Technische Universität Berlin
Fakultät für Elektrotechnik und Informatik

Extraction of visual features from natural video data using Slow Feature Analysis

Diplomarbeit im Studiengang Informatik

vorgelegt von Hannes Nickisch
Matrikel 205 837

Berlin, September 2006

Aufgabensteller:
Prof. Dr. Klaus Obermayer, Fachgebiet Neuronale Informationsverarbeitung

Mitberichter:
Prof. Dr. Günter Hommel, Fachgebiet Prozessdatenverarbeitung und Robotik

Betreuer:
Dipl.-Ing. Roland Vollgraf, Fachgebiet Neuronale Informationsverarbeitung

Eidesstattliche Erklärung

Die selbständige und eigenhändige Anfertigung versichere ich an Eides statt.

Berlin, den 07. September 2006

Hannes Nickisch

Acknowledgments

My initiation to the exciting fields of neural information processing and machine learning took place in Prof. Obermayer's lectures of the NI group and was deepened during project work as a student assistant. Special support was provided by my office mate and supervisor Roland, who rapidly and exhaustively responded to my probing questions.
Laurenz Wiskott's group offered helpful comments and suggestions during a presentation of intermediate results from this thesis.
Last but not least, I would like to thank Susanne for putting up with me and for her proof-reading as well as my parents for financial support and constant sympathy.

This work was funded by Deutsche Forschungsgemeinschaft under grant no. DFG OB 102/7-1.

Zusammenfassung

Das Forschungsprojekt NeuRoBot hat das unüberwachte Erlernen einer neuronal inspirierten Steuerungsarchitektur zum Ziel, und zwar unter den Randbedingungen biologischer Plausibilität und der Benutzung einer Kamera als einzigen Sensor. Visuelle Merkmale, die ein angemessenes Abbild der Umgebung liefern, sind unerlässlich, um das Ziel kollisionsfreier Navigation zu erreichen.

Zeitliche Kohärenz ist ein neues Lernprinzip, das in der Lage ist, Erkenntnisse aus der Biologie des Sehens zu reproduzieren. Es wird durch die Beobachtung motiviert, dass die "Sensoren" der Retina auf deutlich kürzeren Zeitskalen variieren als eine abstrakte Beschreibung. Zeitliche Langsamkeitsanalyse löst das Problem, indem sie zeitlich langsam veränderliche Signale aus schnell veränderlichen Eingabesignalen extrahiert. Eine Verallgemeinerung auf Signale, die nichtlinear von den Eingaben abhängen, ist durch die Anwendung des Kernel-Tricks möglich. Das einzig benutzte Vorwissen ist die zeitliche Glattheit der gewonnenen Signale.

In der vorliegenden Diplomarbeit wird Langsamkeitsanalyse auf Bildausschnitte von Videos einer Roboterkamera und einer Simulationsumgebung angewendet. Zuallererst werden mittels Parameterexploration und Kreuzvalidierung die langsamst möglichen Funktionen bestimmt. Anschließend werden die Merkmalsfunktionen analysiert und einige Ansatzpunkte für ihre Interpretation angegeben. Aufgrund der sehr großen Datensätze und der umfangreichen Berechnungen behandelt ein Großteil dieser Arbeit auch Aufwandsbetrachtungen und Fragen der effizienten Berechnung.

Kantendetektoren in verschiedenen Phasen und mit hauptsächlich horizontaler Orientierung stellen die wichtigsten aus der Analyse hervorgehenden Funktionen dar. Eine Anwendung auf konkrete Navigationsaufgaben des Roboters konnte bisher nicht erreicht werden. Eine visuelle Interpretation der erlernten Merkmale ist jedoch durchaus gegeben.

Abstract

The NeuRoBot project aims at unsupervisedly learning a control architecture for an autonomous mobile robot under the constraint of biological plausibility and the use of a stereo camera as single sensory device. Visual features providing an appropriate representation of the scene are essential for approaching the goal of navigation and collision avoidance.

Temporal coherence is a recent learning paradigm, able to reproduce findings from low-level vision. It is motivated by the observation that the "primary sensors" of the retina vary on much smaller time scales than a high-level description. Slow Feature Analysis solves the problem of extracting slowly varying signals from quickly varying sensory input. A generalization to signals depending nonlinearly on the input is possible by virtue of the kernel trick. The only prior knowledge is temporal smoothness of the extracted signals.

In this thesis, Slow Feature Analysis is applied to image patches of videos from a robot's camera and a simulator. In a first step, optimally slow functions are identifed by parameter exploration and cross-validation. Subsequently, the feature projections are analyzed and some clues for interpretation are supplied. Repeated application of the method leads to a multi-layer architecture which is briefly explored. Due to the large data sets and the very costly calculations, a part of the work also comprises complexity considerations and means of efficient computation.

Edge detectors in different phases and essentially in horizontal orientation constitute the most prominent class of functions emerging from the analysis. A concrete application to robot navigation could not be achieved but visual interpretation of the learned feature images is possible.

Contents

List of Figures

Chapter 1

Introduction

Artificial intelligence, a subfield of computer science, addresses learning and intelligent behavior in machines. First experimental approaches, going back to the 1950s, can be divided into two major branches:

- The former, called the symbolic, classic or neat approach consists in the manipulation of abstract symbolic concepts paving the way to expert systems.
- The latter, termed connectionist or scruffy approach, rather focuses on evolution of intelligence by some process and blossoms into the famous neural networks.

Even the combination of both directions suffers from severe limitations.

Below the line, the early goals such as imitation and simulation of human behavior were totally missed. Marvin Minsky[1], one of the early pioneers, sees "no progress towards a general intelligence". He is even quoted by Drösser [2006] claiming "artificial intelligence is braindead". Man and machine seem to have very opposed capabilities: Computers can perform fast sequential calculations, man is able to evaluate, interpret and understand in parallel.

New attempts, like the concept of behavior-based artificial creatures situated in the world, as promoted by Rodney Brooks[2], receive inspirations from biology and focus more on the performance of a machine than on the processes inside. Many researchers including Sebastian Thrun[3] are of the opinion that "general and everyday knowledge is the 'Holy Grail' of artificial intelligence". There are projects collecting large databases of such facts, but up to now, nobody is able to build a machine taking advantage of it.

Modern machine learning is strongly rooted in statistics and pursues far less ambitious objectives. One is occupied by extracting knowledge from

[1]Professor at the MIT Media and Artificial Intelligence Laboratories and one of the initiators of the famous Darmouth Conference: `http://web.media.mit.edu/~minsky/`

[2]Professor of robotics and director of the MIT Computer Science and Artificial Intelligence Laboratory (CSAIL): `http://people.csail.mit.edu/brooks/`

[3]Associate professor and director of the Stanford Artificial Intelligence Laboratory (SAIL): `http://robots.stanford.edu/`

large corpuses of data, unsupervisedly learning the structure of data sets, recognizing patterns, predicting the generalization of statistical methods and by finding evidence for hypotheses. Machine learning is a practical field solving more concrete problems, e.g. control or speech recognition. Topics like genuine machine intelligence and conscience are rather considered to be philosophical gimmicks.

The NeuRoBot project is a fundamental research study combining robotics and machine learning. As major guideline serves the question, to what extent, recent methods of neural information processing are capable of learning adaptive control.

1.1 Research Project NeuRoBot

This thesis is part of the DFG project NeuRoBot [Musial et al., 2004] which aims at developing a neurally and biologically inspired steering architecture for a mobile robot. Initiated as a collaboration between the Neural Information Processing Group and the Real Time Systems & Robotics Group of the Technische Universität Berlin, a variety of fields of research is involved. For a period of 3 years, starting in July 2004, a group of six people, comprising one post-doc position, one PhD student and four student assistants, jointly work on the project.

The main tasks to solve are collision avoidance and navigation to known places. However, the focus is on biological plausibility rather than on true performance. Much of the learning is supposed to be done in a simulator environment which mainly consists of a modified game engine to prevent the robot from damage, to have an absolutely controlled scenario and to speed up simulations. After a pre-learning phase, the stereo camera serves as solitary sensor.

Continuous reinforcement learning is evaluated based on the sonar beam measurements for collision avoidance by the Real Time Systems & Robotics Group. Furthermore classification and recognition abilities of Leabra networks are assessed. Also the simulator and onboard software of the robot is developed by the Robotics group. Any symbolic body of rules is prevented and neural learning methods are used instead.

Low-level visual information processing and feature extraction falls in the domain of the Neural Information Processing Group. From the camera inputs, in a second step one would like to learn a representation of the scene which is more abstract than the gray values of the pixels and contains information such as motion and landmarks. Higher cognitive tasks like obstacle detection and self-localization should build on that description of the world and naturally lead to intelligent behavior like specific motor control and collision avoidance.

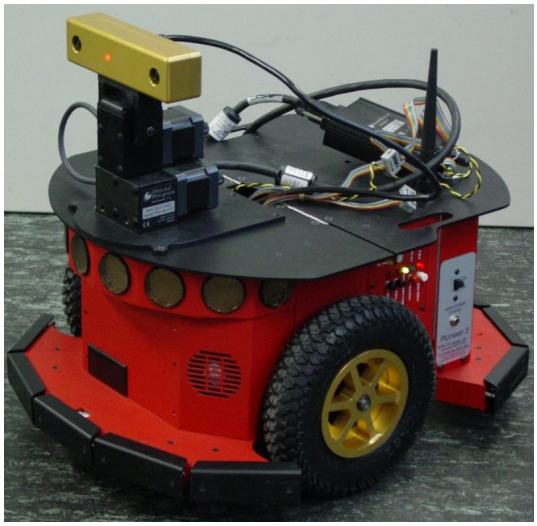


Figure 1.1: The autonomous mobile robot Pioneer 3-DX 8 as manufactured by MOBILEROBOTS Inc (formerly ActivMedia Robotics) is endowed with a stereo camera mounted on a pan-tilt unit.

1.1.1 The robot

Among standard general purpose mobile robots, the Pioneer 3DX built by MOBILEROBOTS Inc[4] as depicted in Figure 1.1 is one of the most widespread platforms currently in use. It is suitable for a large variety of different applications such as research and science as well as security and surveillance. Many different components and accessories like laser, compass, GPS, grippers and sound systems are available.

The present system has a built-in PC with a PIII 850 MHz processor, a 40 GB hard disk and both wireless and Ethernet network adapters. Slopes up to 25 per cent can be handled by the robot, that measures $44\times40\times24$ cm^3 and achieves velocities up to 1.2 $\frac{\text{m}}{\text{s}} \triangleq 4.3\frac{\text{km}}{\text{h}}$ on plain ground. Several sensors provide the system with information:

- Physical contact, e.g. with an obstacle, is measured by bumper panels on the front and on the rear side of the robot with five bumpers each. A pair of 100 g pressure sensors provides the sensory information.
- Distances are metered by a bank of eight ultra-sonic sensors with a field of view comprising 180°. At a rate of 25 Hz ranges reaching from 10 cm to 5 m can be captured.
- Stereo images are supplied by a camera that can be panned and tilted via a serial interface.

A set of 5 batteries allows the robot to independently operate for more than an hour.

[4]See the website `http://www.activrobots.com/` for details on mobile robots and information on Pioneer systems in general.

Figure 1.2: The Bumblebee® stereo camera is produced by POINT GREY Research and grabs color videos at 30 frames per second.

1.1.2 The camera

Two Sony ICX084 color cameras with $\frac{1}{3}$" progressive scan CCD are the main building block for the Bumblebee® 1.2 stereo camera [5].

Videos at a resolution of 2· 640×480 pixels can be grabbed at 30 frames per second for mono and 15 frames per second for stereo images. The "golden brick" weighs approximately 375 g and measures 160×40×50 mm^3. Shutter speed reaches from $\frac{1}{8000}$s to $\frac{1}{30}$s. The device is interfaced by the IEEE-1394 firewire bus which is also the way the power consumption of 2.1 W is satisfied.

There are no major complaints concerning hardware. Problems start if one tries to operate the unit using Linux as operating system. POINT GREY Research seems to favor software development on Windows® XP. Two libraries, namely Digiclops® and Triclops® are provided to access the camera and perform some low-level stereo vision tasks respectively. Both application programming interfaces are written in C/C++ but are only distributed in binary format which obfuscates the underlying mechanisms and require a lot of blind trust on the part of the user. Algorithms regarding disparity and depth maps are not properly documented and are intended to work as black boxes.

However, on Linux systems the camera can directly be operated using the `libdc1394`[6] library and the public domain viewer Coriander [7] which circumvents many of the issues causing trouble.

Together with `libjpeg`[8] it is possible to control the robot via wireless network, grab stereo images at 15 frames per second and compress the stereo images on-line.

[5] More information about the camera and can be found on the corporate website of POINT GREY Research `http://www.ptgrey.com/products/bumblebee/`.

[6] The library is available at `http://sourceforge.net/projects/libdc1394/`.

[7] The software resides at `http://sourceforge.net/projects/coriander/`.

[8] From `http://www.ijg.org/` one can download the actual version 6b originally written in 1998.

1.2 Image Statistics and Unsupervised Learning

Images as we perceive them in natural and artificial environments show some underlying structure. Not all possible configurations of gray valued pixels on a grid have the same chance to be a valid image. One can treat images, independent from the meaning of the scene, as realizations of random processes and characterize properties of these processes. The following section collects some material and some thoughts on statistics of natural images, highlights some relations to current models of visual processing and finally discusses some general principles derived from or used for the analysis of images, that proved to be useful in other domains of machine learning.

1.2.1 Image statistics

In this thesis, the analyzed objects are natural videos. Videos have spatial structure – here in two dimensions: height h and width w. Furthermore there is a temporal component t. In principle, even though images have a finite size in space $\mathcal{H} \times \mathcal{W} = [1..H] \times [1..W]$ and in time $\mathcal{T} = [1 \dots T]$, the first three dimensions can be seen as compact and sampled regions of infinite and continuous domains. Contrary to the first three dimensions, images can additionally be composed of channels, e.g. three or four color channels c (red, green, blue) or two channels s (left, right) for stereo images.

If one fixes the allowable values of images to be discrete $\mathcal{X} = [0, 255]$ or continuous $\mathcal{X} = [0 \dots 1]$ one can understand the set of images as a mapping from an index set $\mathcal{I}$ into the set of gray values $\mathcal{X}$:

$$\mathbf{x} : \mathcal{I} \to \mathcal{X}, \quad \mathcal{I} = \mathcal{H} \times \mathcal{W} \times \mathcal{T} \times \{R, G, B\} \times \{L, R\} \tag{1.1}$$

One single pixel is addressed by a subscript index $\mathbf{t} = [h, w, t, c, s]$:

$$\forall \mathbf{t} \in \mathcal{I} : \quad \mathbf{x}_\mathbf{t} \;=\; \mathbf{x}_{[h,w,t,c,s]} \in \mathcal{X} \tag{1.2}$$

As already mentioned, not all valid configurations of pixels are equally probable. This can be formalized as a random process. The next three paragraphs briefly summarize basic definitions from probability theory.

- The triple $(\Omega, \mathcal{F}, \mathbb{P})$ is an abstract probability space with Ω the set of the outcomes, $\mathcal{F}$ the set of events (the set of Borel measurable subsets $\mathcal{B}(\Omega)$ of Ω) and the probability measure $\mathbb{P}$ assigning a probability to each event. Note that $\mathbb{P}(\Omega) = 1$.

- A random variable is a mapping $X : \Omega \to \mathbb{R}$ such that for any $x \in \mathbb{R}$ the set $X^{-1}((-\infty, x]) := \{\omega \in \Omega | X(\omega) \leq x\}$ is an event. Any random variable X induces a probability measure $\mathbb{P}_X(B) := \mathbb{P}(X^{-1}(B))$ on the Borel σ-algebra of $\mathbb{R}$. The probability distribution of a random variable is given by $F_X(x) := \mathbb{P}(X \leq x) = \mathbb{P}_X((-\infty, x])$ and interval probabilities are expressed as $\mathbb{P}(a < X \leq b) = F_X(b) - F_X(a)$.

- A family of measurable functions $\mathbf{x}_{\mathbf{t}\in\mathcal{I}}$ from $(\Omega, \mathcal{F}) \rightarrow (\mathcal{X}^{\mathcal{I}}, \mathcal{B}(\mathcal{X})^{\otimes\mathcal{I}})$ is a stochastic process with state space $\mathcal{X}$ and index set $\mathcal{I}$.

Given this formal model, one can now collect a corpus of N images and deal with statistics $\phi(\mathbf{x}^1_{[h,w]\in\mathcal{H}\times\mathcal{W}}, \ldots, \mathbf{x}^N_{[h,w]\in\mathcal{H}\times\mathcal{W}})$ to describe the set of images as done in Schaaf [1998]. The simplest possible image statistics would treat each pixel position individually $\phi_{[h,w]}$, e.g. the mean gray value. Assuming translation invariance means $\phi_{[h,w]} = \phi_{[h',w']}$ and reduces the whole image process to a set of identical pixel processes. However, first order statistics capture very little structure because intensities at different positions are correlated. Therefore, one deals with statistics of pairs of pixels $\phi_{[h,w],[h',w']}$, e.g. the gray value covariance.

Some regularities found in natural images f.i. by Ruderman and Bialek [1994] include scale invariance and self similarity. That means, if images are scaled up or down, they show similar behavior. In other words, image patches reveal something about the true structure of images. A review of statistical properties of natural images is provided by Srivastava et al. [2003], where also the non-Gaussian behavior (heavy tails and large curtosis) is stressed. Different models for images either probabilistic ones or models relying on an image manifold are discussed.

Finally, one can also classify images in the frequency domain. In [Torralba and Oliva, 2003], man made environments like city-views or streets and natural environments like fields or beaches are classified in terms of their spectral signature. Man made environments usually contain much more horizontal and vertical structure than natural images.

1.2.2 Relations to visual processing

In primary visual cortex, two main groups of neurons can be discriminated: simple and complex cells. Both types of neurons detect edges or lines. In a nutshell, simple cells respond to bars with a specific orientation and position, whereas complex cells respond to oriented bars but they are invariant to the exact position. The classical model for the output y of these cells given an input image patch (receptive field) $\mathbf{x}$ is written as $y_{\text{simple}} = \mathbf{w}^\top\mathbf{x}$ and $y_{\text{complex}} = \left(\mathbf{w}^\top\mathbf{x}\right)^2 + \left(\mathbf{w}'^\top\mathbf{x}\right)^2$ where the weights $\mathbf{w}$ are Gabor wavelets of different orientations.

Some scientists wondered, according to what principle or underlying fundamental law the weights were adapted. In other words, they tried to establish a link between the observed shapes of receptive fields and statistical properties of images. To state only one example, Hurri [2003] investigates the relations between the statistics of natural stimuli and properties of visual neurons and postulates that there is a strong relation that emerged during evolution.

Another line of thought for explaining the computations in primate visual system is through the notion of invariance or invariant representation [Wiskott, 2003a]. Small changes of the stimulus, e.g. scale of the image can

lead to very significant changes of the retina activity pattern. As our mental representation does not change in response to small shifts or small scalings, somewhere in the visual pathway these invariances have to be extracted. Thus, the study of invariance feature networks can reveal some insight into the computations of the visual system.

1.2.3 Unsupervised learning paradigms

There are four major principles of computation used as candidates to explain self-organization in the visual cortex on the one hand and to unsupervisedly analyze and represent high-dimensional data sets in machine learning. The following overview is taken from [Berkes, 2005b].

