, the rejection cascade was applied to the images and its detections merged. Since the merging operation also combines windows for groups of pedestrians, this has to be handled explicitly. Hence, it was ensured that at most one quarter of the area of each merged rectangle belonged to image regions not marked with positive or neutral labels. If this was not the case, then the detection window was treated as a false alarm in further processing. Accordingly, overly large detection boxes are punished, for example the extreme case of one large box covering the entire image would result in one false alarm per image pair in the mean. Subsequently, it was checked whether at least three quarters of each positive label's area was covered by a detection window. Only if this applied was the pedestrian recorded as detected; otherwise the false negative count was increased.

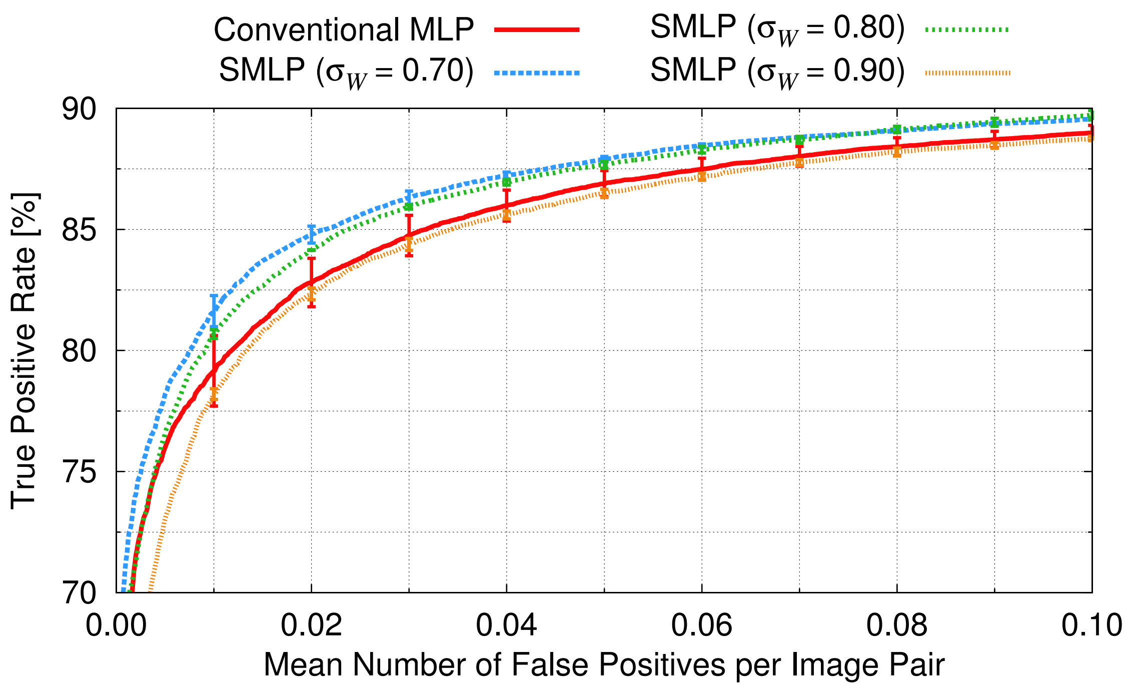


Figure 9.6: ROC curves of the complete detection system, where the training algorithm in the final layer of the rejection cascade was varied. Sparsely connected MLPs with reasonably chosen sparseness degrees not only boost the true positive rate for any given false positive rate, but are also more reliable as the performance scatter is significantly lower than when a conventional, densely connected MLP was used.

The overall statistics on all image pairs from all evaluation sequences could then be used to draw ROC curves. Here, the curve emerged by shifting a threshold value applied to the confidence value of the merged detection windows. The abscissa denotes the mean number of false positives per image pair, it resulted from the conventional false positive rate by normalization using the absolute number of false alarms and the total number of image pairs. The ordinate describes the true positive rate, that is the percentage of positive labels correctly detected with the minimum overlap described above.

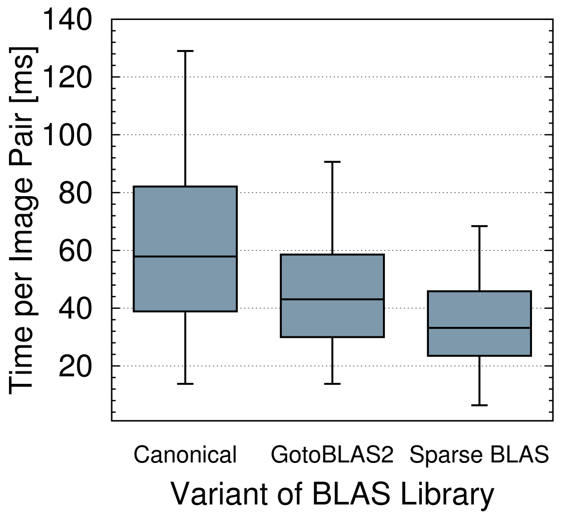
The ROC curves for the rejection cascade where the learning algorithm in the final layer was varied is depicted in Figure 9.6. There were five classifiers available for each variant. The curves belong to the one that achieved the median area under the ROC curve on the evaluation set, and the error bars denote $\pm$ one standard deviation distance. The SMLPs with $\sigma_W \in \{0.60, 0.65, 0.75, 0.85\}$ were omitted in the graphics to avoid clutter; numerical results are reported in Table 9.2. It is evident that except for the extreme case of $\sigma_W = 0.90$, sparsely connected MLPs outperform the conventional MLPs by a large margin. For example, for a mean number of 0.02 false positives per image pair, the conventional MLP is able to detect 82.8% $\pm$ 1.0% of the pedestrians, whereas the SMLP with $\sigma_W = 0.70$ achieves a true

Table 9.2: Detailed results of the detection performance for different parameterizations of the final layer's learning algorithm. Here, the target degree of sparse connectivity σ_W was varied, resulting in different connectivity rates. When sparse connectivity is involved, about 55% of the active synaptic connections use feature values from the FIR images. The last three columns give statistics obtained for different working points, specified by the mean number of false positives per image pair (FPPIP).

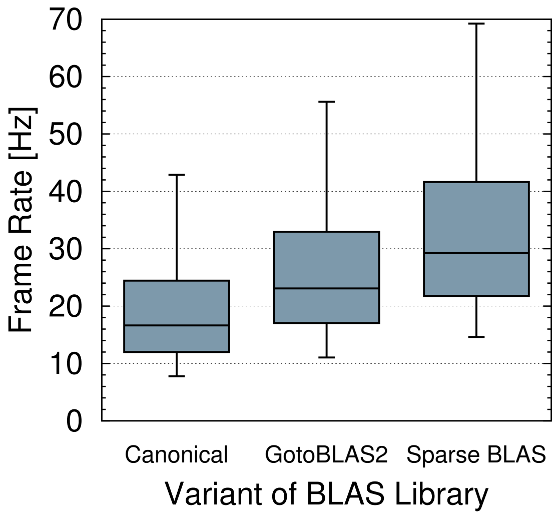
Sparseness degree σ_W	Connectivity rate in hidden layer [%]	Amount of active synaptic connections FIR/NIR [%]	Area under ROC curve for FPPIP ≤ 0.10 [%]	True positive rate [%] at an FPPIP of 0.02	True positive rate [%] at an FPPIP of 0.01
none	100 ± 0.0	50.0/50.0	85.2 ± 0.7	82.8 ± 1.0	79.2 ± 1.5
0.60	19.4 ± 0.7	55.2/44.8	86.4 ± 0.2	84.8 ± 0.3	81.7 ± 0.8
0.65	16.3 ± 0.5	54.9/45.1	86.4 ± 0.3	84.8 ± 0.4	81.5 ± 0.7
0.70	11.6 ± 0.3	54.8/45.2	86.5 ± 0.2	84.8 ± 0.4	81.6 ± 0.6
0.75	8.7 ± 0.3	54.7/45.3	86.2 ± 0.2	84.4 ± 0.2	81.0 ± 0.1
0.80	7.1 ± 0.3	55.3/44.7	86.1 ± 0.1	84.1 ± 0.1	80.7 ± 0.2
0.85	4.8 ± 0.1	57.2/42.8	85.7 ± 0.3	83.6 ± 0.4	80.0 ± 0.6
0.90	2.6 ± 0.2	58.8/41.2	84.6 ± 0.1	82.3 ± 0.3	78.2 ± 0.2

positive rate of $84.8\% \pm 0.4\%$. Note that the true positive rates include pedestrians at a distance of up to 120 meters from the vehicle in challenging scenarios. The SMLP training algorithm hence behaves similarly as in the experiments on the MNIST data set, even when a completely different set of feature vectors is used.

Moreover, the connectivity rate in the hidden layer is quite low. For example when $\sigma_W = 0.70$, less than one eighth of the synaptic connections are still active. This is less than for the experiments conducted on the MNIST data set in Chapter 6, contrary to the intuition that Hoyer's sparseness measure σ should give higher values for larger vectors with the same number of nonzero entries, see Section 2.2.3, but can be explained from the high allowance of 2.5% in terms of the false positive rate during pruning and the high redundancy of the HOG feature descriptor due to cell histogram reuse. It can further be seen from Table 9.2, that about 55% of the active synaptic connections in the hidden layer connect to the FIR half of



(a) Time per image pair in milliseconds required for the entire pedestrian detection procedure.



(b) Resulting frame rate in Hertz. Note that the camera sensors capture images at 30 Hertz.

Figure 9.7: Measurements of the processing time of the rejection cascade for three different BLAS implementations. Clearly, the highly optimized GotoBLAS2 library improves upon a canonical implementation, but exploiting sparse connectivity using Sparse BLAS outperforms even this machine-specific library.

the combined feature vector. Therefore, both sensors are used roughly equally by the rejection cascade's final layer, the slight imbalance might occur due to the better discrimination in the FIR features on account of the strong contrast from body heat.

The overall processing time required to detect pedestrians in an image pair was measured using the detector that achieved the median detection performance for $\sigma_W = 0.70$ in the final layer. For this, an Intel Core i7-980X processor was used, and three different implementations of the matrix-vector product from the Basic Linear Algebra Subprograms (BLAS) library [74], which is required to compute classification decisions in the sparsely connected MLPs. The results are depicted in Figure 9.7, averaged over all image pairs from the evaluation set.

The "Canonical" BLAS library corresponds to a reference implementation of Algorithm 8.1, where the sparse structure of the weight matrices was ignored. Since this implementation was not tuned to the concrete processor, it was the slowest with a median processing time of 57.9 milliseconds. Using the "GotoBLAS2" library [75, 76] which was optimized for the employed processor but not for sparse matrix-vector products, the run-time could be reduced to 43.1 milliseconds in the median. The lowest time requirements are those for the implementation of the canonical computation of the sparse matrix-vector product, that is Algorithm 8.2 used in the "Sparse BLAS" library for the modified ELLPACK storage format, which is the only one of the three implementations to actually make use of sparseness. This achieved a median run-time of 33.2 milliseconds, which is about 75% faster than the canonical implementation and 30% faster than the highly optimized GotoBLAS2, even though the Sparse BLAS library was not optimized for the processor used. Moreover, exploiting sparseness leads to a median frame rate of 29.3 Hertz, in 75% of all images the frame rate is greater than 21.8 Hertz, and the minimum frame rate is 14.6 Hertz attained on the images with very complex backgrounds or numerous pedestrians. The proposed approach therefore facilitates real-time capable pedestrian detection on current hardware with adequate detection accuracy, which demonstrates that a system suitable for the effective protection of vulnerable traffic participants can be built using a rejection cascade with heterogeneous structure.

9.3.2 Dynamics of the Rejection Cascade

A detailed analysis of the rejection cascade's dynamics during the detection phase was conducted to evaluate the heterogeneous structure and the detailed impact of sparse connectivity on the run-time. The detector with $\sigma_W = 0.70$ in the final layer that achieved the median evaluation performance was used for this. An overview of the characteristics of each individual layer and the obtained statistics is given in Table 9.3. The layers one through six are essentially a classical cascade [122] and are denoted collectively as the *front-end*. The remaining layers seven, eight and nine use different feature types and sparsely connected MLPs are employed for classification. These layers are called the *back-end* of the cascade.

To gather data for analysis, the entire cascade was split into nine individual detectors by keeping only the first layers. Next, each of the nine detectors was applied to the entire evaluation set in a sliding window fashion. Here, no sophisticated coarse-to-fine search was used since this would require the entire detector on account of data dependence. After application of each partial detector to an image pair, the number of search windows that gave rise to a detection and the elapsed time during processing was recorded. The computer the detector was run on was equipped with an Intel Core i7-980X processor, and the run-time measurements were carried out repeatedly and the mean taken to eliminate effects of context changes of the operating system and timers that are imprecise in the sub-millisecond range. Because the back-end used SMLPs to classify feature values, the measurements were carried out using the optimized GotoBLAS2 library [75, 76]. The sparse weight matrices were here treated as general matrices without any particular structure. Moreover, the canonical implementation of the sparse matrix-vector product was used for matrices stored in the modified ELLPACK format described in Algorithm 8.2. This facilitated the analysis of run-time efficiency when sparse connectivity is exploited.

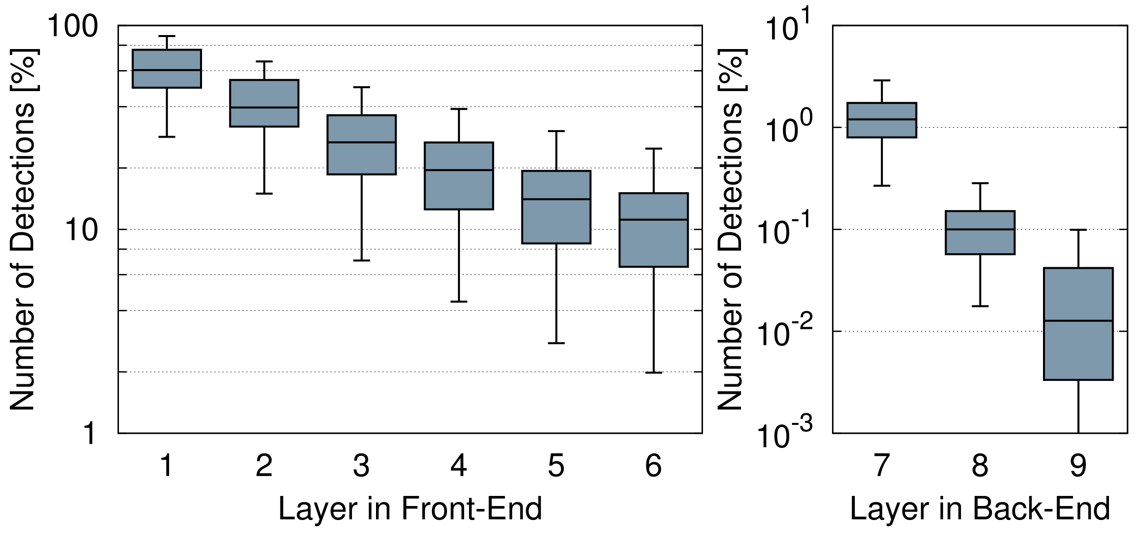
First, the number of detections each layer produced was analyzed. The total number of search window pairs was between 45 000 and 47 000 due to different camera alignments in the evaluation sequences and a rather coarse sampling frequency. Figure 9.8a shows the amount of the number of detections each layer produced relative to the total number of evaluated search window pairs, and in Figure 9.8b the ratio of the number of input search window pairs to the number of detections for each individual layer is depicted, which can be interpreted as

Table 9.3: Structure of the heterogeneous rejection cascade. The last two columns give the time each layer required to process a single search window pair, categorized by whether GotoBLAS2 or Sparse BLAS was used for the matrix-vector products. The first six layers are independent of matrix-vector products and thus the run-time is equal for both BLAS variants.