- **Compactness:** Efficient coding whilst preserving as much structure i.e. variance as possible can, in the linear case, be accomplished by PCA. Minimum description length means in this context, that likely data points can be represented by a small number of components whilst minimizing reconstruction error. As a consequence, redundancy in the data is eliminated or at least significantly reduced.

- **Independence:** According to the principle of independence, a signal is represented as a linear mixture of sources. In a probabilistic setting, the sources are selected such that they are statistically as independent as possible. Redundancy in the data is modeled, not eliminated. The standard review of ICA was published by Hyvärinen [1999]. Independence as detailed in [Hyvärinen and Hoyer, 2000] leads to color selective simple cells. If one takes videos of TV broadcasts instead of images as learning corpus as done by Hateren and Ruderman [1998], spatio-temporal linear filters emerge, selective to orientation and direction of movement. In addition, the filters are spatio-temporarily localized and responsive to different spatio-temporal scales.

- **Sparseness:** Due to Olshausen and Field [1996], simple cell properties also emerge if the data is represented in a sparse way. Sparseness is equivalent to a scenario where only a few cells respond at the same time. Sparse codes are known to decrease the signal-to-noise ratio. Sparseness means non-Gaussianity which implies equivalence of independence and sparseness up to a certain extent.

- **Slowness:** The paradigm of temporal coherence consists of finding transformation showing a smooth temporal behavior. The SFA algorithm introduced by Wiskott and Sejnowski [2002] is a powerful tool to extract invariant features. A nice application to an object recognition task is proposed in [Einhäuser et al., 2005] where viewpoint invariant representations of artificial objects are learned.

It is also possible to combine the mentioned principles even if they represent contrary objectives. For example, Blaschke [2005] profoundly relates

ICA and SFA as special cases of each other and combines the two methods to a powerful source separation algorithm. An interesting framework termed Bubbles and proposed by Hyvärinen et al. [2003] unifies sparseness, temporal coherence and topographic coherence. Temporal activity bubbles are obtained by integration of sparseness and temporal coherence. Combination of topographic coherence i.e. spatial smoothness and sparseness leads to spatially limited "blobs" on the topographic grid. Further adding temporal coherence leads to spatially and temporally located bubbles of activity.

1.3 Outline of the thesis

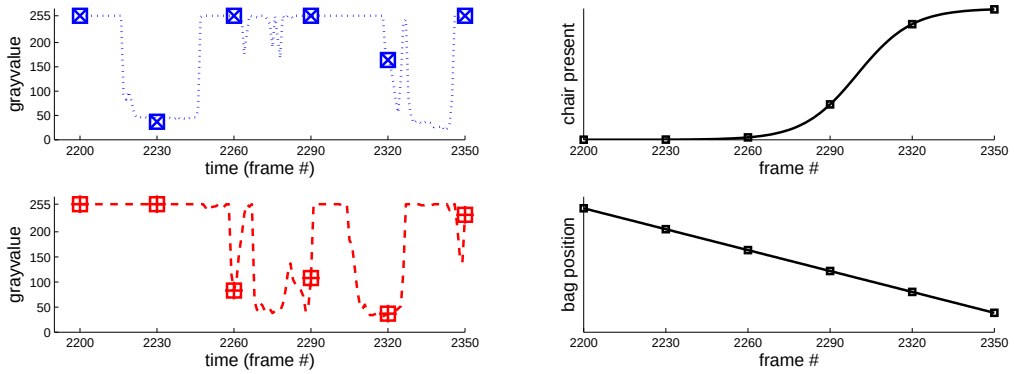
The thesis at hand comprises five chapters and an appendix with some additional material. At the beginning, the context of the project is sketched and an overview of the work is given. Subsequently, in the theoretical chapter, considerations for linear and nonlinear processing are given and the question of empirical measurement is discussed. Concrete details of the implementation can be found in the third chapter - the practical computer science part of the thesis. The next chapter is dedicated to empirical simulations conducted to determine optimal parameter settings and to analyze and interpret the obtained results. Finally, in chapter five, a summary is provided, a conclusion is drawn and some future thoughts are described.

Chapter 2

Slow Feature Analysis

2.1 Temporal coherence as learning principle

(a) Video recorded in lab, frames $\{2200, 2230, \ldots, 2350\}$



(b) Sensor variation

(c) Internal representation

Figure 2.1: The learning principle of temporal coherence is illustrated comparing the quick variations of the pixels of an input video to the slow variation of a more abstract mental representation. According to the principle, meaningful representations should exhibit a temporally coherent or slowly varying structure.

In perceptual tasks like vision, the following observations can be made. If one considers a video like shown in Figure 2.1(a) and if one wants to extract a meaningful description of the scene based on single pixel values, it is apparent that the sensory signals (2.1(b)) vary on short time scales whereas an abstract scene representation (2.1(c)) varies on much longer time scales.

Shifting the whole frame (or the camera) one single pixel would almost change nothing in the abstract representation of the scene, but the gray values of the pixels can significantly change. Therefore, the representation of the scene should be invariant to small local transformations (like translation, rotation or zoom) which cannot be required for the primary sensor signals.

Thus the intuition behind the unsupervised learning paradigm of temporal coherence can be summarized as follows: In order to unsupervisedly extract a representation of the scene, find output signals exhibiting a temporal change which is as smooth as possible given the quickly varying sensory input.

Practical implementations of the principle of temporal coherence were firstly presented by Földiák [1991] who introduced a modified Hebbian learning rule leading to a network that is able to exhibit shift invariance. Subsequently, this work was extended in [Stone and Bray, 1995] and [Stone, 1996]. The iterative minimization of an objective function based on the ratio between a short-term variation measure and a long-term variation measure lead to a combination of a Hebbian and an Anti-Hebbian learning rule which is able to extract disparity information from stereo images.

A neural network operating on natural videos recorded by a camera mounted on a cat's head was proposed by Kayser et al. [2001]. The objective function used here is a sum of squared temporal derivatives that was optimized using gradient based methods. Filters showing properties of complex cells were obtained as result.

Another approach to solve the learning problem written as a constrained optimization was introduced in [Wiskott and Sejnowski, 2002]. The resulting algorithm – termed Slow Feature Analysis – implements a one shoot learning which is guaranteed to find the global optimum and does not need any gradient descent but is based on an eigenvalue problem instead.

2.2 Slow Feature Analysis

2.2.1 Definitions

For completeness, some rather common notations are summarized in the following section. The usual Kronecker indicator function which is zero for $i \neq j$ and equals one for $i = j$ is given by δ_{ij}. Furthermore, the expectation of a time-continuous signal $\mathbf{x}_t \in \mathbb{R}^N$ over the period $[0, T]$ is denoted by $\langle \mathbf{x}_t \rangle := \frac{1}{T} \int_0^T \mathbf{x}_t \mathrm{d}t$. The covariance of two signals of length T along with its unbiased empirical estimator for the time-discrete case is defined as:

$$\mathbf{C}_{\mathbf{x},\mathbf{y}} = \mathrm{cov}(\mathbf{x}_t, \mathbf{y}_t) := \left\langle \mathbf{x}_t \mathbf{y}_t^\top - \langle \mathbf{x}_t \rangle \langle \mathbf{y}_t \rangle^\top \right\rangle \tag{2.1}$$

$$\hat{\mathbf{C}}_{\mathbf{x},\mathbf{y}} = \hat{\mathrm{cov}}(\mathbf{x}_t, \mathbf{y}_t) := \frac{1}{T-1} \mathbf{X}\mathbf{Y}^\top - \frac{1}{T^2} (\mathbf{X}\mathbb{1})(\mathbf{Y}\mathbb{1})^\top \tag{2.2}$$

Covariance of a signal $\mathbf{x}_t$ against a time-shifted version of itself is termed autocovariance.

$$\mathbf{C}_{\mathbf{x},\Delta t} := \mathrm{cov}(\mathbf{x}_t, \mathbf{x}_{t+\Delta t}) \tag{2.3}$$
$$\hat{\mathbf{C}}_{\mathbf{x},\Delta t} := \hat{\mathrm{cov}}(\mathbf{x}_t, \mathbf{x}_{t+\Delta t}) \tag{2.4}$$

Without time-shift, one obtains the symmetric covariance matrix of the signal $\mathbf{x}_t$.

$$\mathbf{C}_{\mathbf{x}} = \mathrm{cov}(\mathbf{x}_t) := \mathrm{cov}(\mathbf{x}_t, \mathbf{x}_t) \tag{2.5}$$
$$\hat{\mathbf{C}}_{\mathbf{x}} = \hat{\mathrm{cov}}(\mathbf{x}_t) := \hat{\mathrm{cov}}(\mathbf{x}_t, \mathbf{x}_t) \tag{2.6}$$

Here, $\mathbb{1}$ denotes a vector containing T times the element 1, $\mathbf{I}$ is the unit matrix and $\mathbf{X} \in \mathbb{R}^{N\times T}$ contains all data points arranged as column vector and is called the data matrix. A version of the data matrix containing the same vectors as $\mathbf{X}$ but shifted Δt in time is denoted $\mathbf{X}_{\Delta t}$.

2.2.2 Original problem statement

The learning task of SFA as originally formulated by Wiskott and Sejnowski [2002] is the following:

Given an input time series $\mathbf{x}_t \in \mathbb{R}^N$, $t \in [0, T]$, find a set of K real-valued instantaneous functions $g_1(\mathbf{x}), \ldots, g_K(\mathbf{x}) \in \mathcal{F}$ that generate the output time series $\mathbf{y}_t = \mathbf{g}(\mathbf{x}_t)$ such that

$$\text{slowness}_j = s(y_{j,t}) := \left\langle \dot{y}_{j,t}^2 \right\rangle \stackrel{!}{=} \text{ min.} \tag{2.7}$$

is minimized under the constraints of

$$\forall j \ \langle y_{j,t} \rangle = 0 \quad \text{zero mean,} \tag{2.8}$$
$$\forall j \ \left\langle y_{j,t}^2 \right\rangle = 1 \quad \text{unit variance,} \tag{2.9}$$
$$\forall i < j \ \langle y_{i,t} y_{j,t} \rangle = 0 \quad \text{decorrelation and order.} \tag{2.10}$$

SFA differs fundamentally from simple-low pass filtering in temporal direction. As the functions g_j have only instantaneous scope, that is they map one single input $\mathbf{x}_t$ at a certain time t to an output $y_{j,t}$ at the same time. The primary objective is to reach temporal smoothness but under the strong limitation of instantaneous processing. The additional constraints to the optimization problem make sure that trivial solutions are excluded. The unit variance prevents constant signals to emerge, the decorrelation constraint enforces distinctness and the zero mean constraint is introduced for convenience only.

Uniqueness of the solution is guaranteed when the functions are extracted one after another such that g_1 is the slowest function in $\mathcal{F}$ and g_j is the slowest function in $\mathcal{F}$ that produces an output signal $y_{j,t}$ which is decorrelated to all signals $y_{i,t}$ for $1 \leq i < j$.

Temporal variation is measured by the squared first derivative, which can be approximated by a finite difference.

$$\dot{y}_{j,t} := \lim_{\Delta t \to 0} \frac{y_{j,t+\Delta t} - y_{j,t}}{\Delta t} \overset{\Delta t=1}{\approx} y_{j,t+1} - y_{j,t} \tag{2.11}$$

In the case of linear functions $g_j(\mathbf{x}_t) = \mathbf{w}_j^T (\mathbf{x}_t - \langle \mathbf{x}_t \rangle)$ with zero mean input signals (to respect constraint (2.8)), the following calculation can be made:

$$\begin{aligned} s(y_{j,t}) &= \left\langle \dot{y}_{j,t}^2 \right\rangle = \left\langle (y_{j,t+1} - y_{j,t})^2 \right\rangle = \left\langle \left(\mathbf{w}_j^T \dot{\mathbf{x}}_t \right)^2 \right\rangle \\ &= \mathbf{w}_j^\top \mathrm{cov}(\dot{\mathbf{x}}_t) \mathbf{w}_j =: \mathbf{w}_j^\top \mathbf{A} \mathbf{w}_j \qquad (2.12) \\ \langle y_{i,t} \cdot y_{j,t} \rangle &= \left\langle \mathbf{w}_i^\top (\mathbf{x}_t - \langle \mathbf{x}_t \rangle) \cdot \mathbf{w}_j^\top (\mathbf{x}_t - \langle \mathbf{x}_t \rangle) \right\rangle \\ &= \mathbf{w}_i^\top \mathrm{cov}(\mathbf{x}_t) \mathbf{w}_j =: \mathbf{w}_i^\top \mathbf{B} \mathbf{w}_j \qquad (2.13) \end{aligned}$$

Solving the generalized eigenvalue problem given by

$$\mathbf{AW} = \mathbf{BW}\mathbf{\Lambda} \tag{2.14}$$

directly yields the optimal weights $\mathbf{w}_{1..K}$ as the eigenvectors corresponding to the smallest eigenvalues $\lambda_{1..K}$. Normalization to $\mathbf{w}_i^\top \mathbf{B} \mathbf{w}_j = \delta_{ij}$ fulfills constraints (2.9) and (2.10). As $\mathbf{A}$ and $\mathbf{B}$ are second order statistics they are positive semidefinite and symmetric and thus all eigenvalues are real and greater than or equal to zero. The slowness for each function g_j is given by $s(y_{j,t}) = \lambda_j$ and the slowness of the K components is $s(\mathbf{y}) = \sum_{j=1}^K \lambda_j$.

2.2.3 Optimal free responses

The question of optimal slow signals given a function g_j of arbitrary complexity that in the limit does not depend on the input signal $\mathbf{x}_t$ anymore is further examined in [Wiskott, 2003b]. Without any constraints on the function space $\mathcal{F}$ and without considering the input $(\mathbf{x}_t)_{t \in [0,T]}$, the slowest possible functions (without boundary conditions) that can be found, take the following form:

$$y_{j,t} = \sqrt{2} \cos\left(\frac{j\pi}{T} t \right) \tag{2.15}$$

The slowest signal $y_{1,t}$ is half a cosine in $[0, T]$, the second signal would be a full cosine, the third would correspond to one and a half cosine and so on. These results are obtained by using two different approaches. One is based on the calculus of variations. Another more algebraic approach is based on a finite basis for the optimal slowness functions. The optimal slow signals naturally fulfill the constraints (2.8-2.10).

2.2.4 Problem in matrix notation

As the images acquired by the robot camera are multidimensional and the present thesis is interested in several slow components, the problem can be written in a more convenient way using matrix notation.

$$\text{tr}\left\langle \dot{\mathbf{y}}_t \dot{\mathbf{y}}_t^\top \right\rangle = \left\langle \dot{\mathbf{y}}_t^\top \dot{\mathbf{y}}_t \right\rangle \stackrel{!}{=} \text{ min.} \quad \text{s. t.} \quad \left\langle \mathbf{y}_t \mathbf{y}_t^\top \right\rangle = \mathbf{I} \text{ and } \left\langle \mathbf{y}_t \right\rangle = \mathbf{0} \tag{2.16}$$

Again, using linear functions $\mathbf{g} : \mathbb{R}^N \to \mathbb{R}^K,\ \mathbf{x}_t \mapsto \mathbf{y}_t = \mathbf{W}^\top \mathbf{x}_t$ with $\mathbf{W} \in \mathbb{R}^{N \times K}$ and assuming zero mean signals, the constrained minimization leading to the generalized eigenproblem reads as follows:

$$\text{tr}(\mathbf{W}^\top \mathbf{A} \mathbf{W}) \stackrel{!}{=} \text{ min. s. t. } \mathbf{W}^\top \mathbf{B} \mathbf{W} = \mathbf{I} \tag{2.17}$$

The above formulation is only determined up to a rotation of the rows of $\mathbf{W}$, i.e. one can also use $\tilde{\mathbf{W}} = \mathbf{W}\mathbf{R}$ with orthogonal $\mathbf{R} \in \mathbb{R}^{K \times K}$. If additionally, the covariance of the derivatives $\mathbf{W}^\top \mathbf{A} \mathbf{W}$ is required to be diagonal which corresponds to decorrelated derivatives, uniqueness can be guaranteed. Another way of solving the optimization taken from [Wiskott and Sejnowski, 2002] is based on twofold PCA and can deal with covariance matrices $\mathbf{B}$ that are not strictly positive definite.

The first PCA calculates the eigenvalue decomposition $\mathbf{B} = \mathbf{U} \mathbf{\Lambda} \mathbf{U}^\top$ and keeps only non-singular dimensions with eigenvalues above a certain threshold τ which is summarized in the coordinate transform $\mathbf{P} := \mathbf{U}_{[:,\lambda>\tau]} \mathbf{\Lambda}_{[\lambda>\tau]} = [\lambda_1 \mathbf{u}_1, \ldots, \lambda_R \mathbf{u}_R]$ where only the $R \geq K$ columns of the eigenvector matrix $\mathbf{U}$ are kept whose eigenvalues are above the threshold. This procedure whitens the covariance matrix $\mathbf{B}$ which implies that the constraints from (2.16) are fulfilled. Any weight matrix of the form $\mathbf{W} = \mathbf{P}\mathbf{R}$ with $\mathbf{R} \in \mathbb{R}^{R \times R}$ an orthonormal matrix also respects the constraints. The weights are determined only up to rotation. In order to minimize the objective function, the following minimization problem has to be solved

$$\text{tr}(\mathbf{R}^\top \mathbf{P}^\top \mathbf{A} \mathbf{P} \mathbf{R}) \stackrel{!}{=} \text{ min. s.t. } \mathbf{R}^\top \mathbf{R} = \mathbf{I} \tag{2.18}$$

The second PCA calculates the eigenvalue decomposition of $\mathbf{P}^\top \mathbf{A} \mathbf{P}$, and the eigenvectors corresponding to the K smallest eigenvalues yield the required rotation matrix $\mathbf{R}$.

2.2.5 Relations to other methods

SFA and BSS

The term Blind Source Separation (BSS) refers to the task to recover N sources s_i from M observations $x_j = \sum_i a_{ji} s_i$ of linear combinations of the sources. The task is equivalent to finding a demixing matrix $\mathbf{D}$ such that $\mathbf{y} = \mathbf{D}\mathbf{x} \approx \mathbf{s}$. Many BSS techniques do not take the temporal structure of the signal into account.

However, there are approaches for separation using time-delayed covariance matrices. The classical idea from Molgedey and Schuster [1994] searches to jointly diagonalize the covariance matrix $\mathbf{C_x}$ and a time-shifted covariance matrix $\mathbf{C}_{\mathbf{x},\Delta t}$ where the time-shift Δt has to be carefully selected. Proposed in [Ziehe and Müller, 1998], the TDSEP algorithm overcomes the difficulty in selecting an appropriate Δt value be jointly diagonalizing several time-shifts simultaneously.