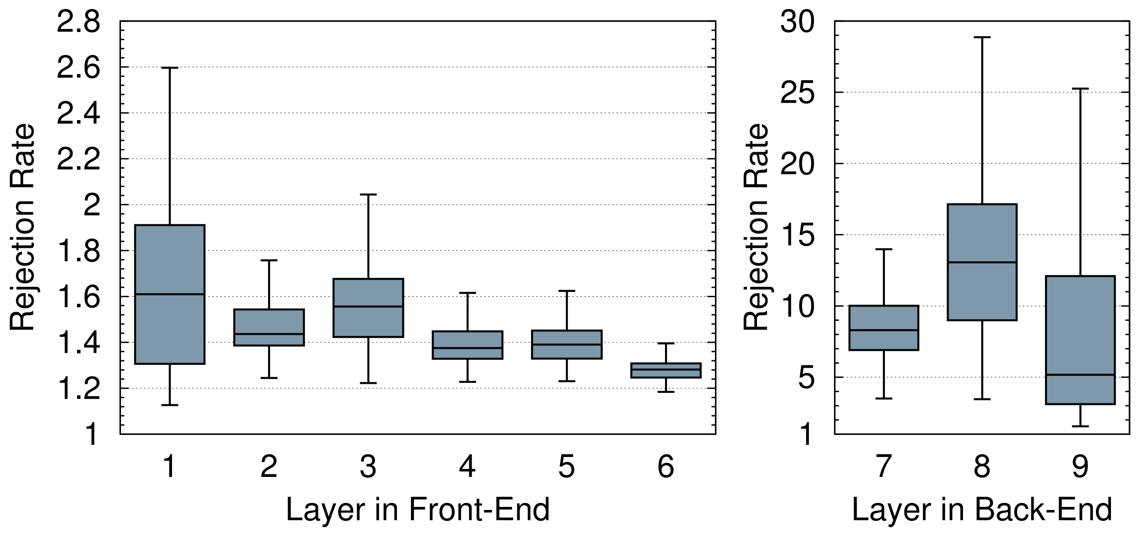
				Time per search window pair [µs]	
Layer index	Feature type	Learning algorithm	Rejection rate	GotoBLAS2	Sparse BLAS
1	Std. dev.	Threshold	1.61 ± 0.41	0.06 ± 0.01	
2	Haar	Boosting	1.44 ± 0.12	0.10 ± 0.01	
3	Haar	Boosting	1.56 ± 0.18	0.16 ± 0.01	
4	Haar	Boosting	1.38 ± 0.08	0.29 ± 0.01	
5	Haar	Boosting	1.39 ± 0.08	0.46 ± 0.03	
6	Haar	Boosting	1.28 ± 0.05	0.77 ± 0.05	
7	Pixels	SMLP	8.31 ± 2.28	5.82 ± 0.20	5.00 ± 0.07
8	Pixels	SMLP	13.1 ± 5.71	15.2 ± 0.93	10.9 ± 0.46
9	HOGs	SMLP	5.17 ± 8.67	20.1 ± 7.56	12.0 ± 6.49

rejection rate. While the layers in the front-end only slowly thin out image regions, each layer of the back-end discards a large number of irrelevant search window pairs. The graphics in Figure 9.8 had to be split for this reason to facilitate analysis. The variance in the data of each layer is rather high, which shows that the images in the evaluation set feature very different scenarios, where there may be multiple pedestrians present or none at all.

The layers in the front-end achieved rejection rates smaller than 1.6 in the median of all image pairs in the evaluation set. Only the first layer, which simply compared standard deviations of image regions with a learned threshold, significantly exceeded a rejection rate of 1.8. This can be explained by the search strategy, since here *all* image patches were analyzed, whereas during learning a maximum of one hundred samples were extracted randomly from each image pair and hence the discrepancy between the false positive rate on the learning set and the achieved rejection rate on the evaluation set is high. The layers in the back-end are far superior at sorting out image regions since here the rejection rates are several times higher than in the front-end. This is due to their better-discriminating feature values and a more sophisticated learning algorithm, capable of processing very large numbers of training samples.



(a) Amount of search window pairs classified as positive by the respective layers relative to the total amount of search window pairs, shown on a logarithmic scale. Since the final layer classified all image regions as negative on some image pairs, the graphics were cropped.



(b) Ratio of the number of search window pairs input to each individual layer to the amount of positively classified search window pairs. The layers of the back-end exhibit a completely different dynamic, they use better discriminating feature values and a more sophisticated learning algorithm.

Figure 9.8: Statistics on the rejection performance of the cascaded classifier, which was applied in a sliding window fashion to the input image pairs. The layers in the front-end were boosted classifiers using Haar-like features, they only slowly decrease the number of search windows potentially containing pedestrians. The back-end layers are far more efficient, they use sparsely connected MLPs on raw pixels or HOGs, and each layer here rejects more search windows than a combination of several layers from the front-end.

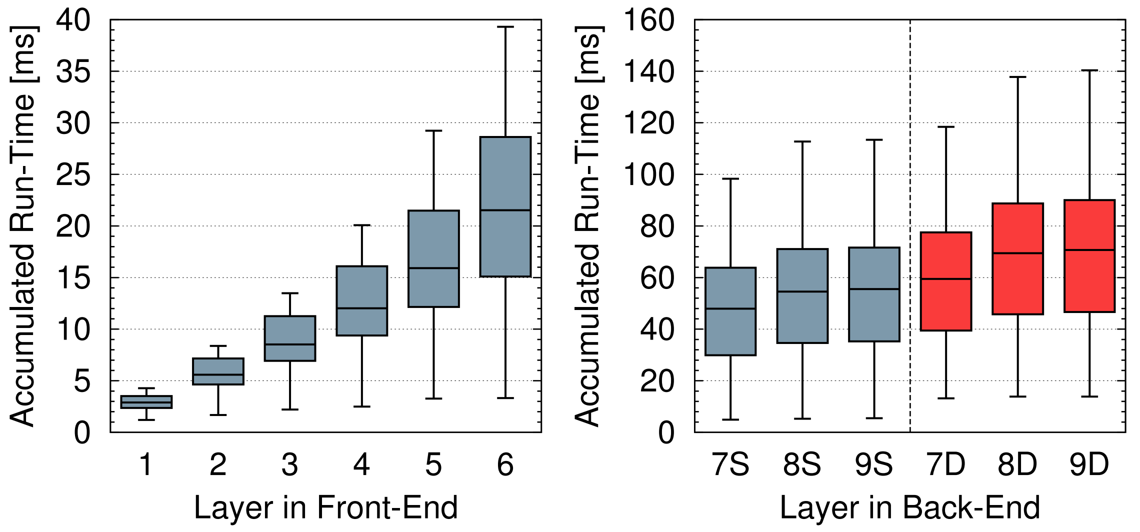
The results of the run-time measurements are depicted in Figure 9.9. Here, the data for the back-end was further divided into whether sparse computations where used, which corresponds to layers 7S, 8S and 9S, or routines from the GotoBLAS2 library which neglect sparseness for the layers 7D, 8D and 9D. The distributions of the accumulated elapsed time for each layer are shown in Figure 9.9a, which resembles the raw measurements of the nine partial detectors.

The individual run-time of each layer could be computed by subtracting the run-time of a layer from that of its predecessor. By further dividing this quantity by the number of search windows the layer was applied to, the distributions of the time needed to classify one search window were yielded. They are depicted in Figure 9.9b. Clearly, the back-end has greatly increased run-time demands compared to the front-end. It took 21.5 ± 8.30 milliseconds to evaluate the front-end in the median of all image pairs. The entire rejection cascade took 55.6 ± 23.2 milliseconds using sparse computations and 70.7 ± 27.1 milliseconds using GotoBLAS2.

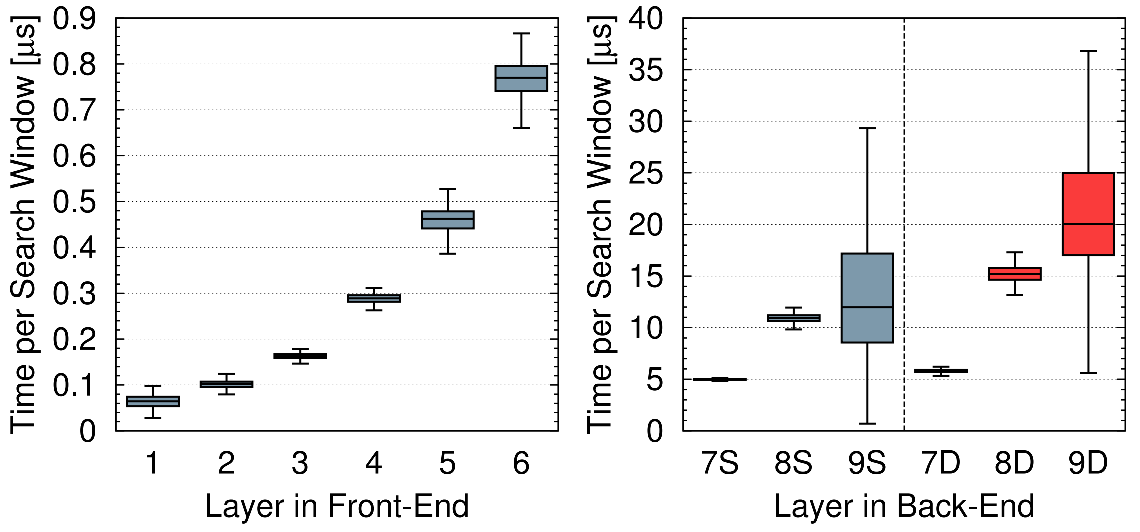
Hence also when a sliding window search strategy was used, exploiting sparse connectivity speeds up the entire detection system by about 30%. There were also image pairs which required significantly more computation time up to 113 milliseconds when sparse computations were employed. These images stem from sequences with a highly structured background and several pedestrians.

The analysis of the time needed to process a single search window pair shows a similar picture. Here, all the layers in the front-end took less than 0.77 microseconds, and the first layer of the back-end already required 5.00 microseconds using sparse computations or 5.82 microseconds using dense computations. The final layer was the most complex, requiring 12.0 microseconds per search window pair using Sparse BLAS and 20.1 microseconds with GotoBLAS2. Here, standard deviations were quite high due to imprecise timers and the relatively small amount of search window pairs that actually made it to the final layer. On the other hand, each layer in the back-end constitutes a much more powerful classifier, as the analysis of the rejection rate has shown.

A comparison with the measured increase in speed due to sparseness conducted in Section 8.3.1 shows that here the increases are not as large. This is because here the time needed for feature vector computation was incorporated in the run-time of the classifier. For layers seven and eight, image regions had to be extracted from both the FIR and NIR images and rescaled with bicubic interpolation. The final layer with index nine required the same rescaled image patches, and additionally the computation of the HOG feature values. These operations are non-trivial and incur a non-negligible cost on the run-time. Moreover, the baseline implementation for conventional matrix-vector products was the highly optimized GotoBLAS2 library. It was compared with a simple untuned implementation of the canonical algorithm for sparse matrix-vector multiplication. On real embedded systems with small memory bandwidth, faster speed-ups are likely to be achieved, especially when the implementation of the sparse matrix-vector product is optimized for the target architecture.



(a) Accumulated real time per image pair in dependence of the layer index. The graphics of the back-end were categorized based on whether Sparse BLAS was used (layers 7S, 8S and 9S) or GotoBLAS2 (layers 7D, 8D and 9D). The latter does not exploit sparse connectivity but featured machine-specific optimizations.



(b) Run-time each layer required to process a single search window pair. The measurements for layer nine were imprecise since there was only a small number of search window pairs available that were not rejected by layer eight.

Figure 9.9: Results of the run-time measurements of the rejection cascade. For (a), nine partial detectors were generated by removing trailing layers, so that the accumulated run-time could be measured. The run-time for each individual layer could be inferred by subtraction of the accumulated run-time from that of the previous layer. The data for (b) could be obtained by dividing the run-time of each layer by the number of search window pairs the layer was applied to. The layers in the back-end are more expensive to evaluate, but contribute greatly to discarding irrelevant image regions.

9.4 Discussion

This chapter studied the application of Sparse Neural Networks to the real-world task of night-time pedestrian detection from a moving vehicle in challenging scenarios, where it is beneficial to combine the sensory data of cameras which are sensitive in the near-infrared spectrum and the far-infrared spectrum, respectively. Marking of detected pedestrians with an intelligent headlight rather than only displaying the scenery in front of the vehicle on a monitor is a superior interface to the vehicle driver. This technology, however, imposes hard constraints on the detection accuracy and the processing time of the detection system, since false alarms cannot be ignored and high latencies cause the target to be missed by the light beam.

The approach proposed in this chapter extends the classical rejection cascade concept by a heterogeneous structure with regard to the feature values and learning algorithms. Here, the first few layers use the well-proven combination of boosting algorithms with Haar-like features. They are followed by sparsely connected MLPs employing raw pixel values and the Histograms of Oriented Gradients descriptor. It was observed that the filters of SMLPs trained on pixel values are morphologically different from their starting values. Blobs were transformed into diagonal gratings and circular contrast fields, similar to the results on the MNIST data set. The absolute statistics of the completed rejection cascade which used HOG features has shown that sparse connectivity improves performance compared to the conventional MLP on the entire range of the ROC curves. As both learning algorithms were provided with the exact same learning set and feature values, this demonstrates the superiority of sparse approaches on a complex task. Moreover, a run-time analysis has shown that when sparse connectivity is exploited in the inference of the classification decision using a simple implementation of the sparse matrix-vector product, then the run-time is reduced by 75% compared to a canonical implementation where sparseness is neglected, and by 30% compared to the optimized GotoBLAS2 library. Therefore sparseness contributes to performance enhancements with respect to both accuracy and resource demands.

10 Discussion and Conclusion

Mammalian neuronal systems do not possess *any* structure. Rather, they have developed to use few resources in fulfilling their function. Neuroscientists discovered that both the connectivity and the activity are sparse in these biological information processing systems. Inspired by these findings, this work investigated how Artificial Neural Networks could profit from a sparse structure.

The formal prerequisite for such an analysis is the mathematical concept of a sparseness measure as discussed in Chapter 2. Hoyer's σ is very appealing for this task, since it is smooth and scale-invariant to name its most important attributes. In contrast to the L_0 pseudo-norm, σ is robust to noise because of its well-behaving analytical properties. It is preferable to employ explicit sparseness constraints, as it is then guaranteed that user-defined sparseness degrees are actually attained.

A fundamental problem arising from this approach is the algorithmic computation of Euclidean projections onto sets on which the chosen sparseness measure attains the target sparseness degree. There is a rich and mature theory on projections onto convex sets, but since Hoyer's σ induces non-convex target sets, this theory is not applicable here. Although Hoyer has proposed an algorithm to compute such sparseness projections, it has only been proved correct in a special case, lacking a satisfactory proof for the general case.

In Chapter 4 in this work, improved projection algorithms were proposed and rigorously proven to always compute the correct solution in finite time. It was further demonstrated that the proposed algorithms are more efficient than Hoyer's original algorithm, as the solution is found in both fewer iterations and less computation time when a concrete computing machine is used. Another result achieved in this work is the fact that the sparseness projection can be cast almost everywhere as vector-valued function, which is differentiable almost everywhere. Algorithms which exploit the special structure of the derivative to increase efficiency have been proposed for the computation of the projection's gradient. These achievements constitute the crucial mathematical tools for a comprehensive theory of Sparse Neural Networks.

A review of existing models incorporating sparseness has lead to the conclusion that none provide both sparse connectivity and sparse activity and are additionally suited for classification tasks subject to hard timing constraints, see Chapter 2. Three new models suitable for sparse classification were proposed in this work, namely Sparse Coding for Fast Classification (SCFC, Chapter 5), sparsely connected MLPs (SMLPs, Chapter 6) and Supervised Online

Auto-Encoder (SOAE, Chapter 7). These models significantly extend the use case of existing models. In doing so, the two fundamental problems of inefficient inference and poor discriminatory capabilities of sparse internal representations were solved. Inefficient inference is the main cause of long run-times until a classification decision can be provided, since in this case a non-trivial optimization problem has to be solved for each sample. Ignoring teacher signals associated with the samples during learning merely leads to information-preserving representations which are difficult to classify, such that sparse representations provide no benefit over the raw data in classification scenarios.

Inefficient inference was addressed in Chapter 5 with the SCFC model by implementing an inference path, a neuronal layer with local transfer function, to provide an easy-to-compute approximation to the latent sparse internal representation. However, having to store a latent representation for each learning sample is very memory-consuming. Moreover, the internal representation was enforced to be discriminative by introducing a classification path to SCFC, thus forming a universal two-layer MLP in conjunction with the inference path. Deviations between the classification decisions of the neural network and the teacher signals could thus be backpropagated to tune the network parameters, including the hidden layer that directly operates on the raw input data. This approach induced a completely new filter morphology, making the filters resemble local contrast fields. Further, the generalization performance was significantly improved compared to when teacher signals were ignored.

To overcome the limitation of SCFC of only being able to process small learning sets due to memory constraints, it was furthermore proposed in Chapter 6 to use it merely for pre-training, followed by a fine-tuning and post-processing stage for SMLPs. This procedure featured no sparse activity, but sparse connectivity was incorporated due to the application of sparseness projections to the degrees of freedom after each learning epoch. The proposed learning algorithm could be applied to very large learning sets, which proved beneficial in terms of generalization capabilities. Sparse activity was again incorporated in Chapter 7 with the SOAE model by implementation of sparseness projections as neural transfer function, thus making latent internal representations obsolete. This required the projection to be differentiable to facilitate gradient-based learning. This modification led to another boost in generalization performance, though computation of projections during run-time of the classifier slightly affects the timing performance. Hoyer's σ has turned out to be more effective than the simple L_0 pseudo-norm due to its smoothness when demanding sparse activity and sparse connectivity.

To evaluate the proposed models, statistical hypothesis testing methods verified that the positive effects of the sparseness concept are not random but rather statistically significant, see Chapter 8. Furthermore, the approaches developed in this work were compared with alternative approaches from the literature subject to the classification performance and the computational complexity of a classifier's recall phase on the MNIST database of handwritten digits which was introduced in Chapter 3. For this, a model of computation was proposed to facilitate a comparison abstracted from a real computing machine, and it was demonstrated through experiments that the model is able to predict relative speed-ups with appropriate accuracy.

The evaluation from Chapter 8 included Multi-Layer Perceptrons, Convolutional Neural Networks, Support Vector Machines, Polynomial Classifiers and Boosted Stumps. Although Support Vector Machines achieved quite few misclassifications, their computational complexity was tremendously high due to the large number of required support vectors. The converse holds for Polynomial Classifiers. Here, the recall phase needed only few resources, but the classification error was rather high. This type of classifier is very limited in the number of features that can be processed, since the memory requirements for training increase with a polynomial degree twice that of the polynomial used for classification. Boosting algorithms applied to raw pixel values did not achieve satisfactory generalization performance, while boosting products of stumps that use the responses of Haar-like filters are quite competitive. The latter require only few resources for a medium classification error and the resource demands are intermediate for small numbers of misclassifications. Therefore, the ideal application for them is in rejection cascades rather than when individual samples should be processed.

Convolutional Neural Networks were inherently designed for image processing and exhibit an improved generalization performance compared to ordinary Multi-Layer Perceptrons which treat the input data as permutation-invariant. By augmenting the learning set with virtual training samples, pixel topology can nevertheless be incorporated in MLPs, greatly improving classification accuracy. By using very large and deep Neural Networks, a very low number of misclassifications can be achieved. The computational complexity, however, is tremendous. It is questionable whether such an approach has practical use when the computational resources are limited. It was demonstrated that using quite small two-layer MLPs a good compromise between classification accuracy and cost in terms of computational complexity can be obtained. Moreover, sparseness can be exploited to further reduce computational complexity by approximately one order of magnitude, while in most cases even the generalization performance was improved. Sparse Neural Networks are hence appealing for real-world applications implemented on embedded devices since they maximize the performance while the resource demands are minimized.

This claim was validated in Chapter 9 on the real-world task of night-time pedestrian detection from a moving vehicle, where cameras sensitive in the near-infrared and far-infrared spectrums continuously record the scenery in front of the vehicle. Image processing techniques should detect vulnerable traffic participants in the camera images so that the driver can be warned long before situations become critical. Real-time constraints are even more important when intelligent headlight systems for directing light beams are used as a sophisticated alternative to simple displays. In this work, a heterogeneous rejection cascade approach was proposed which combined the classical technique of boosted Haar-like filters with sparse MLPs applied to highly discriminative feature values. Even in challenging urban scenarios with suboptimal conditions for the employed camera sensors, the proposed approach turned out to achieve adequate detection accuracy at a low number of false positives. It was demonstrated that when sparseness is exploited in the application of the rejection cascade, the run-time can be reduced by 75% relative to a canonical implementation of the matrix-vector product. Relative to routines that were extensively tuned and optimized for the employed processor, the entire detection system was sped up by 30%. The implementation of the sparse matrix-vector product used here was not optimized for the processor architecture. Even more dramatic performance boosts can be expected for embedded systems with small memory bandwidth and when the sparse matrix-vector multiplication algorithm is tuned for the target architecture.

Sparseness is hence an efficiency concept not only in biological neural networks but also in artificial information processing systems. Its full potential is realized with a smooth sparseness measure. No adverse effects have to be tolerated due to the constrained structure. On the contrary, resources are retained while the classification accuracy actually improves.

A List of Symbols

$\mathbb{N}$, $\mathbb{R}$, $\mathbb{R}_{\geq 0}$	natural numbers including zero, real numbers, non-negative real numbers
$\mathbb{R}^n$	n-dimensional Euclidean space with canonical basis $e_1, \ldots, e_n \in \mathbb{R}^n$
$\mathbb{R}^{d \times n}$	matrices with real entries with d rows and n columns
x_i	entry i of vector x, that is $x_i = e_i^T x$
$A_{i,j}$	entry in row i and column j of matrix A, that is $A_{i,j} = e_i^T A e_j$
E_n	identity matrix of size $n \times n$
$J_{d \times n}$	matrix of size $d \times n$, where each entry is equal to one
$\operatorname{diag}(\cdot)$	diagonal matrix with specified vector on its main diagonal
$\operatorname{vec}(\cdot)$	vectorization operator that stacks the columns of a matrix on top of another
$\langle \cdot, \cdot \rangle$	dot product of two vectors: $\langle \cdot, \cdot \rangle : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, $(x, y) \mapsto x^T y = \sum_{i=1}^n x_i y_i$
$\lVert \cdot \rVert_0$	L_0 pseudo-norm of a vector: $\lVert \cdot \rVert_0 : \mathbb{R}^n \to \mathbb{R}$, $x \mapsto \lvert \{ i \in \{1, \ldots, n\} \mid x_i \neq 0 \} \rvert$
$\lVert \cdot \rVert_1$	L_1 norm of a vector: $\lVert \cdot \rVert_1 : \mathbb{R}^n \to \mathbb{R}$, $x \mapsto \sum_{i=1}^n \lvert x_i \rvert$
$\lVert \cdot \rVert_2$	L_2 norm of a vector: $\lVert \cdot \rVert_2 : \mathbb{R}^n \to \mathbb{R}$, $x \mapsto \sqrt{x^T x} = \sqrt{\sum_{i=1}^n x_i^2}$
$\lVert \cdot \rVert_\infty$	L_∞ norm of a vector: $\lVert \cdot \rVert_\infty : \mathbb{R}^n \to \mathbb{R}$, $x \mapsto \max\{\lvert x_1 \rvert, \ldots, \lvert x_n \rvert\}$
$\lVert \cdot \rVert_F$	Frobenius norm of a matrix: $\lVert \cdot \rVert_F : \mathbb{R}^{d \times n} \to \mathbb{R}$, $X \mapsto \lVert \operatorname{vec}(X) \rVert_2$
$A \circ B$	Hadamard product (element-wise product) of two matrices of equal size
$A \circledcirc B$	element-wise quotient of two matrices of equal size
$A \otimes B$	Kronecker product of two matrices
sgn	sign function, $\operatorname{sgn} \colon \mathbb{R} \to \{0, \pm 1\}$, $\operatorname{sgn}(x) := x/\lvert x \rvert$ for $x \neq 0$ and $\operatorname{sgn}(0) := 0$

$\wp(M)$	power set of M
M^C	complement of set M
S_n	symmetric group of $\{1,\dots,n\}$
$X \in_R \mathcal{N}^{a\times b}$	entries of $X \in \mathbb{R}^{a\times b}$ are sampled from a standard normal distribution
$\mathbb{E}(\cdot)$	empirical mean value of independently drawn observations
$\mathrm{Var}(\cdot)$	empirical variance of independently drawn observations
id_M	identity function on a set M, that is $\mathrm{id}_M\colon M \to M, x \mapsto x$

B Matrix Operations

Matrix notation greatly simplifies many aspects in the study of complex dynamical systems, especially in the domain of neural networks. Wherever possible, gradient information is used to develop efficient algorithms for optimization of objective functions, see Appendix C. Using matrix calculus and related algebraic transformations, matrix notation can be preserved for derivatives, making relationships between individual quantities explicit [146]. This appendix gathers algebraic properties and fundamental rules that are useful for the matrix derivatives deduced in this work.

B.1 Basic Operators

Using the canonical basis $e_1, \ldots, e_n$ of the Euclidean space $\mathbb{R}^n$, access to rows, columns and entries of matrices can be expressed in terms of matrix-vector multiplications. In the remainder of this appendix, the dimensionality of basis vectors e_i is given implicitly from the context. Let $A \in \mathbb{R}^{a \times b}$ be a matrix with a rows and b columns. Then $e_i^T A \in \mathbb{R}^{1 \times b}$ denotes the ith row of A, $Ae_j \in \mathbb{R}^b$ denotes the jth column of A, and $e_i^T A e_j \in \mathbb{R}$ denotes the entry in the ith row and the jth column of A. Consider the following definition [147] of a linear operator that maps a matrix to a vector:

Definition 52. Let $A = (Ae_1, \ldots, Ae_b) \in \mathbb{R}^{a \times b}$ be a matrix with columns $Ae_1, \ldots, Ae_b$. Then

$$\operatorname{vec}(A) := \begin{pmatrix} Ae_1 \\ \vdots \\ Ae_b \end{pmatrix} \in \mathbb{R}^{ab}$$

is called the *vectorization* of A and results from stacking the columns of A one on another.

Clearly, every vector is a fixed point of the vectorization operator. The result of this operator on a matrix and its transpose yields the same result up to permutation of the entries [146]:

Proposition 53. Let $a, b \in \mathbb{N}$. Then there is exactly one permutation $\tau \in S_{ab}$ with $\operatorname{vec}(X) = P_\tau \operatorname{vec}\left(X^T\right)$ for all $X \in \mathbb{R}^{a \times b}$. Here, P_τ is the permutation matrix associated with τ. With the same permutation, $\operatorname{vec}\left(X^T\right) = P_\tau^T \operatorname{vec}(X)$ holds because of $P_\tau^{-1} = P_{\tau^{-1}} = P_\tau^T$.

If two matrices correspond to linear functions over a vector space, then their Kronecker product, which is also a matrix, corresponds to the tensor product of the two linear functions [148]:

Definition 54. Let $A \in \mathbb{R}^{a \times b}$ and $B \in \mathbb{R}^{c \times d}$ be two matrices of arbitrary size. Then

$$A \otimes B := \begin{pmatrix} (e_1^T A e_1)B & \cdots & (e_1^T A e_b)B \\ \vdots & \ddots & \vdots \\ (e_a^T A e_1)B & \cdots & (e_a^T A e_b)B \end{pmatrix} \in \mathbb{R}^{ac \times bd}$$

is called the *Kronecker product* of A and B and results from scalar multiplication of B with entries from A and arranging the result in an appropriate manner.

The special structure of the Kronecker product allows handling certain tensors with order greater than two solely through matrix multiplication. The usual approach is to conduct computations with Kronecker products implicitly and, whenever possible, using algebraic transformations to dissolve them. This dissolving can be achieved using the next result, which can also be used to isolate a quantity from the middle of a matrix product chain [147]:

Theorem 55. Let $A \in \mathbb{R}^{a \times b}$, $X \in \mathbb{R}^{b \times c}$ and $B \in \mathbb{R}^{c \times d}$. Then $\operatorname{vec}(AXB) = \left(B^T \otimes A\right) \operatorname{vec}(X)$.

The following lemma gathers some calculation rules of the Kronecker product [149]:

Lemma 56. Let A, B, C and D be matrices and $\alpha, \beta \in \mathbb{R}$. The dimensionality and existence of the inverse may be chosen such that the following statements are true:

(a) $A \otimes B \neq B \otimes A$ in general.

(b) $(\alpha A) \otimes (\beta B) = \alpha\beta(A \otimes B)$.

(c) $A \otimes (B + C) = A \otimes B + A \otimes C$.

(d) $(A + B) \otimes C = A \otimes C + B \otimes C$.

(e) $A \otimes (B \otimes C) = (A \otimes B) \otimes C$.

(f) $(A \otimes B) \cdot (C \otimes D) = AC \otimes BD$.

(g) $(A \otimes B)^T = A^T \otimes B^T$.

(h) $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$.

The matrix of ones can be used for repetitions and summations:

Definition 57. The matrix $J_{a \times b} \in \mathbb{R}^{a \times b}$ where $e_i^T J_{a \times b} e_j = 1$ for all indices $i \in \{1, \ldots, a\}$ and $j \in \{1, \ldots, b\}$ is called *matrix of ones*.

Lemma 58. The matrix of ones fulfills the following properties for $x \in \mathbb{R}^a$ and $A \in \mathbb{R}^{a \times b}$:

(a) Repetition of a vector: $x \cdot J_{1 \times b} = (x, \ldots, x) \in \mathbb{R}^{a \times b}$.

(b) Sum of entries: $J_{1 \times a} \cdot x = \sum_{i=1}^{a} x_i$.

(c) Sum of columns: $A \cdot J_{b \times 1} = \sum_{j=1}^{b} A e_j \in \mathbb{R}^a$.

(d) Inversion of vector diagonalization: $\operatorname{diag}(x) \cdot J_{a\times 1} = x$.

(e) Kronecker product is index multiplication for matrices of ones: $J_{a\times b} \otimes J_{c\times d} = J_{ac\times bd}$.

Another frequently encountered operator is the entry-wise multiplication of matrices [149]:

Definition 59. Let $A \in \mathbb{R}^{a\times b}$ and $B \in \mathbb{R}^{a\times b}$ be two matrices of equal size. Then the matrix

$$A \circ B \in \mathbb{R}^{a\times b} \text{ with } e_i^T(A\circ B)e_j = (e_i^T A e_j)(e_i^T B e_j) \text{ for all } i \in \{1,\dots,a\} \text{ and all } i \in \{1,\dots,b\}$$

is called the *Hadamard product* of A and B.

The Hadamard product fulfills the following properties, which directly follow from its definition [149]. Its relation to the product with diagonal matrices is of special importance:

Lemma 60. Let $A, B \in \mathbb{R}^{a\times b}$, $x, y \in \mathbb{R}^n$ and $\alpha, \beta \in \mathbb{R}$. Then:

(a) $A \circ B = B \circ A$.

(b) $A \circ (B + C) = A \circ B + A \circ C$.

(c) $(\alpha A) \circ (\beta B) = \alpha\beta(A \circ B)$.

(d) $(A \circ B)^T = A^T \circ B^T$.

(e) $\operatorname{vec}(A \circ B) = \operatorname{vec}(A) \circ \operatorname{vec}(B)$.

(f) $\operatorname{diag}(a) \cdot b = a \circ b = \operatorname{diag}(b) \cdot a$.

(g) $\operatorname{diag}(\operatorname{vec}(A)) \cdot \operatorname{vec}(B) = \operatorname{vec}(A \circ B) = \operatorname{diag}(\operatorname{vec}(B)) \cdot \operatorname{vec}(A)$.

B.2 Matrix Calculus

This section describes the fundamental rules that can be applied when computing the derivative of a function that depends on a vector or a matrix. Considering their dimensionality, the gradients used in this work conform to the Jacobian definition, which is the transpose of the Hessian definition:

Definition 61. Let $f\colon \mathbb{R}^a \to \mathbb{R}^b$, $x \mapsto f(x)$, and $g\colon \mathbb{R}^{a\times b} \to \mathbb{R}$, $X \mapsto g(X)$, be differentiable functions. Then $\partial f/\partial x \in \mathbb{R}^{b\times a}$ and $\partial g/\partial X \in \mathbb{R}^{b\times a}$.

For vector-valued functions the prime notation is used occasionally to denote the function that emerges from mapping points to their gradients: $f'\colon \mathbb{R}^a \to \mathbb{R}^{b\times a}$, $x^* \mapsto \partial f/\partial x|_{x^*}$.