Assuming a zero mean signal $\mathbf{x}_t$ with temporal derivatives approximated by finite differences, the covariance matrix $\mathbf{A}$ of the derivatives can equivalently be written as

$$\begin{aligned}
\mathbf{B} &= \left\langle \mathbf{x}_t \mathbf{x}_t^\top \right\rangle =: \mathbf{C}_0 && (2.19)\\
\mathbf{A} &= \left\langle \dot{\mathbf{x}} \dot{\mathbf{x}}^\top \right\rangle = \left\langle (\mathbf{x}_{t+1} - \mathbf{x}_t) (\mathbf{x}_{t+1} - \mathbf{x}_t)^\top \right\rangle \\
&= \left\langle \mathbf{x}_{t+1} \mathbf{x}_{t+1}^\top \right\rangle + \left\langle \mathbf{x}_t \mathbf{x}_t^\top \right\rangle - \left\langle \mathbf{x}_t \mathbf{x}_{t+1}^\top \right\rangle - \left\langle \mathbf{x}_{t+1} \mathbf{x}_t^\top \right\rangle \\
&\overset{(2.19)}{=} 2\mathbf{C}_0 - \left(\mathbf{C}_{\mathbf{x},1} + \mathbf{C}_{\mathbf{x},1}^\top \right) =: 2\mathbf{C}_0 - \mathbf{C}_1 && (2.20)
\end{aligned}$$

Using that somewhat different notation, the optimization problem (2.17) now reads:

$$\text{tr}(\tilde{\mathbf{W}}^\top \mathbf{C}_1 \tilde{\mathbf{W}}) \overset{!}{=} \text{ max. s. t. } \tilde{\mathbf{W}}^\top \mathbf{C}_0 \tilde{\mathbf{W}} = \mathbf{I} \tag{2.21}$$

Solving the corresponding generalized eigenproblem $\mathbf{C}_1 \tilde{\mathbf{W}} = \mathbf{C}_0 \tilde{\mathbf{W}} \tilde{\mathbf{\Lambda}}$ by the assignment $[\tilde{\mathbf{W}}, \tilde{\mathbf{\Lambda}}] \leftarrow \texttt{eig}(\mathbf{C}_1, \mathbf{C}_0)$ and using the facts (2.19)+(2.20) yields $\mathbf{AW} = \mathbf{BW\Lambda} \Leftrightarrow \mathbf{C}_1\mathbf{W} = \mathbf{C}_0\mathbf{W}(2\mathbf{I} - \mathbf{\Lambda})$. Thus, the two eigenvalue problems $\mathbf{AW} = \mathbf{BW\Lambda}$ and $\mathbf{C}_1 \tilde{\mathbf{W}} = \mathbf{C}_0 \tilde{\mathbf{W}} \tilde{\mathbf{\Lambda}}$ are equivalent by $\tilde{\mathbf{W}} = \mathbf{W}$ and $\tilde{\mathbf{\Lambda}} = 2\mathbf{I} - \mathbf{\Lambda}$.

As in BSS a whitening constraint is present in SFA. But instead of minimizing off-diagonal elements, diagonal elements are maximized with respect to $\mathbf{W}$. In the case of linear functions SFA is equivalent to joint diagonalization of the covariance matrix $\mathbf{C}_0$ and the time shifted covariance matrix $\mathbf{C}_1$ and thus becomes a special method of BSS.

SFA and ICA

As pointed out in [Blaschke, 2005] SFA and Independent Component Analysis (ICA) can be related in the following way.

At first sight, ICA extracts linear features that are as statistically independent as possible whereas SFA extracts uncorrelated nonlinear features that have small temporal variations. The paradigms for the two algorithms can even be mutually exclusive because slow signals tend to be less independent. Notwithstanding, linear SFA and ICA based on second moment statistics are equivalent.

The algorithms can also be combined to Independent Slow Feature Analysis as done by Blaschke and Wiskott [2004].

2.2.6 Probabilistic interpretation for SFA

In a recent article Turner and Sahani [2006] show that maximum likelihood learning in a linear Gaussian state-space with Markovian prior is equivalent to SFA. Building on that insight they suggest some extensions to SFA and provide a probabilistic context for SFA. In a nutshell, the model has the following form:

$$\begin{aligned} p(\mathbf{y}_t|\mathbf{y}_{t-1}, \mathbf{\Lambda}, \mathbf{\Sigma}) &= \mathcal{N}(\mathbf{\Lambda}\mathbf{y}_{t-1}, \mathbf{\Sigma}) \\ p(\mathbf{y}_1|\mathbf{\Sigma}_1) &= \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_1) \end{aligned} \tag{2.22}$$

Here $\mathbf{\Sigma}_1$, $\mathbf{\Sigma}$, and $\mathbf{\Lambda}$ are diagonal matrices containing the initial variance, the process variance and the correlation strengths respectively.

2.3 Expanded Slow Feature Analysis

SFA can be extended to non-linear functions $g_1, \ldots, g_K \in \mathcal{F}$ by introducing an explicit expansion mapping $\mathbf{\Phi} : \mathbb{R}^N \to \mathbb{R}^M$ of the data into some high-dimensional feature space of dimension $M \gg N$. The expansion mapping $\mathbf{\Phi}$ is composed of basis functions $(\phi_1, \ldots, \phi_M)^\top$ of the feature space $\mathcal{F}$ and the functions $g_1, \ldots, g_K$ would be linear combinations such that the resulting slowness function $\mathbf{g} : \mathbb{R}^N \to \mathbb{R}^K$ is given by

$$\mathbf{g}(\mathbf{x}_t) = \mathbf{W}^\top (\mathbf{\Phi}(\mathbf{x}_t) - \mathbf{\Phi}_0) \tag{2.23}$$

where $\mathbf{\Phi}_0 := \langle \mathbf{\Phi}(\mathbf{x}_t) \rangle$ is the temporal mean of the data in feature space and $\mathbf{W} \in \mathbb{R}$ is a matrix containing the weights for the linear combinations. The mean in feature space has to be subtracted to respect the zero mean constraint from (2.8). Solving the optimization (2.16) using $\mathbf{A} := \mathrm{cov}(\dot{\mathbf{\Phi}}_t)$ and $\mathbf{B} := \mathrm{cov}(\mathbf{\Phi}_t)$ with $\mathbf{\Phi}_t := \mathbf{\Phi}(\mathbf{x}_t)$ and $\dot{\mathbf{\Phi}}_t := \mathbf{\Phi}_{t+1} - \mathbf{\Phi}_t$ yields the weights $\mathbf{W}$.

This strategy clearly suffers from the curse of dimensionality as the matrices $\mathbf{A}$, $\mathbf{B} \in \mathbb{R}^{M \times M}$ quickly become unfeasible with growing M. Taking the space of polynomials of degree $d = 2$ as an example, one has

$$\mathbf{\Phi}_t = (x_1, .., x_N, x_1^2, .., x_1 x_N, x_2^2, .., x_N^2)_t^\top$$

and $M = N + \frac{N(N-1)}{2}$ which yields already for reasonable inputs of size $N = 150$ matrices $\mathbf{A}$ and $\mathbf{B}$ of size 11.325×11.325. For a polynomial expansion of order d the feature space has a dimension M which is exponential in d.

$$M = \sum_{k=1}^{d} \binom{N+k-1}{k} \in \mathcal{O}(e^d)$$

If a higher degree of non-linearity is desired, one can iteratively apply expanded SFA and use $\mathbf{y}_t := \mathbf{g}(\mathbf{x}_t)$ from (2.23) as input for another layer of SFA and obtain functions of the form $\mathbf{h}(\mathbf{y}_t) = \mathbf{V}^\top (\mathbf{\Phi}(\mathbf{y}_t) - \mathbf{\Phi}_0)$ with the same expansion as in the previous layer and new weights $\mathbf{V}$. (see Section 2.6)

2.4 Applications to Expanded SFA

2.4.1 Receptive fields

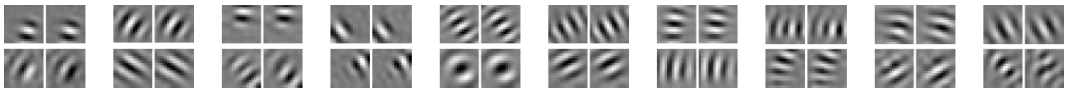


Figure 2.2: In [Berkes and Wiskott, 2005] SFA with polynomial expansion was done by using pairs of temporally neighboring patches of still images that were artificially transformed to simulate a temporal structure. The resulting filters have many properties of receptive fields.
The figure shows the stimuli that lead to the maximal filter output (upper line) and minimal filter output (lower line) respectively.

SFA is an unsupervised learning paradigm that is designed to extract invariances. In the context of vision, complex cells show similar responses to the same stimulus in different phases, which can be thought as phase invariance. Clearly, this cannot be accomplished by a linear filter.

Patches sampled from a collection of still images were used in [Berkes and Wiskott, 2005] to investigate the properties of filters maximizing temporal coherence. It turned out that the learned filters were in many respects similar to complex cells (see Figure 2.2). Transformations, namely translation, scaling and rotation, were applied to the patches which generated an artificial temporal structure. A slight modification was made as the input sequence itself consisted of data points containing two successive patches what makes the resulting filter function not instantaneous anymore.

Non-linearities in the filter included polynomials of degree 2, which lead to filters exhibiting invariances to different transformation like phase shift, position change, size change, frequency change, orientation change and change in curvature. A detailed analysis of the invariances is given in [Berkes and Wiskott, 2006].

2.4.2 Driving forces of time series

In [Wiskott, 2003c] SFA was used to discover slowly varying features in quickly varying nonstationary time series. On a set of toy examples the extraction capabilities in terms of accuracy of the analysis are studied.

2.4.3 Digit classification

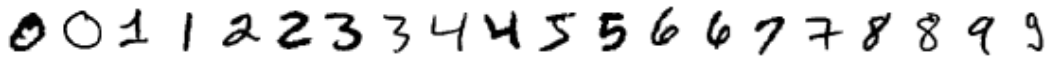


Figure 2.3: A sample of digits taken from the MNIST database of handwritten digits.

Another interesting application to SFA is supervised classification as done on the famous MNIST digit dataset[1] (see figure 2.3) in [Berkes, 2005a]. Training examples in classification do not necessarily have temporal structure, thus the covariance of the temporal derivatives is calculated in a slightly different way. Many time series containing just two patterns of the same class are used to calculate differences. Feature detectors invariant to the transformation from one object to another object of the same class would be the optimal slowest functions. The temporal average is replaced by an average over different series. The training images have a size of 28×28 pixels; their dimensionality is reduced by PCA from $28^2 = 784$ to 35. Cubic Polynomials reach an error rate of 1.5% on the test set which is comparable to other standard algorithms.

2.5 Kernelized Slow Feature Analysis

Another approach for non-linearization makes use of the fact that SFA is completely based on second order statistics. Thus, SFA can be kernelized in line with the extension of PCA to Kernel-PCA by Schölkopf et al. [1998].

A kernel-based implementation of the temporal slowness principle was firstly presented by Bray and Martinez [2002]. However the objective function used here is the objective function of Stone [1996] which is somewhat different from SFA.

A central building block is the concept of a decaying temporal average. This kind of average tends to forget older observations due to the exponential decay implemented by λ. Only the fraction λ (usually close to 1) of the average is kept, the rest is updated using a new observation. Note that if λ_t increases with time according to $\lambda_t \leftarrow \frac{t-1}{t}$, the final estimate $\mathbf{m}_T$ will equal the usual mean $\frac{1}{T}\sum_{t=1}^{T}\mathbf{x}_t$.

$$\mathbf{m}_1^\lambda = \mathbf{x}_1, \quad \mathbf{m}_{t+1}^\lambda = \lambda\mathbf{m}_t^\lambda + (1-\lambda)\mathbf{x}_t, \quad 0 \le \lambda \le 1 \tag{2.24}$$

As a second step, short-term covariances $\mathbf{C}_S$ are calculated based on short-term averages and long-term covariances $\mathbf{C}_L$ based on long-term averages. There is no variable parameter λ_t that allows for exact correspondence to the usual covariance.

$$\mathbf{C}^\lambda = \frac{1}{T}\sum_{t=1}^{T}\left(\mathbf{x}_t - \mathbf{m}_t^\lambda\right)\left(\mathbf{x}_t - \mathbf{m}_t^\lambda\right)^\top \tag{2.25}$$

The only difference between $\mathbf{C}_S$ and $\mathbf{C}_L$ is the decay parameter λ which is adjusted such that the half-life of $\mathbf{C}_S$ is around 100 times shorter than that of $\mathbf{C}_L$.

Concluding, the eigenvalue problem $\mathbf{AW} = \mathbf{BW\Lambda}$ of SFA is based on the covariance of the data $\mathbf{B}$ and on the covariance of the derivatives $\mathbf{A}$

[1]The MNIST database of handwritten digits contains 60,000 training examples of handwritten digits and is available at `http://yann.lecun.com/exdb/mnist/`.

whereas Bray and Martinez consider the problem $\mathbf{C}_S\mathbf{W} = \mathbf{C}_L\mathbf{W}\mathbf{\Lambda}$ with short-term covariance $\mathbf{C}_S$ and long-term covariance $\mathbf{C}_L$.

2.5.1 The kernel trick

By the use of an appropriate kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ the feature space can be defined implicitly. We limit ourselves to $\mathcal{X} = \mathbb{R}^N$ in the following. If k is symmetric, continuous and positive definite i.e. for any set $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subseteq \mathcal{X}$ the matrix $\mathbf{K}_{ij} := k(\mathbf{x}_i, \mathbf{x}_j) \in \mathbb{R}^{n \times n}$ has only strictly positive eigenvalues, then k is said to be a Mercer kernel and one has the following property: There exists a Hilbert space $\mathcal{H}_k$ and a feature map $\mathbf{\Phi} : \mathcal{X} \to \mathcal{H}_k$ such that

$$k(\mathbf{x}, \mathbf{z}) = \langle \mathbf{\Phi}(\mathbf{x}), \mathbf{\Phi}(\mathbf{z}) \rangle_k$$

is an inner product.

Using the feature map $\mathbf{\Phi}(\mathbf{x}) := k(\mathbf{x}, \cdot)$ one obtains the reproducing property.

$$\forall f \in \mathcal{H}_k \qquad f(\mathbf{z}) = \langle f(\cdot), k(\mathbf{z}, \cdot) \rangle_k$$

Furthermore, the kernel k can be orthogonally diagonalized

$$k(\mathbf{x}, \mathbf{z}) = \sum_{j=1}^{\infty} \lambda_j \psi_j(\mathbf{x}) \psi_j(\mathbf{z})$$

with $\langle \psi_i(\cdot), \psi_j(\cdot) \rangle_k = \delta_{ij}$ *and* $\lambda_1 \geq \lambda_2 \geq \cdots \geq 0$ yielding the feature map $\mathbf{\Psi}(\mathbf{x}) := \sum_{j=1}^{\infty} \sqrt{\lambda_j} \psi_j(\mathbf{x})$ which also fulfills $\langle \mathbf{\Psi}(\mathbf{x}), \mathbf{\Psi}(\mathbf{z}) \rangle_k = k(\mathbf{x}, \mathbf{z})$.

Concluding one can say that a kernel function k implicitly defines a feature space on the one hand in terms of the sequence of its eigenfunctions $\sqrt{\lambda_j}\phi_j(\cdot)$ and, on the other hand in terms of the sequence of its partial evaluations $k(\mathbf{z}, \cdot),\ \mathbf{z} \in \mathcal{Z} \subset \mathcal{X}$ for a set $\mathcal{Z}$ of support vectors. These feature spaces can have very large or even infinite dimension. That means one has an appropriate method to calculate inner products in high-dimensional feature spaces implicitly by means of a kernel function.

2.5.2 Kernelization of SFA

We now meet the challenge to find slow functions $\mathbf{g}(\mathbf{x}_t) = \mathbf{U}^\top (\mathbf{\Phi}(\mathbf{x}_t) - \mathbf{\Phi}_0)$ where the feature space, normally given by the mapping $\mathbf{\Phi}$, will implicitly be defined by a kernel k. Again $\mathbf{\Phi}_0 := \frac{1}{T} \sum_{t=1}^{T} \mathbf{\Phi}(\mathbf{x}_t)$ is the empirically estimated temporal mean in feature space, which is subtracted to respect (2.8). The columns of $\mathbf{U}$ live in feature space and can be represented as linear combinations of a set of support vectors $\mathbf{z}_i$ mapped into feature space.

$$\mathbf{u}_j \quad = \quad \sum_{i=1}^{M} w_{ij} \mathbf{\Phi}(\mathbf{z}_i) \tag{2.26}$$

The components of g can be written by using the kernel function as:

$$\begin{aligned} g_j(\mathbf{x}_t) &= \langle \mathbf{u}_j, \mathbf{\Phi}(\mathbf{x}_t) - \mathbf{\Phi}_0 \rangle \overset{(2.26)}{=} \left\langle \sum_{i=1}^{M} w_{ij} \mathbf{\Phi}(\mathbf{z}_i), \mathbf{\Phi}(\mathbf{x}_t) - \mathbf{\Phi}_0 \right\rangle \\ &= \sum_{i=1}^{M} w_{ij} \langle \mathbf{\Phi}(\mathbf{z}_i), \mathbf{\Phi}(\mathbf{x}_t) \rangle - \frac{1}{T} \sum_t \sum_{i=1}^{M} w_{ij} \langle \mathbf{\Phi}(\mathbf{z}_i), \mathbf{\Phi}(\mathbf{x}_t) \rangle \\ &= \sum_{i=1}^{M} w_{ij} \left(k(\mathbf{z}_i, \mathbf{x}_t) - \frac{1}{T} \sum_t k(\mathbf{z}_i, \mathbf{x}_t) \right) \end{aligned} \tag{2.27}$$

Finally, the resulting slowness function in matrix notation reads:

$$\mathbf{g}(\mathbf{x}_t) = \mathbf{W}^\top (\mathbf{k}(\mathbf{x}_t) - \mathbf{k}_0) \tag{2.28}$$

Numerically, the solution is found as follows. Given a set of support vectors $\{\mathbf{z}_i\}_{i=1..M}$ and using the kernel expansion $\mathbf{k}_t = [k(\mathbf{z}_1, \mathbf{x}_t), .., k(\mathbf{z}_M, \mathbf{x}_t)]^\top$, the problem becomes equivalent to expanded SFA.

The weight matrix $\mathbf{W}$ solving the generalized eigenproblem (2.14) with $\mathbf{A} := \mathrm{cov}(\dot{\mathbf{k}}_t)$, $\mathbf{B} := \mathrm{cov}(\mathbf{k}_t)$ is calculated in the usual way. The resulting functions g_j with $j = 1..K$ are then determined by $\mathbf{W}$.

The computational complexity of the method is given by the number of support vectors M and not by the dimension of the feature space anymore. This is the usual effect of the kernel trick which trades dimension of feature space against number of data points. As a consequence, in order to reduce complexity, the number of support vectors M should be kept as small as possible.

In contrast to support vector machines, it is not obvious how the support vector set should be chosen (more details are given in Section 4.3.1).

2.6 Multilayer Slow Feature Analysis

It can be useful to apply several SFA processing steps for several reasons. In the first place the degree of non-linearity can increase, e.g. expanded SFA with polynomials of degree 2 applied two times leads to polynomials of degree 4 at the output. Secondly, as depicted in Figure 2.4 on page 20, the size of the receptive field can grow in every layer such that the spatial support of the function gets larger in every layer. This corresponds to pyramidal data structures in image processing.