The basic principles of scalar calculus apply to matrix calculus as well. The following is a collection of important results from [150, 151], which have been slightly adjusted to suit the situation in this work:

Theorem 62 (Chain Rule). Let $f\colon \mathbb{R}^a \to \mathbb{R}^b$ and $g\colon \mathbb{R}^b \to \mathbb{R}^c$ be differentiable functions and $h := g \circ f\colon \mathbb{R}^a \to \mathbb{R}^c, x \mapsto g(f(x))$ their composition. Then h is differentiable with gradient

$$\frac{\partial h(x)}{\partial x} = \frac{\partial g(f(x))}{\partial f(x)} \cdot \frac{\partial f(x)}{\partial x} \in \mathbb{R}^{c\times a}.$$

Theorem 63 (Product Rule). Let $f\colon \mathbb{R}^a \to \mathbb{R}^b$ and $g\colon \mathbb{R}^a \to \mathbb{R}^b$ be differentiable functions and $h := \langle f,\, g\rangle : \mathbb{R}^a \to \mathbb{R},\, x \mapsto \langle f(x),\, g(x)\rangle$ their dot product. Then h is differentiable with gradient

$$h'(x) = g(x)^T f'(x)^T + f(x)^T g'(x)^T \in \mathbb{R}^{1\times a}.$$

Using these results, the product rule for the Hadamard product can be deduced [149]:

Theorem 64 (Hadamard Product Rule). Let $f\colon \mathbb{R}^a \to \mathbb{R}^b$ and $g\colon \mathbb{R}^a \to \mathbb{R}^b$ be differentiable functions and $h := f \circ g\colon \mathbb{R}^a \to \mathbb{R}^b, x \mapsto f(x) \circ g(x)$ their Hadamard product. Then h is differentiable with gradient

$$\frac{\partial h(x)}{\partial x} = \operatorname{diag}(f(x)) \cdot \frac{\partial g(x)}{\partial x} + \operatorname{diag}(g(x)) \cdot \frac{\partial f(x)}{\partial x} \in \mathbb{R}^{1\times a}.$$

Quotients of scalar-valued functions that depend on a matrix, which occur often in normalization processes, are given analogously to the scalar quotient rule [151]:

Theorem 65 (Quotient Rule). Let $f\colon \mathbb{R}^{a\times b} \to \mathbb{R}$ and $g\colon \mathbb{R}^{a\times b} \to \mathbb{R}$ be differentiable functions and $h\colon \mathbb{R}^{a\times b} \setminus \{A \in \mathbb{R}^{a\times b} \mid g(A) = 0\} \to \mathbb{R}, A \mapsto {}^{f(A)}\!/\!_{g(A)}$ their quotient in $\mathbb{R}$. Then h is differentiable with gradient

$$\frac{\partial h(A)}{\partial A} = \frac{g(A)f'(A) - f(A)g'(A)}{g(A)^2} \in \mathbb{R}^{b\times a}.$$

The next lemma contains statements, some of which were inspired by [150] and [149]. It gathers the gradients of various elementary functions.

Lemma 66. Let $x, a \in \mathbb{R}^d$ and $A \in \mathbb{R}^{d\times d}$. Then the following holds:

(a) $\partial/\partial x\, \|x\|_2 = x^T/\|x\|_2 \in \mathbb{R}^{1\times d}$.

(b) $\partial/\partial x\, \|x\|_2^2 = 2x^T \in \mathbb{R}^{1\times d}$.

(c) $\partial/\partial x (a^T x a) = a a^T \in \mathbb{R}^{d\times d}$.

(d) $\partial/\partial x \left(x^T A x\right) = x^T (A + A^T) \in \mathbb{R}^{1\times d}$.

(e) $\partial/\partial x\, (a \circ x) = \operatorname{diag}(a) \in \mathbb{R}^{d\times d}$.

(f) $\partial/\partial x\, (x \circ x) = 2\operatorname{diag}(x) \in \mathbb{R}^{d\times d}$.

(g) $\partial/\partial x \left(x x^T a\right) = a x^T + \left(x^T a\right) E_d \in \mathbb{R}^{d\times d}$.

(h) $\partial/\partial x \left(x x^T x\right) = 2 x x^T + \left(x^T x\right) E_d \in \mathbb{R}^{d\times d}$.

The following result establishes the relationship between derivatives with respect to a vectorized matrix and derivatives with respect to the original matrix [147]:

Theorem 67. Let $f\colon \mathbb{R}^{a\times b} \to \mathbb{R}$ be a *vectorizable function*, which means that is there is a function $f_{\text{vec}}\colon \mathbb{R}^{ab} \to \mathbb{R}$ such that $f = f_{\text{vec}} \circ \text{vec}$. Then the following holds:

(a) If there is a $G \in \mathbb{R}^{a\times b}$ with $\frac{\partial f_{\text{vec}}(\text{vec}(X))}{\partial\, \text{vec}(X)} = \text{vec}(G)^T$, then $\frac{\partial f(X)}{\partial X} = G^T \in \mathbb{R}^{b\times a}$.

(b) If there is a $G \in \mathbb{R}^{b\times a}$ with $\frac{\partial f_{\text{vec}}(\text{vec}(X))}{\partial\, \text{vec}(X^T)} = \text{vec}(G)^T$, then $\frac{\partial f(X)}{\partial X} = G \in \mathbb{R}^{b\times a}$.

Refer to the proofs of Theorem 46 and Theorem 50 for examples on how the results of this appendix can be used to deduce gradients of functions that map matrices to real numbers without having to use tensors with an order greater than two.

C Optimization of Neural Networks

This appendix discusses various optimization schemes commonly used for neural networks, especially for those discussed in this work. First, the general setup of an objective function is described followed by a discussion of when performing updates to the degrees of freedom is most beneficial. Next, gradient descent methods and related techniques for numerical function minimization are reviewed based on their practicability for large scale problems. The appendix concludes with a short introduction to initialization schemes and a collection of various transfer functions and similarity measures.

C.1 Learning Set and Objective Function

The process of adapting a neural network to a specific task involves a *learning set*. This is a data set where examples are given from which the neural network should learn how to react when a certain input is given. In this work, the term samples is sometimes used instead of examples. The *evaluation set*, on the other hand, is a disjoint data set that is used to assess the generalization capabilities of the network. In other words, the evaluation set is used to verify whether the network was able to capture the structure in the data and therefore has the ability to produce correct answers when an example is given that the network has never seen before [53].

Depending on the task, learning algorithms can be classified as being either *unsupervised* or *supervised* [53]. In an unsupervised setting, the aim is to yield a network that is able to explain the distribution of the samples from the learning set. Examples for unsupervised learning are generative models that are able to reproduce the learning set, or clustering algorithms that compute certain representatives that should be similar to certain subsets of the examples. Supervised learning algorithms, on the contrary, specify exactly which output the network should produce for each sample in the learning set. In doing so, a *teacher signal* or a *target vector* is associated with each sample in the learning set. This is especially the case for classification tasks, where for each example the category it belongs to should be recognized. This category is also called *class membership*.

The optimization schemes discussed in this appendix apply to both unsupervised and supervised learning. For an abstract description, let the learning set consist of M datums of dimensionality $D \in \mathbb{N}$, that is $\mathcal{X}^{(1)}, \dots, \mathcal{X}^{(M)} \in \mathbb{R}^D$. In an unsupervised setting, a datum $\mathcal{X}^{(\mu)}$ consists merely of a single data point of dimensionality $d \in \mathbb{N}$, that is $\mathcal{X}^{(\mu)} = x^{(\mu)} \in \mathbb{R}^d$ and $D = d$ to match the notation in this work. In a supervised setting, a teacher signal of dimensionality $c \in \mathbb{N}$ is additionally associated with each sample. Hence a single datum becomes a pair of an example and a teacher signal, $\mathcal{X}^{(\mu)} \cong \left(x^{(\mu)}, t^{(\mu)}\right) \in \mathbb{R}^d \times \mathbb{R}^c$ and $D = d + c$.

Once task and architecture of a neural network have been defined, a scalar-valued *objective function F* can be inferred. For the networks discussed in this work, the objective function is typically a single similarity measure or a finite sum thereof. This can be, for example, the similarity of a reproduced sample with the original sample in the unsupervised case, or the classification error in the supervised case. The objective function F depends on a single datum $\mathcal{X}^{(\mu)}$ and the *network parameters*, which are collectively denoted by $\mathcal{W}$ in this appendix. For notational convenience, it is assumed that the parameters possess no special structure and can be written in a vector $\mathcal{W} \in \mathbb{R}^L$, where $L \in \mathbb{N}$ denotes the number of the degrees of freedom.

Using the definitions from above, the objective function for a single datum is of the form $F\colon \mathbb{R}^D \times \mathbb{R}^L \to \mathbb{R}$. The objective function for the entire data set is then defined as the empirical mean value of the objective function for single data points:

$$E\colon \mathbb{R}^L \to \mathbb{R}, \qquad \mathcal{W} \mapsto \frac{1}{M}\sum_{\mu=1}^{M} F\left(\mathcal{X}^{(\mu)}, \mathcal{W}\right) \overset{!}{\to} \operatorname{opt}_{\mathcal{W}}.$$

This function E should be optimized by tuning of the network parameters to adapt the neural network to the learning set. As closed-form solutions are not available for most of the more complex types of neural networks, numerical optimization strategies have to be applied. Without loss of generality, it is assumed in the remainder of this appendix that E, and therefore F, shall be minimized. For this, it is further assumed that F is partially differentiable with respect to $\mathcal{W}$, such that gradient information can be exploited.

There are also *block coordinate methods*, where only a subset of $\mathcal{W}$ is updated at a time while the remaining parameters are kept fixed. An example for such an approach is the optimization procedure for Non-negative Matrix Factorization with Sparseness Constraints as described in Section 5.1. There, the matrix of bases W and the latent matrix of code words H are updated alternatingly in an Expectation-Maximization-like fashion [152]. Note that all parameters are determined by a pair $\mathcal{W} = (W, H)$. The alternating method is motivated by the convexity of the objective function in either of the two arguments. It is not convex in both simultaneously, so there is no guarantee that a global optimum will be found.

C.2 Online Learning vs. Batch Learning

Numerical minimization [103] of a function yields a sequence of parameters $\mathcal{W}(t)$ for $t \in \mathbb{N}$, such that $E(\mathcal{W}(t))$ converges to a minimum for $t \to \infty$. $\mathcal{W}(0)$ is called the *starting value*, and may be picked at random or using more sophisticated techniques as discussed in Section C.4.

For $t > 0$, $\mathcal{W}(t)$ is computed from $\mathcal{W}(t-1)$ in an *update step*, that depends on the concrete technique employed. For an abstract description, let the update step be dependent on so-called *update information*.

As the objective function gradually decreases during optimization, the network improves its performance on the samples of the learning set. Therefore, this process is also called *learning*. There are two possible instants of time when an update step can be carried out [53]. The *online learning* protocol specifies that an update is performed once an individual sample has been presented to the network. *Batch learning*, on the contrary, denotes the process of first presenting the entire learning set, or at least a significant portion of it. For each sample, the update information is recorded and accumulated. Finally, one single update step takes place based on the accumulated information. The time span during which the learning set, or a representative subset thereof, is used to adapt the parameters, is called an *epoch* in both protocols.

While batch learning better reflects the idea of minimizing the mean value of the objective function on the entire learning set, there are reasons why online learning may be preferred in certain scenarios [153]. First, the update rate is much higher using online learning. Thus, when there is redundancy among the samples from the learning set, that is some examples look alike, then more updates are carried out on these similar samples. With batch learning, this information would be canceled out in the accumulation of the update information of the entire learning set [54]. In this case an optimum is achieved more quickly using online training. In fact, there is experimental evidence that online learning converges about one order of magnitude faster than batch learning on large, redundant learning sets [154, 153].

A theoretical argument for the superiority of online learning was given by [155], where the situation when learning sets with a large number of samples are available was studied. There, the notion of *optimization speed* versus *learning speed* was introduced. The first term describes how fast an optimum of the objective function on the learning set is achieved, and the second term describes the speed of convergence on the entire distribution where the samples of the learning set were drawn from. As the latter term precisely describes the generalization performance of the neural network, its optimization is preferred to the sole performance on the learning set. In [155], it was shown both theoretically and empirically that the best generalization error is achieved asymptotically through the optimization technique that uses as many training samples as possible. Hence online learning improves on the generalization error compared to batch learning when more samples are available.

However, an advantage of batch learning over online learning is that parallelization can be exploited easily [154]. As the network changes with every sample presentation when online learning is employed, each sample presentation is dependent on a different instance of network parameters. Therefore, only serial processing can be used to account for data dependence. With batch learning, the network stays the same throughout an entire epoch. Hence, the learning set can be divided into multiple bins, and the update information can be computed on each bin in parallel. Once all bins have been processed, the update information can be reduced into a single update step which is the only computation that has to be synchronized. As this operation is very brief compared to sample presentation, parallelism causes speed-ups that are nearly linear to the amount of processing units. Note that there is still an upper bound on possible speed-ups as explained by Amdahl's law [96] because the synchronization overhead cannot be neglected entirely.

An approach that is a trade-off between online and batch learning is the usage of *mini-batches*, see for example [156]. The learning set is divided into subsets with cardinality typically in the tenths or hundredths. Optimization is then parallelized among these mini-batches, and updates take place once a mini-batch has been processed. The drawback is that a large number of updates have to be synchronized. Especially when the number of the network parameters is high, update steps take even longer, dramatically increasing the fraction of operations that have to be performed serially, in turn decreasing parallel efficiency [96]. A heuristic for alleviating this effect was proposed by [156]: By delaying updates when write access to the network parameters cannot be gained, parallel efficiency can be increased. However, this efficiency is bought dearly as this technique degenerates into batch learning under heavy load, with all the drawbacks discussed above.

Though batch and mini-batch approaches can be parallelized, the update rate is always lower rather than when online training is used. As already mentioned, the update rate is the most crucial part for gaining high learning speed. In fact, when the learning set size is large, batch learning can be significantly slower than online learning, even when using 300 machines in parallel [153]. In conclusion, online learning should be preferred over batch methods whenever possible.

C.3 Gradient Descent for Function Minimization

This section describes several methods for numerical function minimization using gradient information [103]. To this end, let $E\colon \mathbb{R}^L \to \mathbb{R}$ be the objective function that is to be minimized. In case of online learning, E becomes the objective function for an individual datum, previously denoted by F, where the datum is kept constant. The objective function is assumed to be differentiable, such that $E'\colon \mathbb{R}^L \to \mathbb{R}^L$ becomes the function associated with the transpose of the gradient at a position $\mathcal{W}$.

The simplest gradient method for minimization is based on the trait of the directional derivative to always be inferior to the gradient [157]. Hence when a step is taken in the opposite direction of the gradient, the function value decreases, and repetition of this process leads to a local minimum. The *step size* for updating the current parameters can always be selected so that E decreases if the gradient does not vanish [157]. The *gradient descent algorithm* can be expressed by the iteration

$$\mathcal{W}(t+1) := \mathcal{W}(t) - \eta(t) \cdot E'(\mathcal{W}(t)), \text{ where } \eta(t) > 0 \text{ is called the } \textit{step size}.$$

This update rule is particularly simple when the gradient of E can be computed efficiently in closed form. However, the main disadvantage is a poor convergence rate, that is a large number of iterations have to be carried out until a minimizer is found. Nevertheless, gradient descent is the preferred method when the number of the degrees of freedom L is large. This is because updates require an effort linear in L once the gradient is known. More sophisticated algorithms, like Newton and quasi-Newton methods or the Levenberg-Marquardt algorithm require an operations and storage space effort that is quadratic in L [158, 53] because they need to compute the Hessian of the objective function and can thus be considered second order optimization methods. Therefore, these methods can only be applied in scenarios where the parameter dimensionality L is low.

Consider for example a two-layer neural network with 1000 hidden units that should be used to classify samples from the MNIST database of handwritten digits, which was discussed in Chapter 6. Then the number of parameters is about $L \approx 795\,000$, and there are about $\binom{L}{2} \approx 316\,000\,000\,000$ entries in the Hessian, where only the upper triangle was counted for symmetry reasons. If the Hessian was stored using double precision IEEE 754 floating point numbers with 8 byte per entry, then about 2.3 terabytes of storage capacity would be necessary. This is one to two orders of magnitude more than the main memory available in a current workstation computer. It is hence not realistic to directly apply second order methods for numerical minimization if L is large, unless computers with a very large main memory capacity are used. This, however, still does not solve the problem of the large computational resources needed to evaluate all 316 billion values of the Hessian. Another possibility would be to use special algorithms that exploit the structure of the Hessian of certain objective functions, such that the entire Hessian is only needed implicitly. An example for this was given by [159], where it is shown that the product of the Hessian with an arbitrary vector can be computed without explicitly computing the Hessian in certain neural network architectures. Approximations to the Hessian, such as computing solely its diagonal elements and similar heuristics are also possible and discussed at length in [160]. It should however be noted that these approximations are rather crude and their use in practical applications is limited.