For the notion of receptive fields to be meaningful, the signal $\mathbf{x_t} = \mathbf{x}_{[t,s]}$ now has temporal and spatial structure indexed by t and s respectively. In the case of images, one has at least height and width being part of the index s. The notion of receptive field can be extended to channels such as stereo or color channels. The output of layer 0 will be the input to layer 1. This is written as:

$$\mathbf{y}^0_{[t,s]} := \mathbf{x}_{[t,s]} = \mathbf{x_t}$$

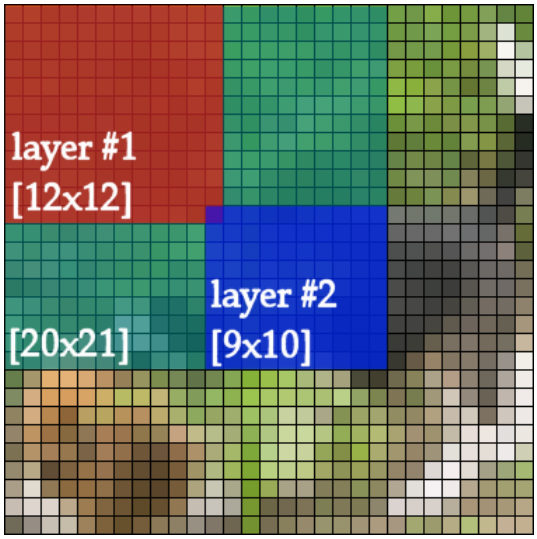


Figure 2.4: Hierarchical SFA allows not only for increasingly non-linear mappings but also for growing receptive fields. Here the first layer has a receptive field of 12×12 pixels, the second layer has a receptive field of 9×10 pixels on top of the first layer which yields together a total receptive field of 20×21 pixels.

In every layer i a filtering function $\mathbf{g}_r^i$ with spatial receptive field r is applied to the output $\mathbf{y}_{[t,s]}^{i-1}$ of layer $i-1$. The output signal $\mathbf{y}_{[t,s\ominus r]}^{i}$ of the layer i can only have smaller spatial extent than the input $\mathbf{y}_{[t,s]}^{i-1}$ which is taken into account by $s \ominus r$.

$$\mathbf{y}_{[t,s\ominus r]}^{i} = \mathbf{g}_r^i \left(\mathbf{y}_{[t,s]}^{i-1}; \mathbf{w}^i \right)$$

Taking the example of Figure 2.4 on page 20 where the image $\mathbf{y}_{[t,(1:29\times 1:29)]}^{0}$ has a spatial extent s of 29×29 pixels. The slowness function $\mathbf{g}_r^1$ in the first layer has a receptive field of $r^1 = 12 \times 12$. Thus, the output of the first layer will have a smaller spatial extent $s^1 = s^0 \ominus r^1 = (1:18 \times 1:18)$ and the output of the second layer $\mathbf{y}_{[t,s\ominus r]}^{2}$ will even be of smaller spatial extent $s^2 = s^1 \ominus r^2 = (1:10 \times 1:11)$. On the other hand, the overall receptive field of the two-layer architecture is given by $R^2 = r^1 \oplus r^2 = 12\times 12 \oplus 9\times 10 = 20\times 21$ pixels.

In each layer, the model becomes more complex and the overall receptive field does not shrink whereas the spatial extent of the output can only but decrease. The overall receptive field should clearly be smaller or equal to the spatial extent of the initial image $R^i \leq s^0$, $\forall i$ where the special case of $r^i = 1 \times 1$ corresponds to iterated SFA with constant receptive fields $R^{i+1} = R^i$ and spatial extents $s^{i+1} = s^i$ as suggested in Section 2.3.

As in layer i the slowness function $\mathbf{g}_r^i$ is applied to all points of $s^{i-1} \ominus r^i$ one can think of the architecture as a large neural network with massive weight sharing in every layer. Although one can think of a backpropagation like update rule for the network as a whole, one focuses on individual learn-

ing of every layer depending on the output of previous layers for a simple reason. Individual SFA in each layer is a one-shot learning algorithm which is guaranteed to find the global optimum. Convergence problems to local minima or hyperparameter adjustments do not have to be feared.

2.7 Measure for empirical slowness

The slowness value (in 2.16) was defined assuming the constraints are precisely met. Due to non-stationarities or noise in the data, it can turn out that the output covariance matrix $\left\langle \mathbf{y}_t \mathbf{y}_t^\top \right\rangle$ has a significant numerical difference to the unit matrix $\mathbf{I}$. If the constraints are violated, slownesses are no longer comparable between different experiments. For that reason one has to correct for the constraints in practice. This can be achieved by the whitening operation $\tilde{\mathbf{y}}_t \leftarrow \mathbf{C}_\mathbf{y}^{-\frac{1}{2}} \mathbf{y}_t$ leading by construction to $\text{cov}(\tilde{\mathbf{y}}_t) = \mathbf{I}$. A corrected empirical slowness value (2.29) along with an empirical estimator (2.30) can then be defined as $s(\mathbf{y}_t) := s(\tilde{\mathbf{y}}_t)$.

$$s(\mathbf{y}_t) = \text{tr}\left(\mathbf{C}_\mathbf{y}^{-\frac{1}{2}} \left\langle \dot{\mathbf{y}}_t \dot{\mathbf{y}}_t^\top \right\rangle \mathbf{C}_\mathbf{y}^{-\frac{1}{2}}\right) = \text{tr}\left(\mathbf{C}_\mathbf{y}^{-1} \left\langle \dot{\mathbf{y}}_t \dot{\mathbf{y}}_t^\top \right\rangle\right) \tag{2.29}$$

$$\hat{s}(\mathbf{y}_t) = \text{tr}\left(\hat{\mathbf{C}}_\mathbf{y}^{-1} \frac{1}{T} \sum_t \dot{\mathbf{y}}_t \dot{\mathbf{y}}_t^\top\right) \tag{2.30}$$

By a short calculation one can see that this measure is invariant to non-singular affine transformations:

$$\text{cov}(\mathbf{A}\mathbf{y}_t + \mathbf{b}) = \text{cov}(\mathbf{A}\mathbf{y}_t) = \mathbf{A}\text{cov}(\mathbf{y}_t)\mathbf{A}^\top \tag{2.31}$$

$$\begin{aligned} s(\mathbf{A}\mathbf{y}_t + \mathbf{b}) &\overset{(2.31)}{=} \text{tr}\left(\left(\mathbf{A}\mathbf{C}_\mathbf{y}\mathbf{A}^\top\right)^{-1} \left\langle \mathbf{A}\dot{\mathbf{y}}_t \dot{\mathbf{y}}_t^\top \mathbf{A}^\top \right\rangle\right) \\ &= \text{tr}\left(\mathbf{A}^{-\top} \mathbf{C}_\mathbf{y}^{-1} \mathbf{A}^{-1} \mathbf{A} \left\langle \dot{\mathbf{y}}_t \dot{\mathbf{y}}_t^\top \right\rangle \mathbf{A}^\top\right) = s(\mathbf{y}_t) \end{aligned} \tag{2.32}$$

Different types of measured slownesses naturally arise in the experiments dealing with sets of videos. The following table lists the nomenclature used for the experiments.

slowness	definition
s_{learn}	empirical slowness value of the learning sample from the training set of the video
s_{train}	empirical slowness value of another sample from the training set of the video
s_{test}	empirical slowness of a sample from the test set of the video
s_{val}	empirical slowness of a sample from another video, the validation video

2.7.1 Interpretation of the slowness values

Temporal variation of the decorrelated output signal $\mathbf{y}_t$ in $[0, T]$ is measured in terms of the slowness value defined to be the mean temporal variation:

$$s(\mathbf{y}_t) = \operatorname{tr}\left\langle \dot{\mathbf{y}}_t \dot{\mathbf{y}}_t^\top \right\rangle = \left\langle \dot{\mathbf{y}}_t^\top \dot{\mathbf{y}}_t \right\rangle = \sum_{j=1}^{K} \Delta(y_{j,t}) = \frac{1}{T} \sum_{j=1}^{K} \int_0^T \dot{y}_{j,t}^2 \mathrm{d}t \tag{2.33}$$

This real number $s(\mathbf{y}_t)$ has no intrinsic meaning or scale. It should only be as small as possible. A way to get an intuitive idea what this number actually signifies is borrowed from the theory of optimal free responses (see Section 2.2.3).

As a theory's result the optimal slow features to be obtained are cosine waves with $\frac{j}{2}$ periods in $[0, T]$ that can be written as $y_{j,t}^* = \sqrt{2}\cos\left(\frac{j\pi}{T}t\right)$ (2.15). Elementary analysis yields $\left\langle y_{j,t}^* \right\rangle = 0$, $\left\langle y_{i,t}^* y_{j,t}^* \right\rangle = \delta_{ij}$ and $\Delta(y_{j,t}^*) = \left(\frac{j\pi}{T}\right)^2$. With that insight, a more intuitive measure for slowness can be defined by

$$\lambda(y_{j,t}) := \frac{2\pi}{\sqrt{\Delta(y_{j,t})}} \tag{2.34}$$

which has a nice interpretation since $\lambda(y_{j,t}^*) = \frac{2T}{j}$ is the period length of the cosine wave $y_{j,t}$. Note that both the slowness value $s(\mathbf{y}_t)$ and the measure $\lambda(\mathbf{y}_t)$ for an output signal do not depend on the number of time steps or the length T of the interval. An illustration is given by Figure 2.5.

That means, the smallest slowness value would be equal to

$$s_K^* := s(\mathbf{y}_t^*) = \frac{\pi^2}{T^2} \sum_{k=1}^{K} k^2 = \frac{\pi^2}{6T^2}\left(2K^3 + 3K^2 + K\right) \tag{2.35}$$

and the smallest corresponding slowness measure would equal

$$\lambda_K^* := \lambda(\mathbf{y}_t^*) = \frac{2\pi}{\sqrt{s(\mathbf{y}_t^*)}} = \frac{2T}{\sqrt{\frac{K^3}{3} + \frac{K^2}{2} + \frac{K}{6}}} \approx 2T\sqrt{3}K^{-\frac{3}{2}} \tag{2.36}$$

– an expression serving as a benchmark value for a given number of slow components K. Even though $s(\mathbf{y}_t^*)$ depends explicitly on T, this does not mean that given a number T of patches sampled from a video, the slowness $s(\mathbf{y}_t)$ of the output $\mathbf{y}_t$ depends on T. This is only true in the sense more training examples might allow for better slowness functions g or possible overfitting leads to bigger slowness values. The relation between $\lambda(\mathbf{y}_t)$ and $s(\mathbf{y}_t)$ is shown in Figure 2.5.

2.7.2 Error bars for the slowness values

Jackknife

Given an estimator $\hat{\theta} = \hat{\theta}(\mathbf{x}_1, \ldots, \mathbf{x}_T)$ for an unknown quantity θ, one is interested in the bias

$$b = \theta - \left\langle \hat{\theta} \right\rangle$$

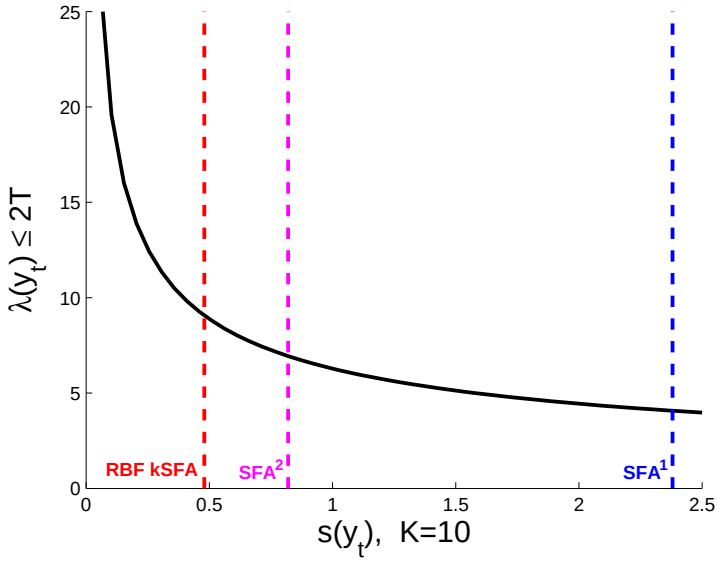


Figure 2.5: The figure relates the slowness values obtained in the simulations to the period lengths of cosine waves, which are known to be the slowest possible signal and thus give an intuition on the meaning of the numbers.
If $K = 10$ slow components are retained, the smallest slowness value would be $s^*_{10} = \frac{3800}{T \cdot T}$. That means for a training sequence of $T = 200,000$ data points $s^*_{10} = 10^{-7}$, which is clearly below the given reference slownesses for linear SFA, quadratic SFA and kernelized SFA using Laplacian kernels. The corresponding period length λ^*_{10} is in the order of 22,000 frames.

that is the expected deviation from the true value and in the variance

$$v = \left\langle \left(\hat{\theta} - \left\langle \hat{\theta} \right\rangle \right)^2 \right\rangle$$

of the estimator which is the degree of accuracy of the estimate. In our case, one would estimate the slowness that can be obtained by applying the SFA algorithm.

Jackknife provides estimates for both bias and variance of an arbitrary estimator based on leave-one-out statistics. The leave-one-out estimate (2.37) and the mean leave-one-out estimate (2.38) are defined as follows:

$$\hat{\theta}_{\neg t} := \hat{\theta}(\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}, \mathbf{x}_{t+1}, \ldots, \mathbf{x}_T) \tag{2.37}$$

$$\hat{\theta}_{(\cdot)} := \frac{1}{T} \sum_{t=1}^{T} \hat{\theta}_{\neg t} \tag{2.38}$$

Finally, these quantities are combined to obtain an estimate for the bias b and the variance v of the estimator $\hat{\theta}$.

$$\hat{b} = (T-1)\left(\hat{\theta}_{(\cdot)} - \hat{\theta}\right) \tag{2.39}$$

$$\hat{v} = \frac{T-1}{T}\sum_{t=1}^{T}\left(\hat{\theta}_{\neg t} - \hat{\theta}_{(\cdot)}\right)^2 \tag{2.40}$$

Cross-validation

A more heuristic approach to measure the generalization performance of an algorithm is the widespread method of cross-validation. In the simplest case, the dataset is split into two disjoint subsets, the training set and the usually smaller test set. The analysis is performed on the training set whereas the test set is used for validation of the results.

When one would also like to use the test set for model selection as well, one can split the data set $\mathbf{X}$ into K disjoint subsets $\mathbf{X}_k$ of same size and iterate the above method so that training is done on the set $\mathbf{X}_{\neg k} := \mathbf{X} \backslash \mathbf{X}_k$ and testing is done on $\mathbf{X}_k$. In so-called K-fold cross-validation one obtains K different estimates that can serve to calculate mean and standard deviation. Typical values for K are in the order of 10.

In the limit when K gets equal to the number of data points, cross-validation becomes equivalent to jackknife.

Efficient implementation using partial statistics

If one wants to run a K-fold cross-validation experiment, one should split the dataset into K disjoint subsets. As the data points in $\mathbf{X}$ have a temporal structure, it is obvious that the subsets should contain a temporal sequence rather than a random collection of data points. Otherwise temporal differences $\mathbf{x}_{t+1} - \mathbf{x}_t$ would be meaningless. As the SFA algorithm needs the statistics $\mathbf{A} = \hat{\text{cov}}(\dot{\mathbf{x}}_t)$ and $\mathbf{B} = \hat{\text{cov}}(\mathbf{x}_t)$ for each run $r \in [1 \dots R]$, one has to calculate $\mathbf{A}_k$, $\mathbf{B}_k$ for testing and $\mathbf{A}_{\neg k}$, $\mathbf{B}_{\neg k}$ for training.

From a computational perspective this would mean the whole data set has to be processed K times. This is a very time consuming issue and one can think of the following approximation. In the context of SFA on from video sampled image patches, cross-validation is done in order to see how good the learned functions deal with non-stationarities present in the data rather than to provide different kinds of transformations in each run. If the first raw moments $\mathbf{m}_k$ and the second raw moments $\mathbf{M}_k$ and $\dot{\mathbf{M}}_k$ of the data are calculated individually for each subset $\mathbf{X}_k$ one can combine those in order to obtain the required statistics by running only once through the data set.

$$\mathbf{m}_k = \frac{1}{2T}\sum_{t=1}^{2T}\mathbf{x}_{k,t} \tag{2.41}$$

$$\mathbf{M}_k = \frac{1}{2T-1}\sum_{t=1}^{2T}\mathbf{x}_{k,t}\mathbf{x}_{k,t}^\top \tag{2.42}$$

$$\dot{\mathbf{M}}_k = \frac{1}{T-1}\sum_{t=1}^{T}(\mathbf{x}_{k,t+T}-\mathbf{x}_{k,t})(\mathbf{x}_{k,t+T}-\mathbf{x}_{k,t})^\top \tag{2.43}$$

Then, it is possible to calculate the statistics as:

$$\begin{aligned}\mathbf{A}_k &= \dot{\mathbf{M}}_k \\ \mathbf{B}_k &= \mathbf{M}_k - \mathbf{m}_k\mathbf{m}_k^\top \end{aligned} \tag{2.44}$$

$$\begin{aligned}\mathbf{A}_{\neg k} &= \frac{1}{R-1}\sum_{\kappa\neq k}\dot{\mathbf{M}}_\kappa \\ \mathbf{B}_{\neg k} &= \frac{1}{R-1}\sum_{\kappa\neq k}\mathbf{M}_\kappa - \frac{1}{(R-1)^2}\left(\sum_{\kappa\neq k}\mathbf{m}_\kappa\right)\left(\sum_{\kappa\neq k}\mathbf{m}_\kappa^\top\right)\end{aligned} \tag{2.45}$$

In case of expanded or kernelized SFA, the signals $\mathbf{x}_t$ have to be replaced by their expanded $\mathbf{\Phi}(\mathbf{x}_t)$ or kernelized $\mathbf{k}(\mathbf{x}_t)$ equivalents.

Chapter 3

Implementation

The next chapter deals with practical considerations and implementation details. First, a brief discussion of the utilized programming language is given, followed by a list of implemented functionality. Subsequently, the data sets along with arising storage issues are introduced. Finally, efficient learning, optimization and filtering are treated.

3.1 Library Implementation

Code written for this thesis is organized as follows: The directories `@preproc/` and `@sfa/` contain two objects implementing PCA and SFA, in `shared/` auxiliary functions are listed and in `experiments/` one finds the scripts for the various numerical simulations.

The code can be put into five main categories:

- data acquisition and processing
  Camera access is assured, storage formats are addressed and data is randomly sampled according to a given policy.

- learning
  Necessary statistics are iteratively updated with expanded and kernelized chunks from the data stream.

- optimization
  Projections can be calculated based on the statistics using SFA and sparse optimization strategies.

- filtering
  The learned filters can efficiently be applied to large multidimensional datasets.

- visualization
  Analysis and interpretation can be done with a set of plotting and drawing functions.

3.1.1 MATLAB and MEX

The choice of the programming language was driven by the nature of the problem and by the suitability of available solutions. As the video data set is structured as a large multidimensional array and the algorithms involve a lot of vector and matrix manipulations a numerical computing environment seems a favorable choice.

MATLAB[1] (a shorthand for MATrix LABoratory) is widely used in prototyping, technical computing and digital image and signal processing. It is a commercial product running on Linux, OS X, Solaris and Windows enabling simple data acquisition, exploration, manipulation and visualization. Although MATLAB is proprietary, many machine learning applications make use of its functionality. This might be partly due to public domain substitutes like SCILAB[2] and OCTAVE[3] offering at least in principle compatibility to MATLAB code. Other arguments comprise the consistent access to standard data acquisition and signal processing libraries or the possibility to include native C-code by the MEX interface.