C.3.1 Projection Methods for Constrained Optimization Problems

Another notion is optimization subject to additional constraints, for example the explicit sparseness constraints as described in Chapter 2. The objective is then to find parameters that lie in a set $\varnothing \neq P \subseteq \mathbb{R}^L$ of feasible parameters, that is to find $\arg\min_{\mathcal{W} \in P} E(\mathcal{W})$. Supposing there is a projection operator $\Pi\colon \mathbb{R}^L \to P$ that finds the closest vector in P for a given parameter vector, the *projected gradient descent algorithm* consists of carrying out the update rule of common gradient descent, followed by application of Π to ensure the new parameters are feasible [103]:

$$\mathcal{W}(t+1) := \Pi\left[\mathcal{W}(t) - \eta(t) \cdot E'(\mathcal{W}(t))\right].$$

The projection then ensures membership of $\mathcal{W}$ in P as loop-invariant. The efficiency of the method strongly depends on the efficiency of the computation of the projection Π. Examples for Π are the sparseness-enforcing projection operator and the projection onto an L_0 pseudo-norm constraint as discussed in Chapter 4. There are theoretical results on the convergence of projected gradient methods when P is convex [103]. Although the feasible set with respect to the sparseness-enforcing projection operator or the L_0 constraint is not convex, the method still works well in practice [26, 79, 101, 66, 69].

C.3.2 Step Size Tuning

Although existence of a step size η that decreases the value of the objective function is guaranteed when the gradient does not vanish, choosing an optimal step size remains an open issue. If η is chosen too small, convergence to a minimum will take many updates and thus involves long optimization runs. If η is chosen too large, the values of the objective function will oscillate, finally leading to divergence. In the remainder of this section, numerous heuristics for time-dependent step sizes $\eta(t)$ will be discussed.

Constant Step Size

The simplest approach [103] is determining an initial step size η_0 and keeping it constant throughout the entire optimization such that $\eta \equiv \eta_0$. It is crucial that η_0 is chosen carefully such that neither divergence nor overly slow convergence results. In practice, cross-validation on the learning set is used to determine a value for η_0 by testing the performance of a certain set of candidate step sizes. The cross-validation is repeated numerous times to account for randomness in parameter initialization and the sequence in which learning samples are used to adapt the parameters. The winning step size can then be found by choosing the one that performed best in the mean, see Figure 6.4 in Chapter 6 for an example.

Annealed Step Size

This heuristic involves finding an initial step size as described before. The step size is then reduced after every update so that it converges to zero, $\eta(t) \to 0$ for $t \to \infty$. This strategy induces convergence [103] when the sum of the individual step sizes does not converge, that is when $\sum_{t=0}^{\infty} \eta(t) = \infty$. This is because otherwise a point is reached from where no substantial progress can be made. A possible choice is setting $\eta(t) = \eta_0/_{t+1}$ for $t \in \mathbb{N}$ which fulfills the required properties. However, as the step size gets small very quickly, numerical inaccuracies and convergence to high-lying local minima render this method practically suboptimal [158].

A better choice is a multiplicative annealing [53], $\eta(t+1) := \alpha\eta(t)$ where $\alpha \in (0,\ 1)$, usually chosen as $\alpha = 0.99$ or $\alpha = 0.999$. As the geometric series converges, there is no theoretical guarantee that this heuristic leads to an optimum. Despite the lack of theoretical justification, it was shown that this method can achieve state-of-the-art results in practice [59, 60].

Eigenvalues of the Hessian

Another method was given in [161], assuming the objective function is local quadratic. The step size is estimated here by determining the largest eigenvalue of the Hessian, which is achieved by combining the power method with a numerical estimate for the Hessian. The largest eigenvalue is filtered using a running average method, and finally its reciprocal value is taken for the step size. The authors computed this optimal step size once at the very beginning of the optimization because they reported that changes to it are very subtle during later stages of optimization. In the experiments performed for this work it was found that this method works well in low complexity scenarios, when very small neural networks and learning sets were used. For reasonable sized networks, the method failed to compute an adequate step size. Re-application of the method during learning to find a more suited step size in later stages of optimization failed as well. Further, the method is very time-consuming, and it was found that the required time is better spent carrying out cross-validation to determine a more reliable estimate of an adequate initial step size.

The "Search Then Converge" Method

Step size schedules such as the "search then converge" heuristic [162, 158] provide different step sizes for different phases during optimization. At the beginning of the optimization, known as the search phase, a large step size is chosen to find a location in the vicinity of an optimum. As optimization proceeds, the step size is continuously transformed to match a reciprocal function of time, this is the converge phase. A simple [162] and a more complex [158] version of this heuristic exists:

$$\eta(t) := \frac{\eta_0}{1+\frac{t}{\tau}} \quad \text{versus} \quad \eta(t) := \eta_0 \frac{1+\frac{c}{\eta_0}\frac{t}{\tau}}{1+\frac{c}{\eta_0}\frac{t}{\tau}+\tau\frac{t^2}{\tau^2}}.$$

Here, η_0 is the initial learning rate, τ controls the length of the search phase – the higher τ the longer the search phase – and c is a gain factor for the initial learning rate. With these step size schedules many of the optimization algorithm parameters have to be determined before-hand. This requires a long cross-validation phase before the actual training procedure and may only be carried out if sufficient computational resources are available. The gain factor c can be

Algorithm C.1: The Bold Driver heuristic for projected gradient descent [163, 26].

Input: Objective function $E\colon P \to \mathbb{R}$ where $\varnothing \neq P \subseteq \mathbb{R}^{a\times b}$ is the set of feasible solutions, projection operator $\Pi\colon \mathbb{R}^{a\times b} \to P$, current location $\mathcal{W}^* \in P$, current step size $\eta > 0$.
Output: $\text{bolddriver}(E, \Pi, \mathcal{W}^*, \eta) = (\mathcal{W}^*_{\text{new}}, \eta_{\text{new}}) \in P \times \mathbb{R}_{>0}$ with $E(\mathcal{W}^*_{\text{new}}) \leq E(\mathcal{W}^*)$.
Algorithm parameters: $\eta_+, \eta_-, \eta_{\min} \in \mathbb{R}_{>0}$.

```
  // Evaluate initial function value and gradient
1 $v := E(\mathcal{W}^*) \in \mathbb{R};\ G := E'(\mathcal{W}^*) \in \mathbb{R}^{a\times b}$;
  // Tune step size to decrease $E$ unless minimum step size already reached
2 while $\eta > \eta_{\min}$ do
      // Perform one step of projected gradient descent
3     $\mathcal{W}^*_{\text{new}} := \Pi(\mathcal{W}^* - \eta G)$;
      // Increase step size and return if function value did not increase
4     if $E(\mathcal{W}^*_{\text{new}}) \leq v$ then return $(\mathcal{W}^*_{\text{new}}, \eta \cdot \eta_+)$;
      // Reduce step size and try again
5     $\eta := \eta \cdot \eta_-$;
6 end
  // Minimum step size was reached, return with vanishing step size
7 return $(\mathcal{W}^*, 0)$;
```

chosen optimally by computation of the smallest eigenvalue of the Hessian of the objective function. However, this is very time-consuming in practical applications as discussed above. Because of these drawbacks more conservative methods should be used in practice to obtain robust results.

The Bold Driver Method

The Bold Driver algorithm is an automatic step size adaptation mechanism aimed at finding parameters in each iteration so that the values of the objective function E are non-increasing. Its formulation is given in Algorithm C.1, which computes the function $\text{bolddriver}(E, \Pi, X^*, \eta)$. Here E is the objective function, Π is the projection operator onto the set of feasible solutions, X^* is the current location where E is evaluated, and η is the current step size. If no projection is to be employed, the identity function $\text{id}_{\mathbb{R}^{a\times b}}\colon \mathbb{R}^{a\times b} \to \mathbb{R}^{a\times b}$, $X \mapsto X$, is passed to the algorithm for uniform notation. The algorithm can be used iteratively, and also in an alternating fashion in more complex scenarios, see for example Chapter 5.

The mode of operation is given as follows: The algorithm first computes the initial value of the objective function. It then computes an updated position using the projected gradient descent update rule, and computes the resulting function value. If that function value is less than or equal to the initial value, the step size is increased by multiplication with η_+, and both the new position and the step size are returned for further processing. If the function value is greater than the original value, the step size is reduced by multiplication with η_-, and that step size is used for subsequent examination. To guarantee that the algorithm terminates and to avoid numerical errors, a minimal step size $\eta_{\min}$ is used which may not be underrun. If the minimal step size is reached, the original position and a vanishing step size is returned. Thus, any algorithm using this minimization heuristic can handle this case explicitly. The algorithm parameters are chosen as $\eta_+ := 1.2$, $\eta_- := 0.5$ and $\eta_{\min} := 10^{-10}$ for practical applications [26].

C.4 Initialization Schemes and Deep Learning

Every method aimed at numerical minimization of a function needs starting values, a point of origin $\mathcal{W}(0) \in \mathbb{R}^L$ from which successive points $\mathcal{W}(t)$ for $t \in \mathbb{N} \setminus \{0\}$ can be computed to improve the value of the objective function. The purpose of parameter initialization is to reduce the number of update steps needed until the minimization algorithm converges and to support the minimization algorithm in discovering the best-suited local minima. Hence in the context of neural networks the training process should be sped up and the generalization capabilities should be increased by proper initialization. This section discusses several initialization schemes commonly used for the training of neural networks, including material published in [164]. For the initialization schemes, it is necessary to know about the structure of the parameter vector $\mathcal{W}$, for example, which entries in this vector correspond to synaptic connections in a network's hidden layer or the corresponding threshold values. The original block structure of $\mathcal{W}$ is important in cases such as those where the hidden layers are initialized in a different way than the output layer.

The simple approach of setting $\mathcal{W}(0) := 0 \in \mathbb{R}^L$ is not advisable, as this will only cause symmetry in the neural network resulting in poor performance [89]. Instead, the network parameters should at least be initialized by sampling from a zero-mean random distribution [89, 53, 87]. The shape of the random distribution should be symmetric around the origin; uniform and normal distributions are common choices. The critical point is selecting the distribution's variance. This value of course should depend on the scaling of the input data and that of the transfer functions in the network [87]. With prior knowledge of the scaling and some heuristics, it is recommended to use the normal distribution with the standard deviation set to the reciprocal of the square root of the number of inputs each neuron receives [87]. Although this approach has been widely adopted in the literature, there is no theoretical justification why this specific choice should be preferred over other values for the standard deviation.

A less common but more efficient initialization scheme of neurons in the first hidden layer of a neural network is the initialization with prototypes. Here, a random subset of the training examples is selected and the degrees of freedom of the network's hidden layer are initialized by simply replicating and scaling these samples. Originally used for Radial Basis Function

Networks [53], the method can also be applied to conventional neural networks that employ the dot product to compute dendritic potentials rather than the Euclidean norm [165]. Similar to the usage of random numbers, the scaling of the input data and the domain of the used transfer functions have to be taken into consideration [165]. In order to prevent saturation of the hidden units, that is that the derivatives of the transfer functions are very small and hence the gradient of the hidden layer is close to zero, it is important that the magnitude of the neurons in the hidden layer is not chosen too large. Otherwise, learning will be extremely slow because of the small norm of the gradient. Despite the simplicity of this initialization scheme, it was found in recent empirical studies to provide starting points from which better classifiers can be gained using gradient descent rather than when random starting values were used [166, 164].

A trend that has emerged in the recent years is unsupervised pre-training for deep learning [125, 167, 168]. Deep learning means in this case that there are more hidden layers than just the one in a conventional two-layer MLP. Deep networks are especially useful if the mapping from input values to target output values is highly nonlinear. For example, some Boolean functions with d variables need a number of neuronal units exponential in d if represented with a two-layer network. They can also be expressed with a number of layers logarithmic in d where each layer has only a number of units linear in d, resulting in a number of units quasi-linear in d [167]. Therefore, although two-layer neural networks can be used to approximate any sufficiently smooth function with arbitrary precision by a potentially very large number of hidden units [90, 91, 92], using more layers may be more efficient. There is however no theoretical result to date that generalizes this argument from Boolean functions to more complicated real-valued functions.

Training of such deep neural networks from random starting points suffers especially from saturation, as the interactions between multitudes of layers are non-trivial to analyze in order to provide an adequate standard deviation for the random distribution [169]. Further, many layers contribute to ill-conditioned gradients with either an almost vanishing or an almost infinite magnitude [169]. This phenomenon has been known for over two decades in the context of recurrent neural networks [170], which can be cast as state-invariant neural networks with numerous layers by unfolding through time [169]. Moreover, gradient descent applied to an objective function measuring the classification error has the disadvantage of easily getting trapped in poor local minima, often ones that are very close to the starting values [167].

It was found recently that results can be improved significantly by first using unsupervised learning algorithms to initialize the network's layers one after another [125, 167]. In doing so, the first hidden layer is initialized using an auto-encoder so the original learning set can be reproduced. Then a second hidden layer is added to the network and trained so it can reproduce the output of the first hidden layer given the learning set. This process is repeated until the desired number of layers has been created. Eventually, all layers are trained simultaneously in a supervised manner such that the classification error is minimized. This approach has the advantage that when the supervised fine-tuning is about to commence, the degrees of freedom are already much closer to a good solution as when random numbers were used. Plain gradient descent is then an appropriate method for classification error minimization in this situation [125, 168]. It is of course also possible to use supervised learning algorithms instead of unsupervised pre-training when subsequently adding layers. This was however shown empirically to produce worse results [167], and it was conjectured that supervised learning causes information to be compressed into the target values too quickly, such that this information is not accessible anymore when training higher layers [167].

While unsupervised pre-training is fairly robust to its own random initialization, a large number of layers induce a multitude of poor local minima, easily trapping pure supervised optimization from random starting points using gradient descent [168]. This can be seen from the visualization of the trajectories gained by embedding the network parameters during training into two-dimensional space, which reveals that pre-training not only enables discovering very distinct minima in an entire new region of the network parameter space, but also that the spatial variance of the found minima in network parameter space is much smaller compared to when random starting points were used [168].

Although the use of unsupervised pre-training is controversial, as for example [59, 60] have achieved results that extended the state of the art using merely randomly initialized networks, it was demonstrated in this work by comparative experiments that unsupervised learning algorithms are especially beneficial in the context of Sparse Neural Networks, see Chapter 6 and Chapter 7. The arguments on why unsupervised pre-training works for deep learning can be transferred to Sparse Neural Networks. Both sparse connectivity and activity give rise to vanishing entries in the objective function's gradient. Hence, it would be impossible to learn a reasonable model from random starting points using plain gradient descent unless the parameters are already in the vicinity of a good minimum. Further, sparseness degrees have to be rather high to achieve a benefit, which greatly narrows down the set of possible solutions and in turn makes the optimization problem hard to solve. Discovering a region where superior minima reside by sophisticated initialization schemes therefore provides useful clues to numerical minimization algorithms. As pre-training comes at almost no extra training time cost, and does not degrade classification capabilities but instead consequently improves them, there is no reason why this technique should be avoided in real-world scenarios.

C.5 Transfer Functions

In neural networks and related models, a *transfer function* is a vector-valued function modeling a mapping from one state to another. The prime example is the mapping from dendritic potentials to axonal potentials in a neural network. As a composition of linear functions is linear itself, nonlinear transfer functions enrich the class of functions that can be approximated by an MLP. In other words, the MLP computes an embedding in a higher dimensional space through its hidden layers, where the samples are easier to classify by another layer [171]. The final layer then compresses the information from the internal representation into a vector with length equal to the number of classes in the classification problem such as to predict class membership of the input sample. The following definition describes a typical class of transfer functions encountered.

Definition 68. A transfer function $f\colon \mathbb{R}^n \to \mathbb{R}^n$ is called *local* if and only if there is a function $g\colon \mathbb{R} \to \mathbb{R}$ such that $f(x_1, \ldots, x_n) = (g(x_1), \ldots, g(x_n))^T$ for all $x = (x_1, \ldots, x_n)^T \in \mathbb{R}^n$. f is then called an *extension* of g.