3.1.2 Objects

Besides its imperative capabilities, MATLAB offers the possibility to build structured data types following the OOP paradigm. Two objects were implemented for the purpose of better reusability .

The first object implements a demeaning and whitening filter for N-dimensional data sets. In fact, the object is a linear filter with a convenient possibility to infer the filter coefficients from the data set.

`preproc/method`	implemented functionality
`set, get`	set and get object properties
`preproc, save`	constructor, load from disk, save to disk
`plot, display`	visualize the object
`apply`	perform filtering
`learn`	add new data to the statistics
`pca`	find principal components

The second object is less trivial and offers lots of capabilities in terms of SFA. Efficient filtering as described in 3.4 and sparse optimization as detailed in 4.3.4 can be done by the object.

[1] See the website `http://www.mathworks.com/` for details.
[2] Visit `http://www.scilab.org/` for further information.
[3] Consult `http://www.gnu.org/software/octave/` for a specific description.

sfa/method	implemented functionality
set, get	set and get object properties
sfa, save	constructor, load from disk, save to disk
plot, display	visualize the object
apply	perform filtering $\mathbf{y} = \mathbf{g}(\mathbf{x})$
learn	add new data to the statistics $\mathbf{A}$ and $\mathbf{B}$
optimize	find (sparse) slow components

3.1.3 Auxiliary Scripts

Visualization and evaluation

For visualization purposes, two public domain tools were used; namely the `subsubplot.m`[4] script allowing for nested subplots in MATLAB and the `laprint.m`[5] package enabling LaTeX-symbols anywhere in MATLAB figures. The nice receptive field images of Figure 2.2 were calculated using the routine `optimal_stimulus.m`[6] taken from [Berkes and Wiskott, 2006].

Selfwritten routines include `colimg.m` – a script producing plots superposing a gray value image with a color coded feature image like Figure 4.20. Efficient linear filtering as described in Section 3.4.1 can be performed with `linfilter.m`.

Quantization

The transfer function of the A-law algorithm from Section 3.2.2 is implemented in `alaw.m` and its inverse is coded in `ialaw.m`.

Quantization of double arrays of arbitrary size into 8bit arrays is accomplished by `alaw_quant8.m` and its counterpart `ialaw_quant8.m`.

Optimization

The numerical solution to the linear SFA problem based on two-fold PCA as proposed in Section 2.2.4 can be found by `pdeig.m`. Singular data covariance matrices do not pose problems to the algorithm as singular dimensions are simply discarded and not taken into account – therefrom the name `pdeig(A,B)` in analogy to `eig(A,B)` where the covariance matrix $\mathbf{B}$ is approximated by a positive definite matrix. Switching between different eigenvalue decompositions is possible.

Sparse solutions as proposed by Vollgraf and Obermayer [2006] and portrayed in Section 4.3.4 involving single slow components can be obtained

[4]The code is available at `http://www.kenschutte.com/subsubplot/`.

[5]Code can be downloaded from `http://www.uni-kassel.de/fb16/rat/matlab/laprint/`.

[6]The site `http://www.gatsby.ucl.ac.uk/~berkes/software/qforms-tk/index.html` offers code and further information.

by application of `ksfmin.m`[7]. The extended version (JHECGD) allowing for joint sparsification of several slow components is termed `ksfmin2.m`. The line search routine can be interchanged by switching between Golden Section Search [Press et al., 1993, §10.1] `goldsect.m` and Brent's Method [Press et al., 1993, §10.2] `brent.m`.

A wrapper combining SFA optimization and sparse optimization in one single interface is provided by `sopt.m`.

SFA auxiliary functions

A fixed number of patches per frame at random positions can be clipped from a stack of video frames with `clippatches.m`. A fixed number of patches serving as support vectors can be drawn from the whole video using `drawsv.m`. Additionally one can impose a certain distance between the support vectors in feature space as proposed by the FVS Algorithm 1 from Section 4.3.2.

Explicit expansion into some high dimensional feature space and implicit expansion via kernels are implemented in the function `expansion.m`; the small script `expdim.m` provides the dimension of the expanded vector. All kernels mentioned in Section 4.1.1 are available.

Finally, `timediff.m` allows to calculate approximations to the temporal derivative of a signal such as finite differences or cubic interpolations. Last but not least, `jaccheck.m` is a tool comparing the numerical and the analytical derivative (Jacobian) $\mathbf{J_X} \in \mathbb{R}^{\mathbf{M}\times\mathbf{N}}$ of a function $\mathbf{F} : \mathbb{R}^{\mathbf{N}} \to \mathbb{R}^{\mathbf{M}}$ with $\mathbf{N} = N_1 \times N_2 \times \cdots \times N_n$ and $\mathbf{M} = M_1 \times M_2 \times \cdots \times M_n$ of a multidimensional array $\mathbf{X} \in \mathbb{R}^{\mathbf{N}}$.

3.1.4 MEX-Functions

Some routines are computationally very demanding especially for filtering. They require many operations or have a structure one cannot easily put into matrix notation. The three filtering routines deal with data of arbitrary dimensionality and can be compiled by executing `make_mex.m`. Linear filtering can be done with `lfiltn.c`, filtering based on quadratically expanded SFA is implemented by `qfiltn.c` and `kfiltn.c` can be used to perform SFA filtering using a variety of kernels.

A second reason for an implementation in plain C is the use of image acquisition libraries and the speedup gained by a more direct implementation. Two functions building up one another are given by `readjpeg.{h,c}` and `jpegpatchextract.c`. They offer the possibility to directly read image files or even videos stored as a sequence of single images into memory.

[7]Code can be obtained from `http://ni.cs.tu-berlin.de/software/hecgd/index.html`.

3.2 Data Acquisition and Storage

(a) Lab

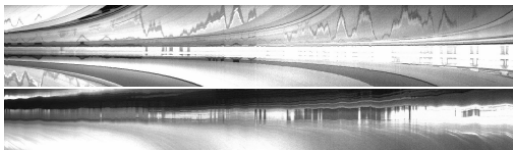
(c) Lab

(b) Floor, simulated

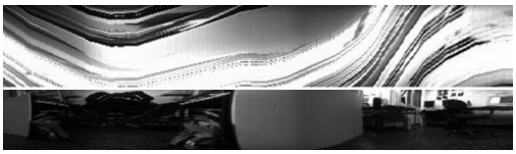
(d) Floor

Figure 3.1: The major part of the numerical simulations were carried out on four different videos. The first two videos contain 26,951 and 22,552 frames respectively and were recorded in the lab of the robot. From the floor in front of the lab, two additional videos were used, a shorter one coming from the simulator with 6,933 frames and a longer one containing 37,406 frames.

Using the blackbox library, Digiclops®, three stereo videos were recorded – two of them in the lab of the robot and one in the floor in front of the lab. Thus, different sets of stimuli were available. In Figure 3.1, typical examples of video frames are shown. The data set has two spatial dimensions height $h \in \{1, \ldots, 240\}$ and width $w \in \{1, \ldots, 320\}$, one temporal dimension $t \in \{1, \ldots, T\}$, consists of two stereo channels $s \in \{L, R\}$ and can contain three color channels $c \in \{R, G, B\}$. So it can be put into an array with five indices (h, w, t, s, c).



(a) Forward movement in the floor



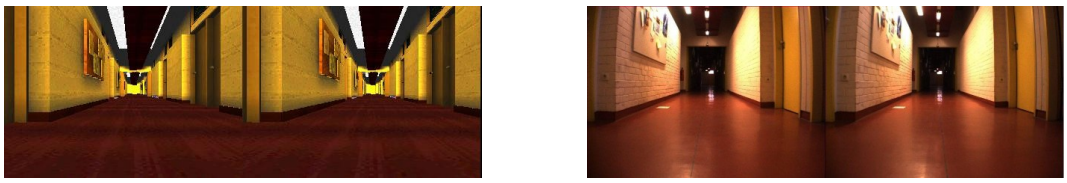
(b) Rotation in the lab

Figure 3.2: Two temporal slices through two different video stacks containing 500 successive frames each are shown. The upper part of the images corresponds to the middle horizontal scanline and the lower part of the images corresponds to the centered vertical scanline. Time corresponds to the horizontal direction. One can clearly see the dominance of movement along the horizontal direction.

Single frames reflect only spatial structure (h, w) and reveal nothing about the temporal dynamics of the data sets. Therefore one can plot slices through

the video stack as shown in Figure 3.2 and thus visualize the pairs (h, t) and (w, t) as done in the upper and lower part of the plots respectively. By visual inspection one can see that the rate of change in horizontal direction (~ 6 pixels/frame) of the image plane is much larger than the change in vertical direction (~ 1.5 pixels/frame) for both rotation and forward translation of the robot.

Finally, in Figure 3.3 one can compare images grabbed by the camera and images generated by the simulator based on the open source game engine, Crystal Space 3D[8].



(a) Floor from simulator (b) True floor

Figure 3.3: The floor was reproduced as good as possible with the simulator. Differences include light sources from the ceiling, reflection on the ground, darkness of distant points and camera distortion clearly visible at the corners.

3.2.1 Image compression

A rough calculation shows that colored images of size $2 \cdot 320 \times 240$ would require 460kB of memory each if stored in raw format. Video compression cannot be done in realtime by the robot's processor. Therefore, frame-wise storage is the only feasible solution facilitating also the access of a special frame. The use of `libjpeg`[9], a standard image compression library based on the discrete cosine transform, can reduce the image size to 30kB on average. Thus, a video containing 25,000 frames needs 750MB instead of 11.5GB. Very good lossless compression is achieved with `libpng`[10]. However, these compression methods are limited to images with a previously known range ($[0..255]$ or $[0..1]$) or a previously adapted range.

3.2.2 Quantization

If confronted with the situation of having to store the first K slow components $\mathbf{Y}^1, \ldots, \mathbf{Y}^K$ (given in double precision) of a given image $\mathbf{X}$ where the range of $\mathbf{Y}^k$ is not known a priori, different methods have to be employed. Storing filtered images in double precision clearly consumes too

[8] Code can be obtained from `http://www.crystalspace3d.org/`.

[9] From `http://www.ijg.org/` one can download the actual version 6b originally written in 1998.

[10] Code is available at `http://www.libpng.org/pub/png/libpng.html`.

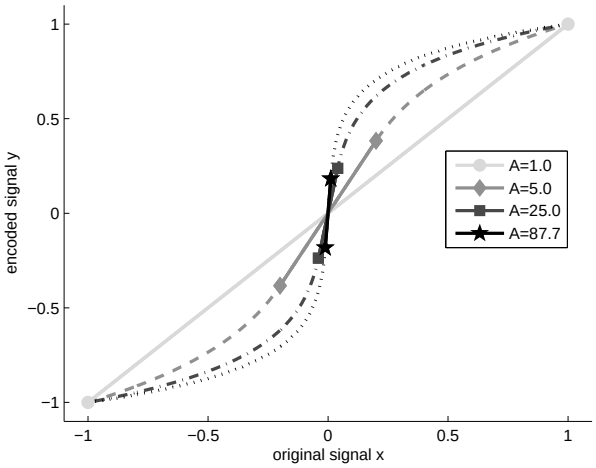


Figure 3.4: For larger values of A the transfer function for A-law compression becomes steeper, the linear part (shown as straight line) becomes smaller and more bits are reserved for small values. The function has a unique inverse and is continuous and differentiable.

much memory. One searches a quantization scheme minimizing the error. Values occurring frequently should be quantized more precisely whereas rare values can be quantized less accurately. From the zero mean and the unit variance constraint for the filter outputs, one can deduce that there will be positive and negative values. Furthermore from experience one knows values to be strongly peaked at zero. Optimal quantization would be obtained for a kept accuracy depending on the log of the relative frequency of occurrence of the value. A good approximation assumes a peaked histogram.

A compression scheme which is used in telecommunication is called A-law encoding and is based on the idea that differences in higher frequencies are less good to distinguish than lower frequencies. One should spend more bits for the low frequencies.

That means, before rounding to the next integer, a transfer function as shown in 3.4 is applied:

$$f(x) = \frac{\operatorname{sign}(x)}{1+\ln(A)} \begin{cases} A\,|x| & 0 \le |x| < \frac{1}{A} \\ 1 + \ln(A\,|x|) & \frac{1}{A} \le |x| \le 1 \end{cases} \tag{3.1}$$

$$f^{-1}(y) = \frac{\operatorname{sign}(y)}{A} \begin{cases} |y| + |y|\ln(A) & 0 \le |y| < \frac{1}{1+\ln(A)} \\ \exp(|y| + |y|\ln(A) - 1) & \frac{1}{1+\ln(A)} \le |y| \le 1 \end{cases} \tag{3.2}$$

The value A interpolates between the linear map $(A = 1)$, a logarithmic map (A larger) and the sign function $(A = \infty)$. At $x = \{-\frac{1}{A}, 0, \frac{1}{A}\}$ the functions f and f^{-1} are continuous and differentiable. The steeper the function is, the denser are the quantization steps and thus the more accurate is the

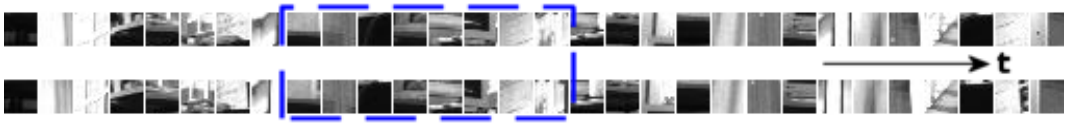


Figure 3.5: Input data is considered to be a stream of patches clipped from successive frames of a video. Following the sequential structure of the video, statistics needed for optimization are iteratively updated based on their old values and on the newly arriving data chunks $\mathbf{X}^*$. Due to this strategy, memory requirements are low because only the statistics and one single data chunk have to be kept in memory simultaneously.

procedure. If compared to equidistantly quantized images, one can measure significant differences in terms of quantization errors.

3.3 Learning and Optimization

3.3.1 Learning

Given a data matrix $\mathbf{X}$, one can, based on the raw moments of first and second order $\mathbf{M}_1(\mathbf{X}) := \frac{1}{T}\mathbf{X}\mathbb{1}$ and $\mathbf{M}_2(\mathbf{X}) := \frac{1}{T}\mathbf{X}\mathbf{X}^\top$, write the estimator for the covariance as:

$$\hat{\text{cov}}(\mathbf{x}_t) \overset{(2.2)}{=} \mathbf{M}_2(\mathbf{X}) - \mathbf{M}_1(\mathbf{X})\mathbf{M}_1(\mathbf{X})^\top \tag{3.3}$$

One can update the raw moments $\mathbf{M}_i,\ i \in \{1, 2\}$ with a recently arrived data chunk $\mathbf{X}^*$ comprising T^* data points by:

$$\mathbf{M}_i(\mathbf{X}, \mathbf{X}^*) = \frac{T}{T+T^*}\mathbf{M}_i(\mathbf{X}) + \frac{T^*}{T+T^*}\mathbf{M}_i(\mathbf{X}^*) \tag{3.4}$$

An illustration of the iterative update rule of the raw moments can be found in Figure 3.5, where the data stream of image patches and a newly arriving data chunk $\mathbf{X}^*$ are depicted.

3.3.2 Optimization

Statistics are not calculated for the data $\mathbf{x}_t$ itself. Except for the linear case, statistics are calculated from the expanded $\mathbf{\Phi}_t$ or kernel expanded data $\mathbf{k}_t$. That means, support vectors and kernels have to be known in advance. Parameter exploration requires recalculation of the statistics for each expansion.

Optimization means in the context of this thesis, calculation of the weight matrices $\mathbf{W}$ for the linear projections in the expanded space. There are two possibilities. Either one applies the SFA algorithm directly or one does some selection of support vectors via sparse regularization or Greedy methods to exclude some dimensions in the expanded space.

3.4 Efficient Filtering

One single image patch $\mathbf{x}$ of size (h, w) will be filtered by

$$\mathbf{y} = \mathbf{g}(\mathbf{x}) = \mathbf{W}^\top \left(\mathbf{\Phi}(\mathbf{x}) - \mathbf{\Phi}_0\right)$$

in the case of linear and expanded SFA and by

$$\mathbf{y} = \mathbf{g}(\mathbf{x}) = \mathbf{W}^\top \left(\mathbf{k}(\mathbf{Z}, \mathbf{x}) - \mathbf{k}_0\right)$$

in the case of kernelized SFA. If one wants to filter a whole image $\mathbf{X}$ of size (H, W), the functions $\mathbf{g} = (g_1, \ldots, g_K)$ have to be applied at all locations (i, j) of the image.

One single slow component $\mathbf{Y}^k$ of size $(H - h + 1, W - w + 1)$ is obtained as follows:

$$\begin{aligned} \mathbf{Y}^k_{i,j} &= \mathbf{w}_k^\top \left(\mathbf{\Phi}(\mathbf{X}_{[i:i+h-1,j:j+w-1]}) - \mathbf{\Phi}_0\right) \\ \mathbf{Y}^k_{i,j} &= \mathbf{w}_k^\top \left(\mathbf{k}(\mathbf{Z}, \mathbf{X}_{[i:i+h-1,j:j+w-1]}) - \mathbf{k}_0\right) \end{aligned} \tag{3.5}$$

As shorthands, one can write:

$$\mathbf{Y}^k = \mathbf{w}_k^\top \mathbf{h}(\mathbf{X}) - \mathbf{Y}^k_0 \text{ with } \mathbf{h} \in \{\mathbf{\Phi}, \mathbf{k}(\mathbf{Z}, \cdot)\} \tag{3.6}$$

Let M be the dimension of the expansion either explicit or in terms of support vectors. Then, the computational complexity of the filtering of one image based on (3.5) for all K slow components jointly can be written as follows:

$$\mathcal{O}\left(\{H - h + 1\}\{W - w + 1\} \cdot hw \cdot M + K \cdot M\right) \tag{3.7}$$

Here, the size of the output images is taken into account as well as the receptive field size of a single patch and the number of components in expanded space. The second summand reflects the projections in expanded space. Note that the second slow component is computationally much cheaper then the first one because the full expansion can be reused.

A video with $25,000$ frames of size 240×320 contains 1.92 billion pixels. Filtering based on $M = 300$ support vectors of size 10×10 requires in the order of 60Tflop. Real-Time processing would require 70 Gflops (like a Cray-1 from 1976) which is clearly too much for the robot's CPU.

Considering the video as a fixed quantity, one has only two obvious possibilities to increase processing speed: Keep the dimension of the expanded space as small as possible (potentially using sparse optimization techniques from Section (4.3.4)) and have small receptive field sizes.

But when does one really need to filter a whole image? On the one hand one may wish to visualize the outputs. On the other hand one may want to learn several layers. In order to learn a higher layer one needs patches clipped from the output of the previous layer. One can switch between filtering the whole image and clipping patches from the filtered video (offline filtering) and clipping larger patches from the first layer and applying the

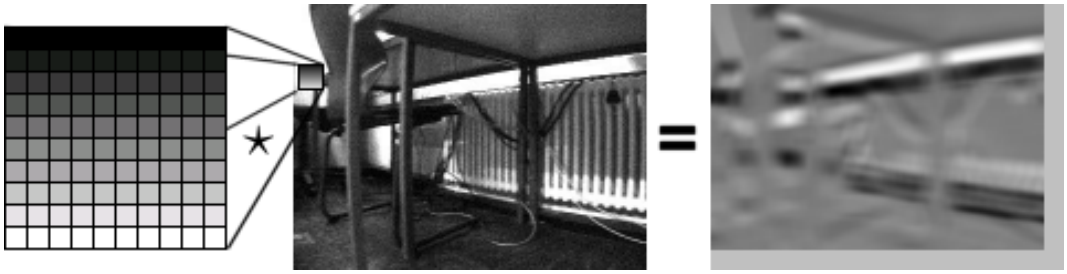


Figure 3.6: A filter mask $\mathbf{z}$ *of size* 10×10 *pixels is applied to the image* $\mathbf{X}$*. The output image* $\mathbf{Y}$ *is* 9×9 *pixels smaller in both height and width.*

previous layer to them (online filtering). Offline filtering is very demanding in terms of memory. Online filtering is computationally more expensive.

The linear projection of the filtering requires a matrix multiplication with a weight matrix $\mathbf{W}$. By using fast matrix multiplication algorithms such as the Strassen's algorithm one can reduce complexity of matrix multiplication from $\mathcal{O}(N^3)$ to $\mathcal{O}(N^{\log_2(7)}) = \mathcal{O}(N^{2.807})$ at least for square matrices of size N.

Finally, all calculations necessary for filtering work pixel by pixel and can consequently carried out in parallel. For example, modern graphics cards with 48 pixel pipelines and advanced programmability are promising candidates for fast parallel processing.

In the following sections, fast linear filtering is reviewed and quadratically expanded SFA as well as kernelized SFA based on dot product and RBF kernels are put down to linear filtering.

3.4.1 Fast linear filtering

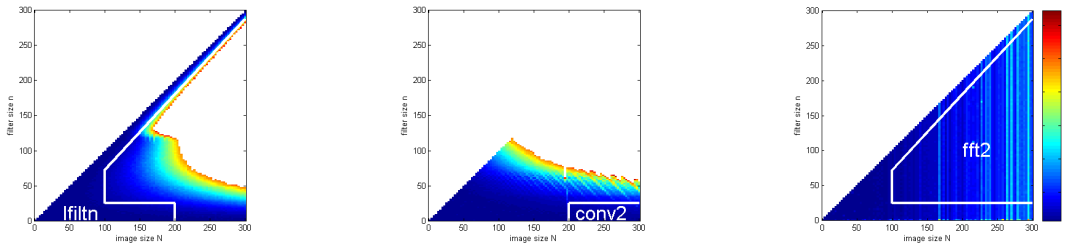
Discrete real linear filtering as shown in Figure (3.6) is done by sliding the filter $\mathbf{z}$ of size $(h \times w)$ over all locations of the image $\mathbf{X}$ of size $(H \times W)$ and calculating a weighted sum corresponding to the dot product between the filter and the image patch $\mathbf{x}$ at the special location:

$$(\mathbf{z} \star \mathbf{X})_{i=1,j=1}^{H,W} = \sum_{k=1}^{h} \sum_{l=1}^{w} \mathbf{z}_{k,l} \mathbf{X}_{i+k-1,j+l-1} = \mathbf{z}^\top \mathbf{x}_{[i:i+h-1,j:j+w-1]} \tag{3.8}$$

The complexity for linear filtering is $\mathcal{O}((H-h+1) \cdot (W-w+1) \cdot h \cdot w)$. From the convolution theorem it is known that a convolution in the time domain becomes a simple multiplication in the frequency domain:

$$\mathbf{z} * \mathbf{X} = \mathcal{F}^{-1} \left\{ \mathcal{F}[\mathbf{z}] \cdot \mathcal{F}[\mathbf{X}] \right\} \tag{3.9}$$

In our case, $\mathcal{F}$ would be the two dimensional discrete Fourier transform (`fft2` in MATLAB) which can be efficiently calculated in $\mathcal{O}(H \cdot W \cdot \log(H \cdot W))$. One has to pad $\mathcal{F}(\mathbf{z})$ with zeros to carry out the multiplication with



(a) Linear filter (b) MATLAB `conv2` (c) MATLAB `fft2`

Figure 3.7: The three subplots show the calculating time (coded as color) necessary for performing linear filtering depending on the size of the image and the filter. MATLAB `conv2.m` is cache-optimized and works best for small filters compared to large images. The function `lfiltn.c` is fastest for filters large compared to the image because only the valid part is calculated. The intermediate region is fastest calculated in the frequency domain. The framed regions correspond to the strategy chosen by the `linfilter.m` routine.

$\mathcal{F}(\mathbf{X})$ which is usually of bigger size. Furthermore, convolution is related to filtering (cross correlation) by

$$\mathbf{z} \star \mathbf{X} = F(\mathbf{z}) * \mathbf{X} \tag{3.10}$$

where the operator F flips the filter $\mathbf{z}$ in both horizontal and vertical direction. Putting all this together, filtering in the frequency domain is done as follows:

$$\mathbf{z} \star \mathbf{X} \quad = \quad \mathcal{F}^{-1}\left\{\mathcal{F}[F(\mathbf{z})] \cdot \mathcal{F}[\mathbf{X}]\right\} \tag{3.11}$$

The three ways of calculating the output of a linear filter, direct sum (3.8), via convolution (3.10) and in the frequency domain (3.11) yield different running times as shown in Figure 3.7. Thus, one can think of a strategy switching between the three methods as depicted in the figure.

As a rule of thumb, one should filter in the frequency domain when the filter mask is larger than 25×25 pixels.

3.4.2 Quadratically expanded SFA

As the dimension of the expanded space containing all polynomials up to degree two has a huge dimension, one usually applies a dimensionality reduction step to the data $\mathbf{x}$ before the expansion. After subtracting the mean $\mathbf{x}_0$, the data is projected onto the first principal components corresponding to the directions retaining most of the variance contained in the original data set.

$$\tilde{\mathbf{x}} = \mathbf{S}^\top(\mathbf{x} - \mathbf{x}_0) \tag{3.12}$$

The linear transformation $\mathbf{S}$ performs a sphering, which means that $\tilde{\mathbf{x}}$ is decorrelated and has unit variance. This is a helpful property preventing too large values due to the squaring. Filtering is done as expansion and linear projection in the expanded space $\mathbf{y} = \mathbf{W}^\top \mathbf{\Phi}(\tilde{\mathbf{x}})$ whose components y_k can be written as a quadratic function in $\tilde{\mathbf{x}}$ and as quadratic function in $\mathbf{x}$:

$$y_k = \mathbf{w}_k^\top \mathbf{\Phi}(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}}^\top \mathbf{H}_k \tilde{\mathbf{x}} + \mathbf{f}_k^\top \tilde{\mathbf{x}} = \mathbf{x}^\top \mathbf{A}_k \mathbf{x} + \mathbf{b}_k^\top \mathbf{x} + c_k \tag{3.13}$$

Dropping the index k and arranging the variables, one finds $\mathbf{H}_k = \mathbf{H}_k^\top$ and $\mathbf{f}_k$ as a function of $\mathbf{w}_k$ in the following way:

$$\begin{aligned} y &= \begin{pmatrix} w_1 \\ \vdots \\ w_N \\ w_{N+1} \\ w_{N+2} \\ w_{N+3} \\ \vdots \\ w_{(N^2+N)/2} \end{pmatrix}^\top \begin{pmatrix} \tilde{x}_1 \\ \vdots \\ \tilde{x}_N \\ \tilde{x}_1^2 \\ \tilde{x}_2\tilde{x}_1 \\ \tilde{x}_2^2 \\ \vdots \\ \tilde{x}_N^2 \end{pmatrix} = \mathbf{w}^\top \mathbf{\Phi}(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}}^\top \mathbf{H} \tilde{\mathbf{x}} + \mathbf{f}^\top \tilde{\mathbf{x}} \\ &= \tilde{\mathbf{x}}^\top \begin{pmatrix} w_{N+1} & \cdots & \frac{1}{2} w_{1+(N^2+N)/2} \\ \frac{1}{2} w_{N+2} & & \\ \vdots & \ddots & \vdots \\ \frac{1}{2} w_{1+(N^2+N)/2} & \cdots & w_{(N^2+N)/2} \end{pmatrix} \tilde{\mathbf{x}} + \begin{pmatrix} w_1 \\ \vdots \\ w_N \end{pmatrix}^\top \tilde{\mathbf{x}} \end{aligned} \tag{3.14}$$

One can plug the sphering step (3.12) into (3.14) to obtain the expressions $\mathbf{A}_k = \mathbf{S}\mathbf{H}_k\mathbf{S}^\top$, $\mathbf{b}_k = \mathbf{S}\mathbf{f}_k - 2\mathbf{A}_k\mathbf{x}_0$ and $c_k = -\mathbf{b}_k^\top \mathbf{x}_0 - \mathbf{x}_0^\top \mathbf{A}_k \mathbf{x}_0$.

If the first N principal components are retained in the preprocessing step the symmetric matrix $\mathbf{A}_k$ will be of rank N. With the help of the eigenvalue decomposition $\mathbf{A}_k = \sum_{n=1}^{N} \lambda_n \mathbf{v}_n \mathbf{v}_n^\top$ the filter can be written as:

$$y_k = \sum_{n=1}^{N} \lambda_{k,n} \left(\mathbf{v}_{k,n}^\top \mathbf{x} \right)^2 + \mathbf{b}_k^\top \mathbf{x} + c_k \tag{3.15}$$

The computational complexity of this expression is much smaller, because the eigenvectors $\mathbf{v}_{k,n}$ can be precomputed and only $N + 1$ linear filtering operations are necessary unlike the $(N^2 + N)/2$ operations needed in the case of explicit expansion.

$$\mathbf{Y}^k = \sum_{j=1}^{J} \lambda_{k,j} \left(\mathbf{v}_{k,j} \star \mathbf{X} \right)^2 + \mathbf{b}_k \star \mathbf{X} + c_k \tag{3.16}$$

One should mention a drawback of the proposed eigenbasis filtering method, that the linear filtering operations cannot be reused for several slow components because the eigenvectors $\mathbf{v}_{k,j}$ depend on k.

3.4.3 Kernelized SFA using dot product kernels

Kernels depending on a general dot product $\langle \mathbf{z}, \mathbf{x} \rangle$ induced by a symmetric and positive definite matrix $\mathbf{C}$ have the form

$$k(\mathbf{z}, \mathbf{x}) = f(\langle \mathbf{z}, \mathbf{x} \rangle) = f(\mathbf{z}^\top \mathbf{C^{-1}} \mathbf{x}) \tag{3.17}$$

and can be rewritten using the substitution $\tilde{\mathbf{z}} = \mathbf{C^{-1}}\mathbf{z}$.

$$k(\tilde{\mathbf{z}}, \mathbf{x}) = f(\tilde{\mathbf{z}}^\top \mathbf{x}) \tag{3.18}$$

Once again, the final filtering routine can be written in terms of linear filtering operations, one can efficiently perform in the frequency domain.

$$\mathbf{Y}^k = \sum_{i=1}^{M} \mathbf{w}_{k,i} k(\mathbf{z}_i, \mathbf{X}) - \mathbf{Y}_0^k = \sum_{i=1}^{M} \mathbf{w}_{k,i} f(\tilde{\mathbf{z}}_i \star \mathbf{X}) - \mathbf{Y}_0^k \tag{3.19}$$

The number of linear filtering operations is equal to the number M of support vectors $\mathbf{z}_i$. Nicely, one can reuse the filtered images $\tilde{\mathbf{z}}_i \star \mathbf{X}$ for all slow components.

3.4.4 Kernelized SFA using RBF kernels

Finally, one can, following Vollgraf et al. [2004], also rewrite the filtering with an RBF kernel based on the Euclidean distance by utilizing linear operations.

$$k(\mathbf{z}, \mathbf{x}) = f(\|\mathbf{z} - \mathbf{x}\|^2) = f(\mathbf{z}^\top \mathbf{z} - 2\mathbf{z}^\top \mathbf{x} + \mathbf{x}^\top \mathbf{x}) \tag{3.20}$$

$$k(\mathbf{z}, \mathbf{X}) = f(\mathbf{z}^\top \mathbf{z} - 2\mathbf{z} \star \mathbf{X} + \mathbb{1} \star \mathbf{X}^2) \tag{3.21}$$

Here, $\mathbb{1}$ denotes a filter of the same size as the support vectors containing ones as entries.

$$\mathbf{Y}^k = \sum_{i=1}^{M} \mathbf{w}_{k,i} f(\|\mathbf{z}_i\|^2 - 2\mathbf{z}_i \star \mathbf{X} + \mathbb{1} \star \mathbf{X}^2) - \mathbf{Y}_0^k \tag{3.22}$$

The involved computations are only slightly more expensive than one single linear filtering of $\mathbf{X}$ with $\mathbf{z}_i$. One can precompute $\mathbb{1} \star \mathbf{X}^2$ by squaring every pixel of the image and filtering the resulting image $\mathbf{X}^2$ with $\mathbb{1}$. Then, one would add the scalar value $\|\mathbf{z}_i\|^2$ to every pixel and subtract the double of the filtered image $\mathbf{z}_i \star \mathbf{X}$.

Chapter 4

Simulation results

4.1 Outline

The following chapter presents numerical results obtained by systematical exploration of expanded SFA and kernelized SFA. The measure of empirical slowness served as main guideline. Another focus was the characterization of the learned filters and the associated features. Most of the experiments report results for kernelized SFA and establish empirical evidence for families of kernels, their parameters and selection of suitable support vectors.

Many figures are presented in the following chapter. This detailed description is made to enable the reader to fully follow the parameter exploration made in this thesis and to serve as a reference containing many little technical hints for further work in the NeuRoBot project.

Plugging the definitions of the statistics $\mathbf{A}$ and $\mathbf{B}$ for the kernelized case into the SFA objective function (2.16) yields the extended version of the kernelized optimization problem

$$\begin{aligned}(\mathbf{W}^*, \mathbf{Z}^*, k^*) &= \underset{\mathbf{W}, \mathbf{k}_t = \mathbf{k}(\mathbf{Z}, \mathbf{x}_t)}{\operatorname{argmin}} = \operatorname{tr}\left(\mathbf{W}^\top \sum_{t=1}^{T-1} (\mathbf{k}_{t+1} - \mathbf{k}_t)(\mathbf{k}_{t+1} - \mathbf{k}_t)^\top \mathbf{W}\right) \\ \text{s.t.} \quad \mathbf{I} &= \mathbf{W}^{*\top}\left(\sum_{t=1}^{T} \mathbf{k}_t^* \mathbf{k}_t^{*\top} - \sum_{t=1}^{T} \mathbf{k}_t^* \sum_{t=1}^{T} \mathbf{k}_t^{*\top}\right)\mathbf{W}^* \end{aligned} \tag{4.1}$$

by using $\mathbf{k}(\mathbf{Z}, \mathbf{x}_t) := [k(\mathbf{z}_1, \mathbf{x}_t), \ldots, k(\mathbf{z}_M, \mathbf{x}_t)]^\top$ with $\mathbf{Z}$ denoting the set of support vectors.

Inspection of (4.1) shows that only the input data $\mathbf{x}_t$ is given, the weights $\mathbf{W}$, the kernel mapping $k(\cdot, \cdot)$ and the set of support vectors $\mathbf{Z}$ are subject to optimization. Provided that $k^*(\cdot, \cdot)$ defines a reasonable similarity measure and that the support vectors $\mathbf{Z}^*$ are sufficiently distinct, the covariance matrices $\mathbf{A}$ and $\mathbf{B}$ will be well conditioned and thus the weights $\mathbf{W}^*$ can easily be obtained by SFA.

Hence, there is the choice of kernel function and the choice of support vector set to be made.

There is a tradeoff between incorporating knowledge into the kernel function and applying a preprocessing to the support vectors. Take for ex-

ample a dot product kernel $k(\mathbf{z}_i, \mathbf{x}_t) := f(\mathbf{z}_i^\top \mathbf{x}_t)$. In that case, transformation of the support vectors $\tilde{\mathbf{z}}_i := \mathbf{M}\mathbf{z}_i$ ($\mathbf{M} = \mathbf{M}^\top$) is equivalent to kernel modification $\tilde{k}(\mathbf{z}_i, \mathbf{x}_t) := f(\mathbf{z}_i^\top \mathbf{M}\mathbf{x}_t)$. In the remainder, one considers these transformations to be part of the kernel.

As there are two quantities to be optimized, one can adopt the strategy of keeping one quantity fixed whilst optimizing the other quantity. No joint optimization will be considered in this thesis.

4.1.1 Optimal kernel for a given set of support vectors

An exhaustive search in the space of kernels is computationally not tractable, therefore one focuses on some known parametrized families of kernels, whose parameters can efficiently be optimized by using cross-validation techniques.

That means, one deals with the problem of finding an optimal parameter θ in the context of a given family of kernel mappings $\mathcal{K} = \{\theta \in \mathbf{\Theta} \mid k_\theta(\mathbf{z}, \mathbf{x})\}$ and a given set of support vectors $\mathbf{Z}$. The following five families of kernels are taken into account:

1. $\mathcal{K}_{\text{exp}} = \left\{(\sigma, d) \in \mathbb{R}_+ \times (0, 2] \mid k_{(\sigma,d)}(\mathbf{z}, \mathbf{x}) = \exp\left(-\left(\frac{\|\mathbf{z}-\mathbf{x}\|}{\sqrt{2}\sigma}\right)^d\right)\right\}$
   Special cases include the Gaussian kernel ($d = 2$) and the Laplacian kernel ($d = 1$). For results see Figure 4.11 and Section 4.4.3. For some ideas on coverage see Appendix A.2.2.

2. $\mathcal{K}_{\text{plum}} = \left\{(\sigma, d) \in \mathbb{R}_+ \times \mathbb{R}_+ \mid k_{(\sigma,d)}(\mathbf{z}, \mathbf{x}) = \frac{\sigma^d}{\|\mathbf{x}-\mathbf{z}\|^d + \sigma^d}\right\}$
   The Plummer kernel is the second class of Radial Basis Function (RBF) kernels and has the advantage of not requiring the evaluation of a transcendent function which can mean a gain in simulation time. For results see Figure 4.11 and Section 4.4.3.

3. $\mathcal{K}_{\text{poly}} = \left\{(a, b, c) \in \mathbb{R}_+ \times \mathbb{R} \times \mathbb{N}_+ \mid k_{(a,b,c)}(\mathbf{z}, \mathbf{x}) = \left(\frac{\mathbf{z}^\top \mathbf{x}}{a} + b\right)^c\right\}$
   Widely used in image processing and in some respects equivalent to polynomial expansion (see Appendix A.2.1), the polynomial kernel is one of the studied dot product kernels. For results see Figure 4.11 and Section 4.4.1.

4. $\mathcal{K}_{\text{NN}} = \left\{(\kappa, \theta) \in \mathbb{R}_+ \times \mathbb{R} \mid k_{(\kappa,\theta)}(\mathbf{z}, \mathbf{x}) = \tanh\left(\kappa \mathbf{z}^\top \mathbf{x} + \theta\right)\right\}$
   In some performance aspects similar to Gaussian kernels, the neural net (or sigmoid) kernel is one of the often used dot product kernels. For results see Section 4.4.2.