Note that the gradient of a local function is always a diagonal matrix, that is in the situation of the definition it is $f'(x) = \operatorname{diag}(g'(x_1), \ldots, g'(x_n))$. Locality can be exploited in gradient computation to use a Hadamard product instead of a matrix product using the statements from Lemma 60. Consider the following result for a more formal statement:

Lemma 69. Let $g\colon \mathbb{R} \to \mathbb{R}$ be a differentiable function with derivative $g'\colon \mathbb{R} \to \mathbb{R}$. Consider the general entry-wise expansions of g and g' to matrices, $f_{a\times b}\colon \mathbb{R}^{a\times b} \to \mathbb{R}^{a\times b}$ and $f'_{a\times b}\colon \mathbb{R}^{a\times b} \to \mathbb{R}^{a\times b}$. They have the following properties for all $X \in \mathbb{R}^{a\times b}$:

(a) $\operatorname{vec}(f_{a\times b}(X)) = f_{ab\times 1}(\operatorname{vec}(X))$.

(b) $\partial \operatorname{vec}(f_{a\times b}(X)) / \partial \operatorname{vec}(X) = \operatorname{diag}(\operatorname{vec}(f'_{a\times b}(X))) = \operatorname{diag}(f'_{ab\times 1}(\operatorname{vec}(X)))$.

Some commonly used transfer functions that can be extended to local vector-valued functions are listed in the following result:

Lemma 70. The functions

$$\begin{aligned}
\mathrm{TanHyp}\colon \mathbb{R} \to \mathbb{R}, &\qquad x \mapsto \tanh(\beta x) \text{ where } \beta > 0,\\
\mathrm{TanHypPot}\colon \mathbb{R} \to \mathbb{R}, &\qquad x \mapsto (\tanh(\beta x))^q \text{ where } \beta > 0 \text{ and } q \in \mathbb{N},\ q > 0, \text{ and}\\
\mathrm{Elliot}\colon \mathbb{R} \to \mathbb{R}, &\qquad x \mapsto \tfrac{\beta x}{1+|\beta x|} \text{ where } \beta > 0
\end{aligned}$$

are differentiable everywhere. Their derivatives are given by

$$\begin{aligned}
\mathrm{TanHyp}'(x) &= \beta\left(1 - \tanh(\beta x)^2\right),\\
\mathrm{TanHypPot}'(x) &= q\beta\,(\tanh(\beta x))^{q-1}\left(1 - (\tanh(\beta x))^2\right), \text{ and}\\
\mathrm{Elliot}'(x) &= \beta/(1+|\beta x|)^2.
\end{aligned}$$

Definition 71. Using the function definitions from Lemma 70, TanHyp is called *hyperbolic tangent transfer function* [53], TanHypPot is called *exponentiated hyperbolic tangent transfer function* [79], and Elliot is called *Elliot's sigmoid transfer function* [172]. The function $\mathrm{Fermi}\colon \mathbb{R} \to \mathbb{R}$, $x \mapsto (1 + \exp(-\beta x))^{-1}$, where $\beta > 0$, is called *logistic sigmoid transfer function* or *Fermi transfer function* [53]. It is essentially a linear transformation of the hyperbolic tangent transfer function.

The number $\beta > 0$ is a parameter of these transfer functions and controls their steepness. The parameter q of TanHypPot controls the shape in the vicinity of the origin, see Section 5.2.4. A very simple non-local transfer function is the normalization with respect to the L_2 norm and the softmax transfer function encountered frequently in multi-class classification tasks:

Definition 72. The function Normlz: $\mathbb{R}^n \to \mathbb{R}^n$, $x \mapsto {}^{x}/_{\|x\|_2}$ is called *normalization transfer function*. The function Softmax: $\mathbb{R}^n \to \mathbb{R}^n$, $x \mapsto y$ where $y_i = {}^{\exp(x_i)}/_{\sum_{j=1}^{n} \exp(x_j)}$ is called *softmax transfer function*.

The softmax transfer function is especially useful for non-binary classification when class labels are represented by one-of-c codes where c is the number of classes [53]. The following result shows that the gradients of Normlz and Softmax are non-diagonal, but symmetric:

Lemma 73. The normalization transfer function is differentiable almost everywhere with gradient $\text{Normlz}'(x) = {}^{1}/_{\|x\|_2} \cdot \left(E_n - {}^{xx^T}/_{\|x\|_2^2}\right)$. The softmax transfer function is differentiable almost everywhere with derivative $\text{Softmax}'(x) = \text{diag}(y) - yy^T$, where $y = \text{Softmax}(x) \in \mathbb{R}^n$.

C.6 Similarity Measures

The models investigated in this work realize a mapping $\mathbb{R}^d \to \mathbb{R}^n$, a representation $y \in \mathbb{R}^n$ is computed from an input sample $x \in \mathbb{R}^d$. During the optimization, discrepancies between the output y and a ground truth value $t \in \mathbb{R}^n$ shall be minimized. Therefore, the formulation of the overall objective function that is to be optimized involves a *similarity measure*. These are functions $M \times M \to \mathbb{R}$, where $M \subseteq \mathbb{R}^n$, which have to be differentiable for use with gradient methods.

Definition 74. Let $M \subseteq \mathbb{R}^n$. A similarity measure $s\colon M \times M \to \mathbb{R}$ is called *minimal* if and only if $s(t,t) \leq s(y,t)$ for all $y,t \in M$, and it is called *maximal* if and only if $s(t,t) \geq s(y,t)$ for all $y,t \in M$.

As the formulation of gradient descent yields a minimizer of a function, a minimal similarity measure should be used. Note that every maximal similarity measure can be turned into a minimal one by inverting its sign for every pair of points. The perhaps most popular similarity measure is the Euclidean distance between two vectors [53]:

Definition 75. The function SE: $\mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, $(y,t) \mapsto \|t - y\|_2^2$ is called *squared error similarity measure*.

Note that the gradient of this similarity measure follows directly from Lemma 66(b) and exhibits a very simple structure. For measuring similarity of probability distributions, the Kullback-Leibler divergence [173] is well-established:

Definition 76. Let $M := \{ x \in \mathbb{R}^n_{\geq 0} \mid \sum_{i=1}^n x_i = 1 \}$ be the canonical n-simplex. Then the function $\mathrm{KL}\colon M \times M \to \mathbb{R}$, $(y,t) \mapsto \sum_{i=1}^n t_i \log\left({}^{t_i}/{}_{y_i}\right)$ is called *Kullback-Leibler divergence*, and the function $\mathrm{uKL}\colon \mathbb{R}^n_{\geq 0} \times \mathbb{R}^n_{\geq 0} \to \mathbb{R}$, $(y,t) \mapsto \sum_{i=1}^n \left(t_i \log\left({}^{t_i}/{}_{y_i}\right) + y_i - t_i\right)$ is called *unnormalized Kullback-Leibler divergence*.

A common notation for the Kullback-Leibler divergence is also given by $D(t \| y) := \mathrm{KL}(y,t)$. The unnormalized variant compensates for the scaling of y and t, and is a common measure used for Non-negative Matrix Factorization [43]. Also note that $\mathrm{uKL}(y,t) = \mathrm{KL}(y,t)$ for all $y,t \in M$ with M as given in Definition 76. Based on the Kullback-Leibler divergence there is a popular similarity measure which is used often in practice, especially for multi-class problems and in conjunction with the softmax transfer function [53, 102]:

Definition 77. When $M := \{ x \in \mathbb{R}^n_{\geq 0} \mid \sum_{i=1}^n x_i = 1 \}$ denotes the canonical n-simplex, then the function $\mathrm{CE}\colon M \times M \to \mathbb{R}$, $(y,t) \mapsto -\sum_{i=1}^n t_i \log(y_i)$ is called *cross entropy similarity measure*.

Note that the cross entropy as defined here is identical to the Kullback-Leibler divergence up to an additive constant, when t is kept fixed. This is because $\mathrm{KL}(y,t) = \mathrm{CE}(y,t) + H(t)$, where $H(t) := \sum_{i=1}^n t_i \log(t_i)$ denotes the Shannon entropy [174] of t.

Lemma 78. The following holds for the cross entropy similarity measure, where $y,t \in \mathbb{R}^n$, and where $\circledcirc$ denotes the entry-wise quotient:

(a) $\mathrm{CE}(y,t) \geq 0$.

(b) $\mathrm{CE}(y,t) \neq \mathrm{CE}(t,y)$ in general.

(c) $\mathrm{CE}(y,t) = -J_{1\times n} \cdot (t \circ \log(y))$.

(d) $\frac{\partial \mathrm{CE}(y,t)}{\partial y} = -(t \circledcirc y)^T \in \mathbb{R}^{1\times n}$.

The following similarity measure is used widely in text-mining applications [175]:

Definition 79. The function $\mathrm{CSM}\colon \mathbb{R}^n \times \mathbb{R}^n \to [-1,\ +1]$, $(y,t) \mapsto \frac{\langle y,\ t \rangle}{\|y\|_2 \cdot \|t\|_2}$ is called *cosine similarity measure*.

This measure has some useful properties, such as the invariance to scaling of its arguments. The gradient can be computed from the quotient rule.

Lemma 80. The following statements hold for the cosine similarity measure, where $y,t \in \mathbb{R}^n$, $a,b \in \mathbb{R} \setminus \{0\}$, and where $\measuredangle$ denotes the angle between two vectors:

(a) $\mathrm{CSM}(y,t) \in [-1,\ +1]$.

(b) $\mathrm{CSM}(t,t) = 1$.

(c) $\mathrm{CSM}(y,t) = \mathrm{CSM}(t,y)$.

(d) $\mathrm{CSM}(ay,bt) = \mathrm{sgn}(a)\,\mathrm{sgn}(b)\,\mathrm{CSM}(y,t)$.

(e) $\cos(\measuredangle(y,t)) = \mathrm{CSM}(y,t)$.

(f) $\frac{\partial \mathrm{CSM}(y,t)}{\partial y} = \frac{t^T}{\|y\|_2 \cdot \|t\|_2} - \frac{\mathrm{CSM}(y,t) y^T}{\|y\|_2^2}$.

Adding invariance to translation leads to the following definition [107]:

Definition 81. The function

$$\rho\colon \mathbb{R}^n \times \mathbb{R}^n \to [-1,\ 1], \qquad (y,t) \mapsto \frac{\langle y,\ t\rangle - \frac{1}{n}\langle e,\ y\rangle\langle e,\ t\rangle}{\left(\|y\|_2^2 - \frac{1}{n}\langle e,\ y\rangle^2\right)^{1/2}\left(\|t\|_2^2 - \frac{1}{n}\langle e,\ t\rangle^2\right)^{1/2}},$$

is called *correlation coefficient*. Here, $e := J_{n\times 1} \in \mathbb{R}^n$ consists of only ones.

The correlation coefficient is a measure of linear dependence of two vectors:

Lemma 82. The following holds for the correlation coefficient, where $y,t \in \mathbb{R}^n$, $a,c \in \mathbb{R}\setminus\{0\}$ and $b,d \in \mathbb{R}$:

(a) $\rho(y,t) \in [-1,\ +1]$.

(b) $\rho(t,t) = 1$.

(c) $\rho(y,t) = \rho(t,y)$.

(d) $\rho(ay+be, ct+de) = \operatorname{sgn}(a)\operatorname{sgn}(b)\rho(y,t)$.

The gradient of ρ can be derived using the quotient rule to yield

$$\left(\frac{\partial\rho(y,t)}{\partial y}\right)^T = \frac{1}{\sqrt{\alpha\beta}}\left(t - \frac{\langle e,\ t\rangle}{n}e\right) - \frac{\rho(y,t)}{\alpha}\left(y - \frac{\langle e,\ y\rangle}{n}e\right),$$

where $\alpha := \|y\|_2^2 - \frac{1}{n}\langle e,\ y\rangle^2 \in \mathbb{R}$ and $\beta := \|t\|_2^2 - \frac{1}{n}\langle e,\ t\rangle^2 \in \mathbb{R}$.

Bibliography

[1] S. B. Laughlin and T. J. Sejnowski, “Communication in Neuronal Networks,” *Science*, vol. 301, no. 5641, pp. 1870–1874, 2003.

[2] E. R. Kandel, J. H. Schwartz, and T. M. Jessell, Eds., *Principles of Neural Science*, 4th ed. McGraw-Hill, 2000.

[3] J. Karbowski, “How Does Connectivity Between Cortical Areas Depend on Brain Size? Implications for Efficient Computation,” *Journal of Computational Neuroscience*, vol. 15, no. 3, pp. 347–356, 2003.

[4] A. Mason, A. Nicoll, and K. Stratford, “Synaptic Transmission between Individual Pyramidal Neurons of the Rat Visual Cortex in vitro,” *Journal of Neuroscience*, vol. 11, no. 1, pp. 72–84, 1991.

[5] C. Holmgren, T. Harkany, B. Svennenfors, and Y. Zilberter, “Pyramidal Cell Communication within Local Networks in Layer 2/3 of Rat Neocortex,” *Journal of Physiology*, vol. 551, no. 1, pp. 139–153, 2003.

[6] H. Markram, J. Lübke, M. Frotscher, A. Roth, and B. Sakmann, “Physiology and Anatomy of Synaptic Connections between Thick Tufted Pyramidal Neurones in the Developing Rat Neocortex,” *Journal of Physiology*, vol. 500, no. 2, pp. 409–440, 1997.

[7] S. Song, P. J. Sjöström, M. Reigl, S. Nelson, and D. B. Chklovskii, “Highly Nonrandom Features of Synaptic Connectivity in Local Cortical Circuits,” *PLOS Biology*, vol. 3, no. 3, p. e68, 2005.

[8] Y. Yoshimura, J. L. M. Dantzker, and E. M. Callaway, “Excitatory Cortical Neurons Form Fine-Scale Functional Networks,” *Nature*, vol. 433, no. 7028, pp. 868–873, 2005.

[9] D. H. Hubel and T. N. Wiesel, “Receptive Fields of Single Neurones in the Cat’s Striate Cortex,” *Journal of Physiology*, vol. 148, no. 3, pp. 574–591, 1959.

[10] B. A. Olshausen and D. J. Field, “Emergence of Simple-Cell Receptive Field Properties by Learning a Sparse Code for Natural Images,” *Nature*, vol. 381, no. 6583, pp. 607–609, 1996.

[11] B. A. Olshausen and D. J. Field, “Sparse Coding of Sensory Inputs,” *Current Opinion in Neurobiology*, vol. 14, no. 4, pp. 481–487, 2004.

[12] P. Lennie, "The Cost of Cortical Computation," *Current Biology*, vol. 13, no. 6, pp. 493–497, 2003.

[13] W. B. Levy and R. A. Baxter, "Energy Efficient Neural Codes," *Neural Computation*, vol. 8, no. 3, pp. 531–543, 1996.

[14] B. A. Olshausen, "Sparse Coding Simulation Software." Available online at `http://redwood.berkeley.edu/bruno/sparsenet`.

[15] D. Gabor, "Theory of Communication," *Journal of the Institute of Electrical Engineers – Part 3: Radio and Communication Engineering*, vol. 93, no. 26, pp. 429–457, 1946.

[16] D. H. Hubel and T. N. Wiesel, "Receptive Fields and Functional Architecture of Monkey Striate Cortex," *Journal of Physiology*, vol. 195, no. 1, pp. 215–243, 1968.

[17] W. E. Vinje and J. L. Gallant, "Sparse Coding and Decorrelation in Primary Visual Cortex During Natural Vision," *Science*, vol. 287, no. 5456, pp. 1273–1276, 2000.

[18] N. Hurley and S. Rickard, "Comparing Measures of Sparsity," *IEEE Transactions on Information Theory*, vol. 55, no. 10, pp. 4723–4741, 2009.

[19] M. Rehn and F. T. Sommer, "A Network that Uses Few Active Neurones to Code Visual Input Predicts the Diverse Shapes of Cortical Receptive Fields," *Journal of Computational Neuroscience*, vol. 22, no. 2, pp. 135–146, 2007.

[20] B. K. Natarajan, "Sparse Approximate Solutions to Linear Systems," *SIAM Journal on Computing*, vol. 24, no. 2, pp. 227–234, 1995.

[21] J. Weston, A. Elisseeff, B. Schölkopf, and M. Tipping, "Use of the Zero-Norm with Linear Models and Kernel Methods," *Journal of Machine Learning Research*, vol. 3, pp. 1439–1461, 2003.

[22] R. M. Karp, "Reducibility among Combinatorial Problems," in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds., 1972, pp. 85–103.