5. $\mathcal{K}_{\text{sin}} = \left\{(a, \phi) \in \mathbb{R}_+ \times [0, 2\pi) \mid k_{(a,\phi)}(\mathbf{z}, \mathbf{x}) = \sin\left(\frac{\mathbf{z}^\top \mathbf{x}}{a} + \phi\right)\right\}$
   The theory of optimal free responses (see Section 2.2.3) motivates the use of the sine kernel which is also one of the commonly used dot product kernels, even if it is not positive definite. For results see Section 4.4.2.

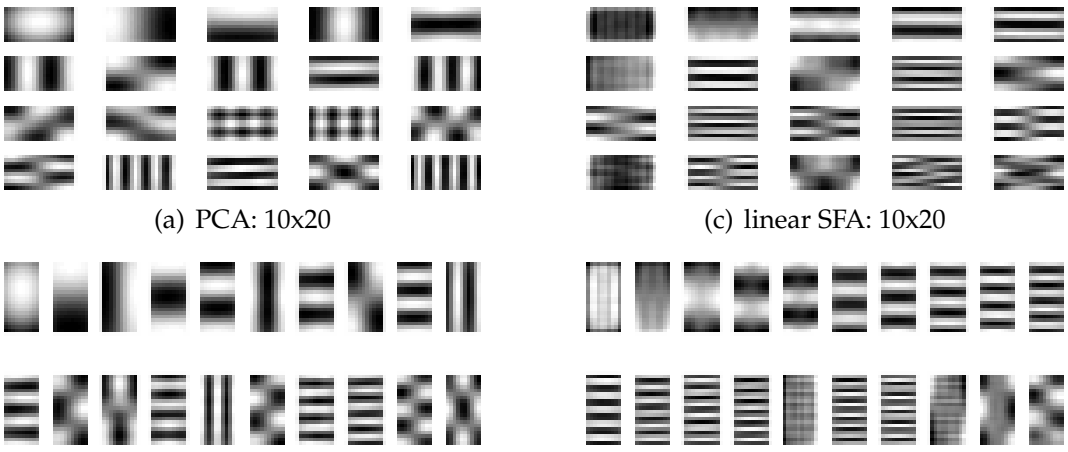
4.1.2 Optimal set of support vectors for a kernel

A somewhat converse approach consists of keeping the kernel k fixed and searching for optimal support vectors. From the representer theorem one knows that the optimal slowness function can be obtained as linear combination of the functions $k(\mathbf{x}_t, \cdot)$ for all training points $\mathbf{x}_t$. Choosing an optimal subset $\{\mathbf{z}_i\} \subset \{\mathbf{x}_t\}$ with a fixed number $M = |\{\mathbf{z}_i\}|$ of elements turns out to be a combinatorical optimization problem. On the other hand, one can completely detach from the training set and search for heuristics selecting good support vectors $\mathbf{z}_i \in \mathbb{R}^N$.

Here one makes the distinction between support vector selection prior to learning (see Section 4.3.2) and after learning (see Section 4.3.4) i.e. apply an algorithm or heuristic for support vector selection before calculating any image patch statistics and choosing good support vectors based on SFA, respectively.

4.2 Results for Expanded SFA

As detailed in Section 2.2.4, SFA can be solved by applying a whitening operation $\mathbf{P}$ in order to respect the constraints and in a second step to search for a rotation $\mathbf{R}$ that minimizes the objective function.



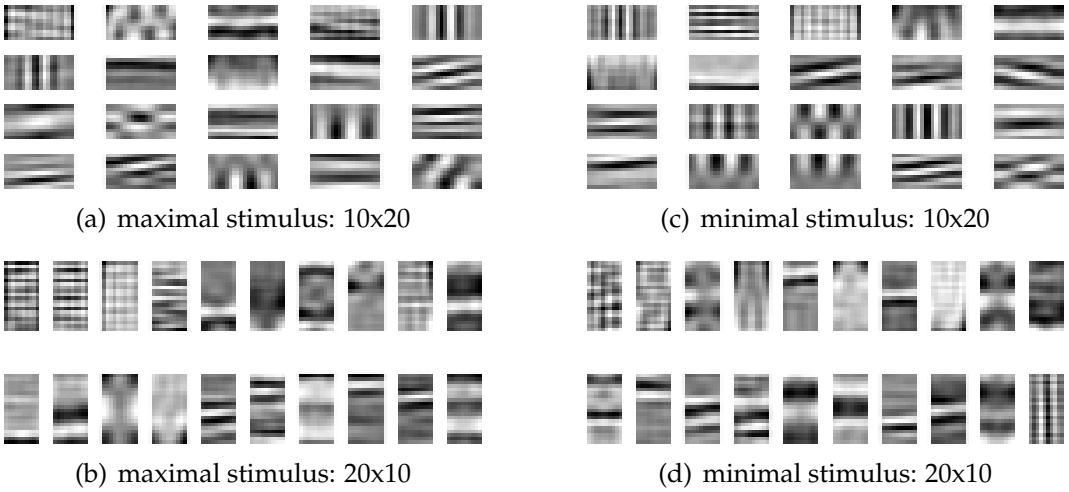
(a) PCA: 10x20 (c) linear SFA: 10x20

(b) PCA: 20x10 (d) linear SFA: 20x10

Figure 4.1: The figure shows the first 20 principal components and the corresponding 20 slowest linear feature detectors for horizontally and vertically elongated patches. In both cases, horizontal filters with increasing spatial frequencies constitute the slowest filters. Interestingly, the first slow filters are linear combinations of only a few principal components.

For visual comparison, the principal components of image patches as well as the slowest varying linear filters are shown in 4.1. Principal components of images are known to correspond to low-pass filters in different

orientations with increasing spatial frequencies. Analysis of the rotational part $\mathbf{R}$ reveals that the slowest varying linear features are linear combinations of very few principal components with large eigenvalues. That means the slowest linear features live in the subspace induced by a few low-pass filters, that capture most of the variance of the image patch. As a result, the slowest linear filters are spatial low-pass filters as well. Concluding one can say, that temporal low-pass filtering using instantaneous functions can be achieved by linear combination of spatial low-pass filters. The special form, i.e. filters constant in horizontal direction and periodic in vertical direction can be explained by the structure of the video where far more horizontal movement is present due to the structure of the rooms and the two dimensional trajectory of the robot.



(a) maximal stimulus: 10x20 (c) minimal stimulus: 10x20

(b) maximal stimulus: 20x10 (d) minimal stimulus: 20x10

Figure 4.2: The figure visualizes the first 30 quadratic slow features by the stimuli that generate minimal and maximal responses respectively. Again, horizontal structure dominates the stimuli.

Following Section 2.4.1, quadratically expanded SFA was applied to image patches. On the one hand, it is interesting to know if real world image transformations lead to filters that exhibit similar receptive field properties as those obtained from artificial transformation sequences on still images. Quadratic filters cannot be visualized as easy as linear filters. A common but very coarse way consists of showing the image patches leading to minimal or maximal output as done in Figure 4.2. Some filters look like Gabor wavelets with large spatial extent in one direction. Analysis of the invariances implemented by the filters uncovers that translation invariance can mainly be found in horizontal direction. This matches the intuition of the robot in the lab doing many rotational movements resulting in horizontal translation in the image plane.

An interesting difference between linear and quadratically expanded SFA is shown in Figure 4.3 where slowness is plotted as a function of receptive field size. Intuitively, larger receptive fields allow for more complex functions and therefore for slower responding output functions. On

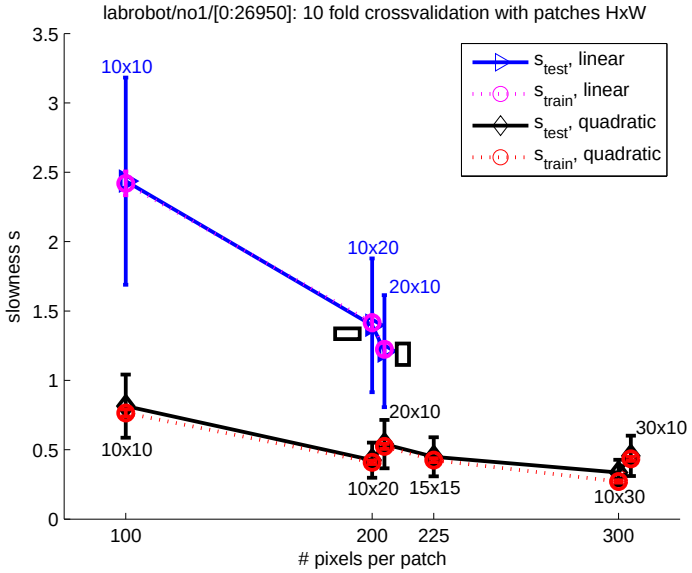


Figure 4.3: A number of $400,000$ *patches of sizes* 10×10, 10×20, 20×10, 15×15, 10×30 *and* 30×10 *was sampled from a video of the robot lab. Linear SFA and quadratic SFA based on the fully expanded 40 dominant principal components was performed and the first* $K = 10$ *slow features were extracted by using 10-fold cross-validation. Note the different behavior for horizontal and vertical patches with the same number of pixels.*

the other hand, adopting the spatial low-pass filter perspective, it is clear that larger receptive fields permit smoother spatial filters potentially being also temporal low-pass filters. However, there is a difference between horizontally elongated patches and vertically elongated patches. Linear SFA achieves slower outputs for vertically elongated patches which is probably due to the fact that these patches show less initial temporal variance and that the strategy of doing a spatial averaging in horizontal direction produces worse results. Quadratical filters can implement much more complex invariances such as phase invariance along the horizontal direction. Indeed, this is the case, and certainly explains the horizontal vertical flip in the figure, relating linear and quadratic SFA. A trivial result to state is the slower response of quadratic SFA compared to linear SFA. The test slowness s_{test} for 10×10 patches of $s_{test} = 0.82$ will serve as a reference value below. Quadratic expansion is not directly applied to the input data set because the dimensionality of the expanded space would be numerically less tractable than the quadratic expansion of the most dominant principal components. On the other hand, the whitening in input space improves the condition of the optimization problem. The horizontal-vertical flip is not an effect of the PCA preprocessing.

From the beginning, it is not clear, how many data points are necessary for a reliable estimation of a slow filter. This question is examined for quadratically expanded SFA in Figure 4.4. More data points produce

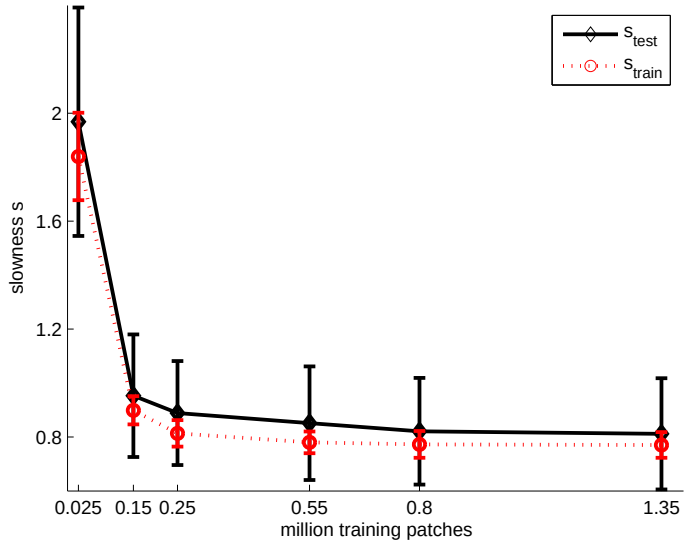


Figure 4.4: An increasing number of training samples leads to slower responses. Here patches of size 10×10 *were used to perform quadratic SFA to extract the* $K = 10$ *slowest feature in 5-fold cross-validation.*

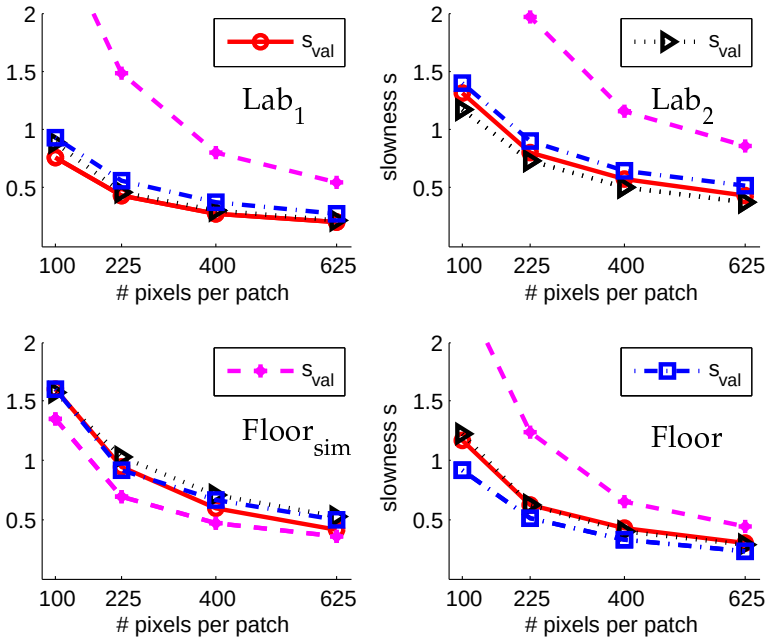


Figure 4.5: The figure displays slowness values obtained by training on one video and validating on another video. The simulator video has a behavior different from the camera videos. 1,000,000 patches of sizes 10×10, 15×15, 20×20 *and* 25×25 *were used as input to quadratic SFA keeping the* $K = 10$ *slowest components.*

slower filters. As a rule of thumb, depending on the patch size, one should at least have 500,000 data points; more points do no harm. More data points also cause the variance of the estimator to shrink and decrease the distance between test value and the training value.

The question arises, up to what extent filters learned on a special video produce slow outputs for another video. Motivated by the availability of a simulator, one has to examine the possible difference between artificially generated image sequences and recorded videos. As depicted in Figure 4.5 there is a remarkable difference between these two categories of images. Training videos are depicted in Figure 3.1. If training is done on camera recorded videos, the filters always perform bad on the simulator video. If one trains on the artificial video, the slowness on the camera grabbed videos is in the same range as on the artificial video which is less slow than if trained on the real world video. This gives rise to the conclusion that the transformations occurring in the simulator video are a superset of the transformations in the real world dataset. Additional transformations such as texture mapping effects can influence the performance of the filters. Admittedly, the gap between the slownesses of the two classes of videos becomes smaller for larger patches.

4.3 Selection of Support Vectors

As the number of support vectors M is a multiplicative factor in the complexity of the filtering of large data-sets $\mathcal{O}(M \cdot Rest)$ it is desirable to keep the number of support vectors as small as possible i.e. to select the best support vectors beforehand (Section 4.3.2) or to perform the expanded analysis with many support vectors and find a good subset afterwards (Section 4.3.4).

4.3.1 Support vectors from data set or not

In SV classification or regression, e.g. SVMs, the support vectors i.e. the set of data points that constitute the solution naturally stem from the training set. The representer theorem guarantees that the optimal weight vectors $\mathbf{u}_j$ in feature space lie in the span $\mathcal{L}\{\mathbf{\Phi}(\mathbf{x}_t)\}$ of the data points in feature space.

That means one could choose $\{\mathbf{z}_i\} = \{\mathbf{x}_t\}$ as support vector set. In the case of SFA one typically deals with samples of $500,000$ data points or more which is not a computationally tractable set of support vectors. Driven by the hope that the data points $\{\mathbf{x}_t\}$ do not have linearly independent images in feature space, one can sample a random subset of the data $\{\mathbf{z}_i\} \subseteq \{\mathbf{x}_t\}$ to serve as support vectors. In the case of RBF kernels, that depend on local distances between data points, this intuitively seems to be a good idea. Numerical simulations revealed that slowest responses are obtained for support vectors originating from the data set. Distances in RBF kernels are usually measured as Euclidean distances:

$$d(\mathbf{x}, \mathbf{y}) = \|\mathbf{z} - \mathbf{x}\| = \sqrt{(\mathbf{z} - \mathbf{x})^\top (\mathbf{z} - \mathbf{x})} \tag{4.2}$$

Intuitively that means, distances are dominated by deviations along the principal components of the data. One can suspect that a metric taking the shape of the data cloud into account would make a difference. Distances in whitened space are measured by:

$$d_{\mathbf{C_x}}(\mathbf{z}, \mathbf{x}) = \sqrt{(\mathbf{z} - \mathbf{x})^\top \mathbf{C}_\mathbf{x}^{-1} (\mathbf{z} - \mathbf{x})} \tag{4.3}$$

However, experiments conducted with that PCA preprocessing produced less slow results than in the usual Euclidean case.

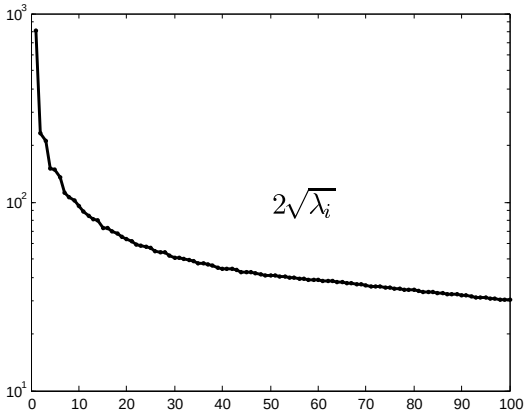


Figure 4.6: The double square roots of the eigenvalues λ_i of the covariance matrix of image patches of size 10×10 are semi-logarithmically plotted in order to illustrate the shape of the data manifold. As the patches have values in the range of $[0, 255]$, the units of the figure are gray value differences. The longest axis of the corresponding hyperellipsoid has the length $2\sqrt{\lambda_1} = 815$, the second longest $2\sqrt{\lambda_2} = 220$ and the shortest axis has a length of around $2\sqrt{\lambda_{100}} = 30$.

Nevertheless, dot product kernels $k(\mathbf{z}, \mathbf{x}) = f(\mathbf{z}^\top \mathbf{x})$ allow for support vectors $\mathbf{z}_i$ that are scaled versions of training points and thus do not originate from the data distribution. One can suppose that both the support vector $\mathbf{z}$ and the data point $\mathbf{x}$ originate from the data distribution approximated by a hyperellipsoid with axes of lengths $2\sqrt{\lambda_i}$ where λ_i is the ith eigenvalue of the covariance matrix. From Figure 4.6 one sees that the spectrum of the covariance matrix rapidly decays, which means that there are a few dominant eigenvalues and many very small eigenvalues. In other words, two vectors stemming from such an elongated hyperellipsoid (one can think of it as in shape of a high-dimensional cigar) are almost collinear and point either in the same or the opposite direction. Thus, the dot product has a large absolute value and can be considered as constant plus a little noise which induces very badly conditioned covariance matrices. PCA as preprocessing can solve this problem and provide more meaningful dot products in terms of similarity measurement. Nevertheless, by empirical simulations it turned out that drawing support vectors from a standard normal distribution $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ produces significantly slower response functions. An intuitive explanation is given by the following reasoning: The dot product is a

projection operation of the data point $\mathbf{x}$ onto the the support vector $\mathbf{z}$. That means the kernel is most sensitive to deviations of $\mathbf{x}$ in direction of $\mathbf{z}$. Functional complexity with respect to shattering of point sets is consequently present only in direction of $\mathbf{z}$. Directions orthogonal to $\mathbf{z}$ do not change the dot product. Finally, the slowness function, being a linear combination of kernel mappings evaluated at the support vectors, will have complexity in direction of the support vectors. From the empirical observation, that random support vectors produce slower responses, one can deduce that it seems to be a good strategy to select support vectors pointing into as many different directions as possible. In other words complexity should best be distributed in an isotropic way.