[23] U. Schöning, *Algorithmik*. Spektrum Akademischer Verlag, 2001.

[24] L. Fortnow, "The Status of the P versus NP Problem," *Communications of the ACM*, vol. 52, no. 9, pp. 78–86, 2009.

[25] D. L. Donoho, "For Most Large Underdetermined Systems of Linear Equations the Minimal ℓ_1-norm Solution Is Also the Sparsest Solution," *Communications on Pure and Applied Mathematics*, vol. 59, no. 6, pp. 797–829, 2006.

[26] P. O. Hoyer, "Non-Negative Matrix Factorization with Sparseness Constraints," *Journal of Machine Learning Research*, vol. 5, pp. 1457–1469, 2004.

[27] A. J. Laub, *Matrix Analysis for Scientists and Engineers*. Society for Industrial and Applied Mathematics, 2004.

[28] H. Hotelling, “Analysis of a Complex of Statistical Variables Into Principal Components,” *Journal of Educational Psychology*, vol. 24, no. 6, pp. 417–441, 1933.

[29] M. Heiler and C. Schnörr, “Learning Sparse Representations by Non-Negative Matrix Factorization and Sequential Cone Programming,” *Journal of Machine Learning Research*, vol. 7, pp. 1385–1407, 2006.

[30] R. Tibshirani, “Regression Shrinkage and Selection via the Lasso,” *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[31] S. S. Chen, D. L. Donoho, and M. A. Saunders, “Atomic Decomposition by Basis Pursuit,” *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 33–61, 1998.

[32] H. Kim and H. Park, “Sparse Non-Negative Matrix Factorizations via Alternating Non-Negativity-Constrained Least Squares for Microarray Data Analysis,” *Bioinformatics*, vol. 23, no. 12, pp. 1495–1502, 2007.

[33] M. Ranzato, Y.-L. Boureau, and Y. LeCun, “Sparse Feature Learning for Deep Belief Networks,” in *Advances in Neural Information Processing Systems*, vol. 20, 2008, pp. 1185–1192.

[34] S. Boyd and L. Vandenberghe, *Convex Optimization*, 7th ed. Cambridge University Press, 2009.

[35] P. Comon, “Independent Component Analysis, a New Concept?” *Signal Processing*, vol. 36, no. 3, pp. 287–314, 1994.

[36] A. Hyvärinen, “Fast and Robust Fixed-Point Algorithms for Independent Component Analysis,” *IEEE Transactions on Neural Networks*, vol. 10, no. 3, pp. 626–634, 1999.

[37] A. Hyvärinen, J. Hurri, and P. O. Hoyer, *Natural Image Statistics – A Probabilistic Approach to Early Computational Vision*. Springer, 2009.

[38] A. J. Bell and T. J. Sejnowski, “The "Independent Components" of Natural Scenes are Edge Filters,” *Vision Research*, vol. 37, no. 23, pp. 3327–3338, 1997.

[39] J. H. van Hateren and A. van der Schaaf, “Independent Component Filters of Natural Images Compared with Simple Cells in Primary Visual Cortex,” *Proceedings of the Royal Society B: Biological Sciences*, vol. 265, no. 1394, pp. 359–366, 1998.

[40] A. Hyvärinen, J. Karhunen, and E. Oja, *Independent Component Analysis*. John Wiley & Sons, Inc., 2001.

[41] P. Paatero and U. Tapper, “Positive Matrix Factorization: A Non-Negative Factor Model with Optimal Utilization of Error Estimates of Data Values,” *Environmetrics*, vol. 5, no. 2, pp. 111–126, 1994.

[42] D. D. Lee and H. S. Seung, “Learning the Parts of Objects by Non-Negative Matrix Factorization,” *Nature*, vol. 401, no. 6755, pp. 788–791, 1999.

[43] D. D. Lee and H. S. Seung, “Algorithms for Non-Negative Matrix Factorization,” in *Advances in Neural Information Processing Systems*, vol. 13, 2001, pp. 556–562.

[44] S. Z. Li, X. Hou, H. Zhang, and Q. Cheng, “Learning Spatially Localized, Parts-Based Representation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2001, pp. 207–212.

[45] Y. Wang, Y. Jia, C. Hu, and M. Turk, “Fisher Non-Negative Matrix Factorization for Learning Local Features,” in *Asian Conference on Computer Vision*, 2004.

[46] K. Kavukcuoglu, M. Ranzato, and Y. LeCun, “Fast Inference in Sparse Coding Algorithms with Applications to Object Recognition,” Computational and Biological Learning Laboratory, Courant Institute, New York University, Tech. Rep. CBLL-TR-2008-12-01, 2008.

[47] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman, “Supervised Dictionary Learning,” in *Advances in Neural Information Processing Systems*, vol. 21, 2009, pp. 1033–1040.

[48] D. M. Bradley and J. A. Bagnell, “Differentiable Sparse Coding,” in *Advances in Neural Information Processing Systems*, vol. 21, 2009, pp. 113–120.

[49] J. Kivinen and M. K. Warmuth, “Exponentiated Gradient versus Gradient Descent for Linear Predictors,” *Information and Computation*, vol. 132, no. 1, pp. 1–67, 1997.

[50] Y. LeCun, J. S. Denker, and S. A. Solla, “Optimal Brain Damage,” in *Advances in Neural Information Processing Systems*, vol. 2, 1990, pp. 598–605.

[51] B. Hassibi and D. G. Stork, “Second Order Derivatives for Network Pruning: Optimal Brain Surgeon,” in *Advances in Neural Information Processing Systems*, vol. 5, 1993, pp. 164–171.

[52] Y. LeCun and C. Cortes, “The MNIST Database of Handwritten Digits,” 1998. Available online at `http://yann.lecun.com/exdb/mnist`.

[53] C. M. Bishop, *Neural Networks for Pattern Recognition*. Clarendon Press, 1995.

[54] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-Based Learning Applied to Document Recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[55] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A Fast Learning Algorithm for Deep Belief Nets,” *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[56] B. Schölkopf, “Support Vector Learning,” Ph.D. dissertation, Technische Universität Berlin, 1997.

[57] D. DeCoste and B. Schölkopf, “Training Invariant Support Vector Machines,” *Machine Learning*, vol. 46, no. 1–3, pp. 161–190, 2002.

[58] P. Y. Simard, D. Steinkraus, and J. C. Platt, “Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis,” in *Proceedings of the International Conference on Document Analysis and Recognition*, 2003, pp. 958–963.

[59] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, “Deep, Big, Simple Neural Nets for Handwritten Digit Recognition,” *Neural Computation*, vol. 22, no. 12, pp. 3207–3220, 2010.

[60] D. C. Cireşan, U. Meier, and J. Schmidhuber, “Multi-Column Deep Neural Networks for Image Classification,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3642–3649.

[61] N. Le Roux, Y. Bengio, P. Lamblin, M. Joliveau, and B. Kégl, “Learning the 2-D Topology of Images,” in *Advances in Neural Information Processing Systems*, vol. 20, 2008, pp. 841–848.

[62] J. B. Tenenbaum, V. de Silva, and J. C. Langford, “A Global Geometric Framework for Nonlinear Dimensionality Reduction,” *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[63] F. J. Theis, K. Stadlthanner, and T. Tanaka, “First Results on Uniqueness of Sparse Non-Negative Matrix Factorization,” in *Proceedings of the European Signal Processing Conference*, vol. 3, 2005, pp. 1672–1675.

[64] J. von Neumann, *Functional Operators, Volume II: The Geometry of Orthogonal Spaces*. Princeton University Press, 1950.

[65] R. L. Dykstra, “An Algorithm for Restricted Least Squares Regression,” *Journal of the American Statistical Association*, vol. 78, no. 384, pp. 837–842, 1983.

[66] M. Thom and G. Palm, “Sparse Activity and Sparse Connectivity in Supervised Learning,” *Journal of Machine Learning Research*, vol. 14, pp. 1091–1143, 2013.

[67] F. Deutsch, *Best Approximation in Inner Product Spaces*. Springer, 2001.

[68] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra, “Efficient Projections onto the ℓ_1-Ball for Learning in High Dimensions,” in *Proceedings of the International Conference on Machine Learning*, 2008, pp. 272–279.

[69] T. Blumensath and M. E. Davies, “A Simple, Efficient and Near Optimal Algorithm for Compressed Sensing,” in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2009, pp. 3357–3360.

[70] C. Michelot, “A Finite Algorithm for Finding the Projection of a Point onto the Canonical Simplex of $\mathbb{R}^n$,” *Journal of Optimization Theory and Applications*, vol. 50, no. 1, pp. 195–200, 1986.

[71] Y. Chen and X. Ye, “Projection Onto A Simplex,” University of Florida, Tech. Rep. arXiv:1101.6081v2, 2011.

[72] J. Liu and J. Ye, “Efficient Euclidean Projections in Linear Time,” in *Proceedings of the International Conference on Machine Learning*, 2009, pp. 657–664.

[73] J.-B. Hiriart-Urruty, “At What Points is the Projection Mapping Differentiable?” *The American Mathematical Monthly*, vol. 89, no. 7, pp. 456–458, 1982.

[74] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley, “An Updated Set of Basic Linear Algebra Subprograms (BLAS),” *ACM Transactions on Mathematical Software*, vol. 28, no. 2, pp. 135–151, 2002.

[75] K. Goto and R. A. van de Geijn, “Anatomy of High-Performance Matrix Multiplication,” *ACM Transactions on Mathematical Software*, vol. 34, no. 3, pp. 12:1–12:25, 2008.

[76] K. Goto and R. A. van de Geijn, “High-Performance Implementation of the Level-3 BLAS,” *ACM Transactions on Mathematical Software*, vol. 35, no. 1, pp. 4:1–4:14, 2008.

[77] V. K. Potluru, S. M. Plis, J. Le Roux, B. A. Pearlmutter, V. D. Calhoun, and T. P. Hayes, “Block Coordinate Descent for Sparse NMF,” in *Proceedings of the International Conference on Learning Representations*. arXiv:1301.3527v2, 2013.

[78] M. Thom and G. Palm, “Efficient Sparseness-Enforcing Projections,” Ulm University, Tech. Rep. arXiv:1303.5259v1, 2013.

[79] M. Thom, R. Schweiger, and G. Palm, “Supervised Matrix Factorization with Sparseness Constraints and Fast Inference,” in *Proceedings of the International Joint Conference on Neural Networks*, 2011, pp. 973–979.

[80] B. Willmore and D. J. Tolhurst, “Characterizing the Sparseness of Neural Codes,” *Network: Computation in Neural Systems*, vol. 12, no. 3, pp. 255–270, 2001.

[81] A. Gut, *Probability: A Graduate Course*. Springer, 2005.

[82] S. H. Rice, “The Expected Value of the Ratio of Correlated Random Variables,” Texas Tech University, Tech. Rep., 2009.

[83] R. Heijmans, “When Does the Expectation of a Ratio Equal the Ratio of Expectations?” *Statistical Papers*, vol. 40, no. 1, pp. 107–115, 1999.

[84] D. Guillamet and J. Vitrià, “Determining a Suitable Metric When Using Non-Negative Matrix Factorization,” in *Proceedings of the International Conference on Pattern Recognition*, vol. 2, 2002, pp. 128–131.

[85] C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems*. Society for Industrial and Applied Mathematics, 1995.

[86] K. Bredies and D. A. Lorenz, “Linear Convergence of Iterative Soft-Thresholding,” *Journal of Fourier Analysis and Applications*, vol. 14, no. 5–6, pp. 813–837, 2008.

[87] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, “Chapter 1: Efficient BackProp,” in *Neural Networks: Tricks of the Trade*, 2nd ed., G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Springer, 2012, pp. 9–48.

[88] Y. LeCun, I. Kanter, and S. A. Solla, “Eigenvalues of Covariance Matrices: Application to Neural-Network Learning,” *Physical Review Letters*, vol. 66, no. 18, pp. 2396–2399, 1991.

[89] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning Representations by Back-Propagating Errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[90] G. Cybenko, “Approximation by Superpositions of a Sigmoidal Function,” *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, pp. 303–314, 1989.

[91] K. Funahashi, “On the Approximate Realization of Continuous Mappings by Neural Networks,” *Neural Networks*, vol. 2, no. 3, pp. 183–192, 1989.

[92] K. Hornik, M. Stinchcombe, and H. White, “Multilayer Feedforward Networks are Universal Approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[93] K. Gregor and Y. LeCun, “Learning Fast Approximations of Sparse Coding,” in *Proceedings of the International Conference on Machine Learning*, 2010, pp. 399–406.

[94] X. Glorot, A. Bordes, and Y. Bengio, “Deep Sparse Rectifier Neural Networks,” in *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.

[95] L. Dagum and R. Menon, “OpenMP: An Industry-Standard API for Shared-Memory Programming,” *IEEE Computational Science & Engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[96] G. M. Amdahl, “Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities,” in *Proceedings of the Spring Joint Computer Conference*, 1967, pp. 483–485.

[97] B. A. Olshausen and D. J. Field, “Sparse Coding with an Overcomplete Basis Set: A Strategy Employed by V1?” *Vision Research*, vol. 37, no. 23, pp. 3311–3325, 1997.

[98] J. A. Nelder and R. Mead, “A Simplex Method for Function Minimization,” *The Computer Journal*, vol. 7, no. 4, pp. 308–313, 1965.

[99] C. J. C. Burges, “A Tutorial on Support Vector Machines for Pattern Recognition,” *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[100] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, “LIBLINEAR: A Library for Large Linear Classification,” *Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.

[101] M. Thom, R. Schweiger, and G. Palm, “Training of Sparsely Connected MLPs,” in *Lecture Notes in Computer Science*, vol. 6835, 2011, pp. 356–365.

[102] R. A. Dunne and N. A. Campbell, “On The Pairing Of The Softmax Activation And Cross-Entropy Penalty Functions And The Derivation Of The Softmax Activation Function,” in *Australasian Conference on Neural Networks*, 1997, pp. 181–185.

[103] D. P. Bertsekas, *Nonlinear Programming*, 2nd ed. Athena Scientific, 1999.

[104] J. R. Rice and R. F. Boisvert, *Solving Elliptic Problems Using ELLPACK*. Springer, 1985.

[105] S. Geman, E. Bienenstock, and R. Doursat, “Neural Networks and the Bias/Variance Dilemma,” *Neural Computation*, vol. 4, no. 1, pp. 1–58, 1992.

[106] A. Hyvärinen, P. O. Hoyer, and E. Oja, “Sparse Code Shrinkage: Denoising by Nonlinear Maximum Likelihood Estimation,” in *Advances in Neural Information Processing Systems*, vol. 11, 1999, pp. 473–479.

[107] J. L. Rodgers and W. A. Nicewander, “Thirteen Ways to Look at the Correlation Coefficient,” *The American Statistician*, vol. 42, no. 1, pp. 59–66, 1988.

[108] J. Demšar, “Statistical Comparisons of Classifiers over Multiple Data Sets,” *Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

[109] J. Pizarro, E. Guerrero, and P. L. Galindo, “Multiple Comparison Procedures Applied to Model Selection,” *Neurocomputing*, vol. 48, no. 1–4, pp. 155–173, 2002.

[110] R. J. Grissom, “Probability of the Superior Outcome of One Treatment Over Another,” *Journal of Applied Psychology*, vol. 79, no. 2, pp. 314–316, 1994.

[111] L. Acion, J. J. Peterson, S. Temple, and S. Arndt, “Probabilistic Index: an Intuitive Non-Parametric Approach to Measuring the Size of Treatment Effects,” *Statistics in Medicine*, vol. 25, no. 4, pp. 591–602, 2006.

[112] S. S. Shapiro and M. B. Wilk, “An Analysis of Variance Test for Normality (Complete Samples),” *Biometrika*, vol. 52, no. 3–4, pp. 591–611, 1965.

[113] H. Levene, “Robust Tests for Equality of Variances,” in *Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling*, I. Olkin, S. G. Ghurye, W. Hoeffding, W. G. Madow, and H. B. Mann, Eds. Stanford University Press, 1960, pp. 278–292.

[114] W. H. Kruskal and W. A. Wallis, “Use of Ranks in One-Criterion Variance Analysis,” *Journal of the American Statistical Association*, vol. 47, no. 260, pp. 583–621, 1952.

[115] Y. Hochberg and A. C. Tamhane, *Multiple Comparison Procedures.* John Wiley & Sons, Inc., 1987.

[116] F. Downton, "The Estimation of Pr(Y < X) in the Normal Case," *Technometrics*, vol. 15, no. 3, pp. 551–558, 1973.

[117] J.-M. Muller, *Elementary Functions: Algorithms and Implementation*, 2nd ed. Birkhäuser, 2006.

[118] R. Tolimieri, M. An, and C. Lu, *Algorithms for Discrete Fourier Transform and Convolution*, 2nd ed. Springer, 1997.

[119] D. Scherer, A. Müller, and S. Behnke, "Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition," in *Lecture Notes in Computer Science*, vol. 6354, 2010, pp. 92–101.

[120] J. Schürmann, *Polynomklassifikatoren für die Zeichenerkennung: Ansatz, Adaption, Anwendungen.* Oldenbourg, 1977.

[121] U. Kreßel and J. Schürmann, "Chapter 2: Pattern Classification Techniques Based on Function Approximation," in *Handbook on Optical Character Recognition and Document Image Analysis*, H. Bunke and P. S. P. Wang, Eds. World Scientific Publishing Company, 1997, pp. 49–78.

[122] P. Viola and M. Jones, "Rapid Object Detection Using a Boosted Cascade of Simple Features," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2001, pp. I–511–I–518.

[123] D. Benbouzid, R. Busa-Fekete, N. Casagrande, F.-D. Collin, and B. Kégl, "MultiBoost: A Multi-Purpose Boosting Package," *Journal of Machine Learning Research*, vol. 13, pp. 549–553, 2012.

[124] P. Ein-Dor, "Grosch's Law Re-Revisited: CPU Power and the Cost of Computation," *Communications of the ACM*, vol. 28, no. 2, pp. 142–151, 1985.

[125] G. E. Hinton and R. R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[126] U. Meier, D. C. Cireşan, L. M. Gambardella, and J. Schmidhuber, "Better Digit Recognition with a Committee of Simple Neural Nets," in *Proceedings of the International Conference on Document Analysis and Recognition*, 2011, pp. 1250–1254.

[127] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Handwritten Digit Recognition with a Back-Propagation Network," in *Advances in Neural Information Processing Systems*, vol. 2, 1990, pp. 396–404.

[128] M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun, “Efficient Learning of Sparse Representations with an Energy-Based Model,” in *Advances in Neural Information Processing Systems*, vol. 19, 2007, pp. 1137–1144.

[129] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, “Convolutional Neural Network Committees for Handwritten Character Classification,” in *Proceedings of the International Conference on Document Analysis and Recognition*, 2011, pp. 1135–1139.

[130] C. J. C. Burges and B. Schölkopf, “Improving the Accuracy and Speed of Support Vector Machines,” in *Advances in Neural Information Processing Systems*, vol. 9, 1997, pp. 375–381.

[131] K. Labusch, E. Barth, and T. Martinetz, “Simple Method for High-Performance Digit Recognition Based on Sparse Coding,” *IEEE Transactions on Neural Networks*, vol. 19, no. 11, pp. 1985–1989, 2008.

[132] U. Kreßel, “Chapter 15: Pairwise Classification and Support Vector Machines,” in *Advances in Kernel Methods: Support Vector Learning*, B. Schölkopf, C. J. C. Burges, and A. J. Smola, Eds. MIT Press, 1998, pp. 255–268.

[133] B. Kégl and R. Busa-Fekete, “Boosting Products of Base Classifiers,” in *Proceedings of the International Conference on Machine Learning*, 2009, pp. 497–504.

[134] World Health Organization, *Global Status Report on Road Safety: Time for Action*. WHO Press, Geneva, 2009.

[135] S. Plainis, I. J. Murray, and I. G. Pallikaris, “Road Traffic Casualties: Understanding the Night-Time Death Toll,” *Injury Prevention*, vol. 12, no. 2, pp. 125–128, 2006.

[136] M. Thom, W. Ritter, and J. Moisel, “Pedestrian Highlighting Using Programmable LED Headlights,” in *Proceedings of the International Symposium on Automotive Lighting*, 2011, pp. 1032–1046.

[137] M. Thom, “Verfahren zum Detektieren eines Objekts für ein Fahrerassistenzsystem,” German Patent DE 10 2010 033 773 A1, 2012.

[138] P. Knoll, “Nachtsichtsysteme,” in *Fahrstabilisierungssysteme und Fahrerassistenzsysteme*, K. Reif, Ed. Vieweg+Teubner Verlag, 2010, pp. 210–213.

[139] J.-E. Källhammer, “Imaging: The Road Ahead for Car Night-Vision,” *Nature Photonics*, pp. 12–13, September 2006.

[140] R. Schweiger, “Echtzeitfähiges Fusionssystem zur Fußgängererkennung bei Nacht,” Ph.D. dissertation, Universität Ulm, 2012.

[141] H. Eggers, J. Moisel, S. Töpfer, and S. Tattersall, "A Night Vision System with Spotlight / Marking Light," in *Proceedings of the International Symposium on Automotive Lighting*, 2011, pp. 471–483.

[142] J. Moisel, R. Ackermann, and M. Griesinger, "Adaptive Headlights Utilizing LED-Arrays," in *Proceedings of the International Symposium on Automotive Lighting*, 2009, pp. 287–296.

[143] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2005, pp. 886–893.

[144] P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian Detection: An Evaluation of the State of the Art," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 4, pp. 743–761, 2012.

[145] A. P. Bradley, "The Use of the Area Under the ROC Curve in the Evaluation of Machine Learning Algorithms," *Pattern Recognition*, vol. 30, no. 7, pp. 1145–1159, 1997.

[146] J. W. Brewer, "Kronecker Products and Matrix Calculus in System Theory," *IEEE Transactions on Circuits and Systems*, vol. 25, no. 9, pp. 772–781, 1978.

[147] H. Neudecker, "Some Theorems on Matrix Differentiation with Special Reference to Kronecker Matrix Products," *Journal of the American Statistical Association*, vol. 64, no. 327, pp. 953–963, 1969.

[148] D. S. Dummit and R. M. Foote, *Abstract Algebra*, 3rd ed. John Wiley & Sons, Inc., 2004.

[149] K. B. Petersen and M. S. Pedersen, "The Matrix Cookbook," Technical University of Denmark, 2008, version November 14, 2008.

[150] W. J. Vetter, "Derivative Operations on Matrices," *IEEE Transactions on Automatic Control*, vol. 15, no. 2, pp. 241–244, 1970.

[151] S. Hildebrandt, *Analysis 1*, 2nd ed. Springer, 2006.

[152] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977.

[153] D. R. Wilson and T. R. Martinez, "The General Inefficiency of Batch Training for Gradient Descent Learning," *Neural Networks*, vol. 16, no. 10, pp. 1429–1451, 2003.

[154] U. A. Müller, A. Gunzinger, and W. Guggenbühl, "Fast Neural Net Simulation with a DSP Processor Array," *IEEE Transactions on Neural Networks*, vol. 6, no. 1, pp. 203–213, 1995.

[155] L. Bottou and Y. LeCun, "Large Scale Online Learning," in *Advances in Neural Information Processing Systems*, vol. 16, 2004, pp. 217–224.

[156] G. Louppe and P. Geurts, "A Zealous Parallel Gradient Descent Algorithm," in *NIPS Workshop on Learning on Cores, Clusters and Clouds*, 2010.

[157] D. Jungnickel, *Optimierungsmethoden: Eine Einführung*, 2nd ed. Springer, 2008.

[158] C. Darken, J. Chang, and J. Moody, "Learning Rate Schedules for Faster Stochastic Gradient Search," in *Proceedings of the IEEE-SP Workshop on Neural Networks for Signal Processing*, 1992, pp. 3–12.

[159] B. A. Pearlmutter, "Fast Exact Multiplication by the Hessian," *Neural Computation*, vol. 6, no. 1, pp. 147–160, 1994.

[160] W. L. Buntine and A. S. Weigend, "Computing Second Derivatives In Feed-Forward Networks: A Review," *IEEE Transactions on Neural Networks*, vol. 5, no. 3, pp. 480–488, 1994.

[161] Y. LeCun, P. Y. Simard, and B. Pearlmutter, "Automatic Learning Rate Maximization by On-Line Estimation of the Hessian's Eigenvectors," in *Advances in Neural Information Processing Systems*, vol. 5, 1993, pp. 156–163.

[162] C. Darken and J. Moody, "Note on Learning Rate Schedules for Stochastic Optimization," in *Advances in Neural Information Processing Systems*, vol. 3, 1991, pp. 832–838.

[163] T. P. Vogl, J. K. Mangis, A. K. Rigler, W. T. Zink, and D. L. Alkon, "Accelerating the Convergence of the Back-Propagation Method," *Biological Cybernetics*, vol. 59, no. 4–5, pp. 257–263, 1988.

[164] R. Wagner, M. Thom, R. Schweiger, G. Palm, and A. Rothermel, "Learning Convolutional Neural Networks From Few Samples," in *Proceedings of the International Joint Conference on Neural Networks*, 2013, pp. 1884–1890.

[165] T. Denœux and R. Lengellé, "Initializing Back Propagation Networks with Prototypes," *Neural Networks*, vol. 6, no. 3, pp. 351–363, 1993.

[166] A. Coates and A. Y. Ng, "The Importance of Encoding Versus Training with Sparse Coding and Vector Quantization," in *Proceedings of the International Conference on Machine Learning*, 2011, pp. 921–928.

[167] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy Layer-Wise Training of Deep Networks," in *Advances in Neural Information Processing Systems*, vol. 19, 2007, pp. 153–160.

[168] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why Does Unsupervised Pre-Training Help Deep Learning?" *Journal of Machine Learning Research*, vol. 11, pp. 625–660, 2010.

[169] X. Glorot and Y. Bengio, "Understanding the Difficulty of Training Deep Feedforward Neural Networks," in *Proceedings of the International Conference on Artificial Intelligence and Statistics*, vol. 9, 2010, pp. 249–256.

[170] J. Hochreiter, "Untersuchungen zu dynamischen neuronalen Netzen," Diploma thesis, Technische Universität München, 1991.

[171] T. M. Cover, "Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition," *IEEE Transactions on Electronic Computers*, vol. EC-14, no. 3, pp. 326–334, 1965.

[172] D. Elliott, "A Better Activation Function for Artificial Neural Networks," Institute for Systems Research, University of Maryland, Tech. Rep. ISR TR 93-8, 1993.

[173] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *Anneals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[174] C. E. Shannon, "A Mathematical Theory of Communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[175] M.-C. Kim and K.-S. Choi, "A Comparison of Collocation-Based Similarity Measures in Query Expansion," *Information Processing and Management*, vol. 35, no. 1, pp. 19–30, 1999.

Publications

M. Thom, R. Schweiger, and G. Palm, "Supervised Matrix Factorization with Sparseness Constraints and Fast Inference," in *Proceedings of the International Joint Conference on Neural Networks*, 2011, pp. 973–979.

M. Thom, R. Schweiger, and G. Palm, "Training of Sparsely Connected MLPs," in *Lecture Notes in Computer Science*, vol. 6835, 2011, pp. 356–365.

M. Thom, M. Ulken, L. Krüger, and W. Ritter, "Headlight Range Calibration During Driving Operation," in *Proceedings of the International Symposium on Automotive Lighting*, 2011, pp. 535–549.

M. Thom, W. Ritter, and J. Moisel, "Pedestrian Highlighting Using Programmable LED Headlights," in *Proceedings of the International Symposium on Automotive Lighting*, 2011, pp. 1032–1046.

M. Thom and G. Palm, "Sparse Activity and Sparse Connectivity in Supervised Learning," *Journal of Machine Learning Research*, vol. 14, pp. 1091–1143, 2013.

Co-authored Publications

T. Ehlgen, M. Thom, and M. Glaser, "Omnidirectional Cameras as Backing-Up Aid," in *Proceedings of the International Conference on Computer Vision*, 2007.

R. Wagner, M. Thom, R. Schweiger, M. Gabb, A. Röhlig, and A. Rothermel, "Influence of Image Compression on Cascade Classifier Components," in *Proceedings of the IEEE Symposium on Intelligent Systems and Informatics*, 2012, pp. 367–371.

R. Wagner, M. Thom, M. Gabb, M. Limmer, R. Schweiger, and A. Rothermel, "Convolutional Neural Networks for Night-Time Animal Orientation Estimation," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2013, pp. 316–321.

R. Wagner, M. Thom, R. Schweiger, G. Palm, and A. Rothermel, "Learning Convolutional Neural Networks From Few Samples," in *Proceedings of the International Joint Conference on Neural Networks*, 2013, pp. 1884–1890.

R. Wagner, M. Thom, M. Gabb, C. Feller, R. Schweiger, and A. Rothermel, "Image Compression with Discriminative Dictionaries," in *Proceedings of the IEEE International Conference on Consumer Electronics – Berlin*, 2013, pp. 249–253.

D. Nuss, M. Thom, A. Danzer, and K. Dietmayer, "Fusion of Laser and Monocular Camera Data in Object Grid Maps for Vehicle Environment Perception," in *Proceedings of the International Conference on Information Fusion*, 2014.

Patent Applications

O. Löhlein, W. Ritter, R. Schweiger, and M. Thom, "Verfahren zur Mustererkennung," *DE 10 2009 019 025.2*, April 2009.

H. Eggers, M. Lallinger, J. Moisel, W. Ritter, S. Tattersall, M. Thom, and B. Woltermann, "Verfahren und Vorrichtung zur Steuerung eines Fahrlichts eines Fahrzeugs," *DE 10 2009 051 485.6*, October 2009.

M. Thom and M. Serfling, "Verfahren zum Justieren und/oder Kalibrieren einer optischen Einheit eines Fahrzeugs," *DE 10 2010 010 909.6*, March 2010.

W. Ritter, M. Thom, and D. Pollinger, "Vorrichtung zum Sauberhalten eines optischen Elements oder einer vor diesem angeordneten Schutzabdeckung," *DE 10 2010 022 163.5*, May 2010.

T. Ruland and M. Thom, "Verfahren und Vorrichtung zur Justierung einer an einem Fahrzeug angeordneten Erfassungseinheit," *DE 10 2010 027 344.9*, July 2010.

M. Thom, "Verfahren zum Detektieren eines Objekts für ein Fahrerassistenzsystem," *DE 10 2010 033 773.0*, August 2010.

O. Löhlein, W. Ritter, F. Schüle, R. Schweiger, and M. Thom, "Verfahren zum aufmerksamkeitsabhängigen Initiieren einer Fahrzeugaktion und Fahrerassistenzsystem zur Durchführung des Verfahrens," *DE 10 2010 048 273.0*, October 2010.

M. Thom, S. Hahn, W. Ritter, and J. Gloger, "Verfahren zum Justieren und/oder Kalibrieren zumindest eines Scheinwerfers eines Fahrzeugs," *DE 10 2010 048 689.2*, October 2010.

M. Szczot, M. Serfling, O. Löhlein, W. Ritter, F. Schüle, R. Schweiger, and M. Thom, "Verfahren zur Bestimmung eines Fahrspurverlaufs für ein Fahrzeug," *DE 10 2010 049 214.0*, October 2010.

O. Löhlein, W. Ritter, F. Schüle, R. Schweiger, and M. Thom, "Verfahren zur Bestimmung einer Fahrzeugumgebung," *DE 10 2010 049 215.9*, October 2010.

L. Viglieri, O. Männlein, and M. Thom, "Verfahren und eine Vorrichtung zur Einstellung eines Fahrzeugscheinwerfers," *DE 10 2010 051 772.0*, November 2010.

M. Thom and O. Männlein, "Verfahren und Vorrichtung zur Einstellung eines Fahrzeugscheinwerfers während einer Fahrt eines Fahrzeugs," *DE 10 2011 100 614.5*, May 2011.

H. Eggers, J. Gloger, S. Hahn, L. Krüger, J. Moisel, W. Ritter, J. Seekircher, M. Thom, M. Ulken, and B. Woltermann, "Verfahren zum Justieren und/oder Kalibrieren zumindest eines Scheinwerfers eines Fahrzeugs," *DE 10 2011 109 440.0*, August 2011.

M. Thom, H. Eggers, M. Lallinger, J. Moisel, W. Ritter, and R. Schweiger, "Verfahren zur Steuerung einer Fahrlichtverteilung für ein Fahrzeug," *DE 10 2012 002 226.3*, February 2012.