4.3.2 Selection of support vectors prior to learning

Clustering in feature space

A rule of thumb builds on the intuition that support vectors should be distinct to a certain degree. Otherwise kernel matrices and covariance matrices would not be of full rank which could give rise to numerical problems. If support vectors originate from the training set, one can also tend to favor support vector sets which – mapped into feature space – cover the feature space in an optimal way. A feature vector selection algorithm equivalent to clustering in feature space is proposed by Bray and Martinez [2002].

The basic idea is the following:

Given T training points $\mathbf{x}_t$, the kernel matrix $\mathbf{K} = [k(\mathbf{x}_\tau, \mathbf{x}_t)]$ has often a rank much smaller than T. This means that the manifold in the (potentially very high-dimensional) feature space has a low dimension and can therefore be approximated by a subset $\{\tilde{\mathbf{\Phi}}_s\}$ of the support vectors $\mathbf{\Phi}_t \approx \sum_{s=1}^{S} \alpha_{t,s} \tilde{\mathbf{\Phi}}_s$ forming an approximate basis of the relevant feature subspace.

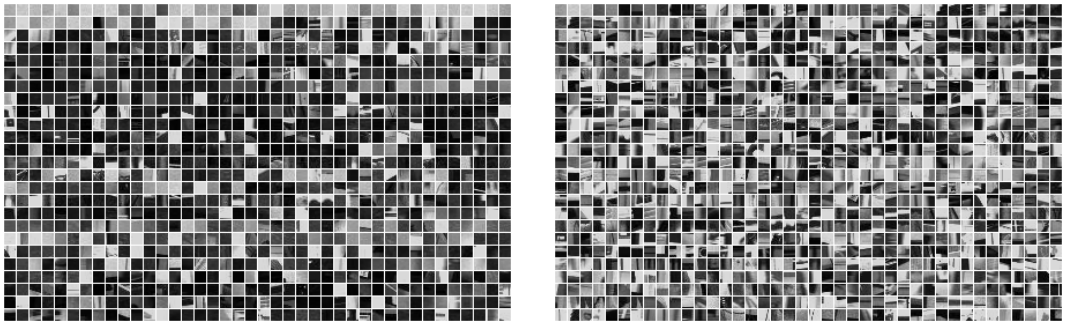
The relative reconstruction error $\left\| \mathbf{\Phi}_t - \sum_{s=1}^{S} \alpha_{t,s} \tilde{\mathbf{\Phi}}_s \right\| / \|\mathbf{\Phi}_t\|$ is minimized which yields the feature vector selection algorithm detailed in [Baudat and Anouar, 2001] and rewritten as Algorithm 1. In a nutshell, feature or support vectors are selected having a minimum distance in feature space to each other.

Algorithm 1 *Feature Vector Selection (FVS)*

```
init FV ← ∅,  τ_act ← τ_0,  λ < 1
for ∀x ∈ X and while |FV| < N_SV
    if ∀z ∈ FV  |k(z, x)| < τ_act
      • FV ← FV ∪ {x}          (add new vector)
      • τ_act ← λ τ_act              (annealing)
    end if
end for
```

In the case of RBF kernels $k(\mathbf{z}, \mathbf{x}) = f(\|\mathbf{z} - \mathbf{x}\|)$, depending directly on a

distance measure in input space, the FVS algorithm corresponds to clustering in input space as $k(\mathbf{z}, \mathbf{x}) < \tau \Leftrightarrow f^{-1}(\tau) = \tilde{\tau} < \|\mathbf{z} - \mathbf{x}\|$ assuming strictly decreasing f and dropping the absolute value.



(a) $N_{SV} = 1000, 10 \leq d(\mathbf{z}_1, \mathbf{z}_2)$ (b) $N_{SV} = 1000, 200 \leq d(\mathbf{z}_1, \mathbf{z}_2)$

Figure 4.7: Clustering by using the selection rule from Algorithm 1 with RBF kernels is equivalent to clustering using a threshold $\tilde{\tau}$ on the pairwise distance between two data points e.g. $\tilde{\tau} = -\sqrt{2}\sigma \ln \tau < d(\mathbf{z}, \mathbf{x})$ in the case of Laplacian kernels.
The threshold for the left example is low, which means the support vectors are distributed like the data. Whereas the threshold for the right example is quite large which means the support vectors are much broader distributed than the data and show thus a larger variety.

The sampling of support vectors also reveals something about the structure of the data points. In Figure 4.7 two different thresholds $\tilde{\tau}$ were used for support vector selection prior to learning. The left image contains many quasi-constant image patches occurring frequently in the video. Higher thresholds $\tilde{\tau}$ require the support vectors to be more distinctive what is obviously reflected by the left image.

Support vector selection is not interesting as such in this thesis; it serves only as a heuristic to improve the results of kernelized SFA. In Figure 4.8 the slight improvement of slowness is depicted. Different thresholds τ on the kernel function were used to select support vectors beforehand. Then kernelized SFA employing a Gaussian kernel was applied in order to study the slowness behavior. Summarizing, one can say that a slight distinctness of support vectors improves slownesses. However, with growing numbers of support vectors, this effect becomes less prominent. For further results see Section 4.3.4 and Figure 4.10.

4.3.3 Selection of support vectors during learning

In this section, two results are briefly mentioned for reasons of completeness, although no further investigation of the proposed methods was done in the context of kernelized SFA.

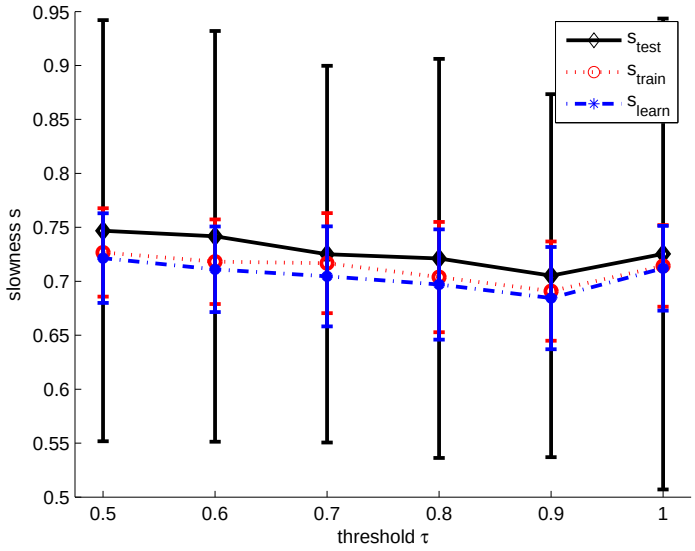


Figure 4.8: A 5-fold cross-validation experiment using a Gaussian kernel with $\sigma = 500$ based on $T = 600,000$ training samples and $N_{SV} = 200$ support vectors was conducted in order to show that clustering in feature space slightly influences the resulting slowness values for the $K = 10$ slowest components.

Modification of the constraints

Among the approaches for obtaining sparse solutions for unsupervised kernel algorithms, Smola et al. [1999] replace the constraints in feature space by constraints in coefficient space. As a result, the admissible expansion coefficients are bounded by a polyhedron, which leads to vertex solutions that are naturally sparse.

Approximation of the covariance in feature space

Sparse kernelized PCA as proposed by Tipping [2000] is achieved by an approximation of the covariance matrix of the data centered in feature space with a subset of weighted data points.

$$\hat{\text{cov}}(\mathbf{\Phi}(\mathbf{x}_t)) = \frac{1}{T-1}\sum_{t=1}^{T}\mathbf{\Phi}_t\mathbf{\Phi}_t^\top \approx \sigma^2\mathbf{I} + \sum_{i=1}^{N_{SV}} w_i\mathbf{\Phi}(\mathbf{z}_i)\mathbf{\Phi}(\mathbf{z}_i)^\top \tag{4.4}$$

Therefore, only N_{SV} support vectors are involved in the solution instead of the whole dataset containing T points.

4.3.4 Selection of support vectors after learning

Results reported in [Vollgraf and Obermayer, 2006] suggest that, in line with kernel PCA, given the kernel covariance matrix for a large set of sup-

port vector candidates, one can apply a retroactive selection to these candidates by using a regularization term that punishes solutions containing many support vectors. The initial objective function (variance conserved or reconstruction error) is augmented with a regularization term $\rho(\mathbf{v})$ which favors sparse weight vectors $\mathbf{v}$ and provides a means of candidate elimination. The optimization is done with an iterative algorithm called HECGD, which is basically a minimization routine operating on the admissible set of weight vectors in shape of a hyperellipsoid. An implementation is provided in the file `ksfmin.m` (for details see Section 3.1.3)

Column sparseness

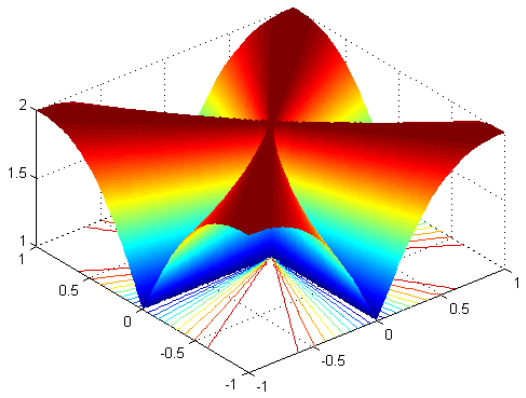


Figure 4.9: For the case of dimension $M = 2$ the sparseness regularizer $\rho(\mathbf{v}) = \|\mathbf{v}\|_1^2 / \|\mathbf{v}\|_2^2$ has the above shape. It can be seen that regions close to the coordinate axes have small values which corresponds to the intuition of as many zero coordinates in $\mathbf{v}$ as possible.

Normally, in SFA one minimizes (2.17) to obtain the first K slow components. If additional prior knowledge shall be incorporated in the solution, one can replace the objective term $s(\mathbf{W})$ by an augmented objective term $s(\mathbf{W}) + \lambda\rho(\mathbf{W})$. Here, $\rho(\mathbf{W})$ is a regularization term that is low if $\mathbf{W}$ matches the prior knowledge. Solutions not compatible with prior knowledge are punished by large values of $\rho(\mathbf{W})$. The scalar value λ interpolates between the primal objective and the regularization term. Regularization can often improve the conditioning of ill-posed problems. A known regularizer enforcing sparseness is the ratio between the L_1- and the L_2-norm:

$$\rho(\mathbf{v}) = \frac{\|\mathbf{v}\|_1^2}{\|\mathbf{v}\|_2^2} = \frac{\left(\sum_{i=1}^{N} |v_i|\right)^2}{\mathbf{v}^\top\mathbf{v}} = \frac{\left(\mathbf{1}^T |\mathbf{v}|\right)^2}{\mathbf{v}^\top\mathbf{v}} \qquad \mathbf{v} \neq \mathbf{0} \tag{4.5}$$

The function is scale invariant i.e. $\rho(\alpha\mathbf{v}) = \rho(\mathbf{v})$ and special function values comprise the coordinate axes $\mathbf{e}_i = (0, \ldots, 1, \ldots 0)^\top$ with $\rho(\mathbf{e}_i) = 1$ and the hyperdiagonals $\mathbf{d}_h \in \{-1, 1\}^M$ with $\rho(\mathbf{d}_h) = M$. The limit of $\rho(\mathbf{v})$ for

$\mathbf{v} \to \mathbf{0}$ does not exist.

$$\frac{\partial \rho}{\partial \mathbf{v}} = 2\frac{\|\mathbf{v}\|_1}{\|\mathbf{v}\|_2^4}\left[\|\mathbf{v}\|_2^2 \operatorname{sign}(\mathbf{v}) - \|\mathbf{v}\|_1 \mathbf{v}\right] \tag{4.6}$$

The gradient equals $\mathbf{0}$ for $\mathbf{v} \neq \mathbf{0}$ only if $\|\mathbf{v}\|_2^2 \operatorname{sign}(\mathbf{v}) - \|\mathbf{v}\|_1 \mathbf{v}$. Consequently, vectors satisfying $\sum_i v_i^2 = \sum_i |v_i v_j|$ for all j are of special interest. This is the case for the coordinate axes and the hyperdiagonals as $\frac{\partial \rho}{\partial \mathbf{v}}(\mathbf{e}_j) = \mathbf{0}$ and $\frac{\partial \rho}{\partial \mathbf{v}}(\mathbf{d}_h) = \mathbf{0}$.

In Figure 4.9 one can see the regularizer $\rho(v)$ for the case of dimension $M = 2$. The function has local saddle point minima along the coordinate axes, signifying a zero entry for a special support vector. Furthermore, the function has saddle point maxima along the diagonals. Non-zero elements of $\mathbf{v}$ are punished, so an optimal $\mathbf{v}$ contains as many zeros as possible.

Sparse solutions with the help of $\rho(\mathbf{v})$ can iteratively be obtained in the following way: Assuming, one already has a partial solution consisting of $L < K$ vectors $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_L]$. The next vector $\mathbf{w}_{L+1} =: \mathbf{v} \in \mathbb{R}^M$ is found by minimizing $\mathbf{v}^\top \mathbf{A}\mathbf{v} + \lambda\rho(\mathbf{v})$ under the constraints $\mathbf{v}^\top \mathbf{B}\mathbf{v} = 1$ and $\mathbf{W}^\top \mathbf{B}\mathbf{v} = \mathbf{0}$. An efficient way to do so is HECGD, as the algorithm implements a gradient descent which by construction respects the quadratic constraints on $\mathbf{v}$. If the method is iteratively applied for $L = 0, 1, \ldots, M-1$, an optimal weight matrix $\mathbf{W}$ can be constructed. Sparse weight vectors $\mathbf{w}_i$ are not of special value as long as they do not have zeros in the same row for all columns. If one thinks of applying the filters $g_i(\mathbf{X}) = \sum_j w_{ji} k(\mathbf{z}_j, \mathbf{X})$ to an image $\mathbf{X}$, speed improvement is only obtained when coefficients w_{ij} are zero for a special row j and all columns i implying that the quantity $k(\mathbf{z}_j, \mathbf{X})$ does not need to be computed.

Joint sparseness

Generalization of the sparseness regularizer (4.5) to matrix arguments can be done in the following way:

$$\begin{aligned}\rho(\mathbf{W}) &= \frac{\text{squared sum of Euclidean row-norms}}{\text{Frobenius matrix norm}} = \frac{\left(\sum_i \left\|\mathbf{w}_{[i,:]}\right\|_2\right)^2}{\|\mathbf{W}\|_2^2} \\ &= \frac{\left(\sum_i \left\|\mathbf{w}_{[i,:]}\right\|_2\right)^2}{\sum_i \left\|\mathbf{w}_{[i,:]}\right\|_2^2} = \frac{\left(\sum_i \sqrt{\sum_{j=1}^n w_{ij}^2}\right)^2}{\sum_i \sum_{j=1}^n w_{ij}^2} = \frac{\left(\operatorname{tr}\sqrt{\mathbf{W}\mathbf{W}^\top}\right)^2}{\operatorname{tr}(\mathbf{W}\mathbf{W}^\top)}\end{aligned} \tag{4.7}$$

The vector valued regularizer (4.5) emerges as a special case of (4.7) if $\mathbf{W}$ contains only one single column and the ith row $\mathbf{w}_{[i,:]}$ reduces to a scalar value w_i. Note that the regularization term $\rho(\mathbf{W}) = \rho(\alpha\mathbf{W})$ is invariant to the scale of $\mathbf{W}$ which guarantees that the solution is not biased to small or large values. The gradient of $\rho(\mathbf{W})$ is given by:

$$\frac{\partial \rho}{\partial \mathbf{W}} = 2\xi\left(\operatorname{dg}^{-\frac{1}{2}}\left(\mathbf{W}\mathbf{W}^\top\right) - \xi\mathbf{I}\right)\mathbf{W} \qquad \text{with} \qquad \xi = \frac{\operatorname{tr}\sqrt{\mathbf{W}\mathbf{W}^\top}}{\operatorname{tr}(\mathbf{W}\mathbf{W}^\top)} \tag{4.8}$$

For a derivation see Appendix (A.5). Geometrically, the gradient $\frac{\partial \rho}{\partial \mathbf{W}} = \mathbf{DW}$ is obtained by rescaling the rows of $\mathbf{W}$ with a diagonal matrix $\mathbf{D}$ with entries $d_{ii} = 2\xi \left\|\mathbf{w}_{[i,:]}\right\|_2^{-1} - 2\xi^2$ such that $\frac{\partial \rho}{\partial w_{ij}} = d_{ii} w_{ij}$. Considering formally the length l of the partial derivative of a column, one finds

$$l = \left\| \frac{\partial \rho}{\partial \mathbf{w}_{[i,:]}} \right\|_2 = \left\| d_{ii} \mathbf{w}_{[i,:]} \right\|_2 = 2\xi \left| 1 - \xi \left\| \mathbf{w}_{[i,:]} \right\|_2 \right|$$

The length l vanishes for $\left\|\mathbf{w}_{[i,:]}\right\|_2 = \xi^{-1}$. For example, the M-dimensional unit matrix $\mathbf{W} = \mathbf{I}$ yields a local maximum of $\rho(\mathbf{I}) = M$. The gradient also vanishes for matrices $\alpha \mathbb{1}$ with constant entries α. In numerical simulations, the convention

$$\left. \frac{\partial \rho}{\partial \mathbf{w}_{[i,:]}} \right|_{\mathbf{0}} := \mathbf{0}$$

is used because the limit

$$\lim_{\mathbf{w}_{[i,:]} \to \mathbf{0}} \frac{\partial \rho}{\partial \mathbf{w}_{[i,:]}}$$

does not exist.

The modified minimization problem now reads as follows

$$\text{min. } s(\mathbf{W}) + \lambda \rho(\mathbf{W}) \quad \text{s. t.} \quad \mathbf{W}^\top \mathbf{B} \mathbf{W} = \mathbf{I} \tag{4.9}$$

with regularization parameter λ. Obviously, the eigenproblem structure is lost. A mathematically correct treatment would involve an optimization of $s(\mathbf{W}) + \lambda \rho(\mathbf{W})$ on the manifold

$$\mathbf{S_B} := \left\{ \mathbf{W} \in \mathbb{R}^{M \times K} \mid \mathbf{W}^\top \mathbf{B} \mathbf{W} = \mathbf{I} \right\} \tag{4.10}$$

of matrices diagonalizing $\mathbf{B}$. By a substituting $\tilde{\mathbf{W}} \leftarrow \mathbf{B}^{-\frac{1}{2}} \mathbf{W}$ the problem can be transformed into an optimization of an objective function restricted to the Stiefel manifold:

$$\mathbf{S} := \left\{ \tilde{\mathbf{W}} \in \mathbb{R}^{M \times K} \mid \tilde{\mathbf{W}}^\top \tilde{\mathbf{W}} = \mathbf{I} \right\} \tag{4.11}$$

Methods for dealing with that optimization by means of a proper gradient defined on that manifold are extensively discussed by Edelman et al. [1999]. Even conjugate gradients are provided. In the context of ICA, orthogonal matrix gradient methods have already been studied by Plumbley [2004] and Fiori [2002].

While experimenting with gradient descent on Stiefel manifolds it turned out that an extension of the HECGD optimization scheme from Vollgraf and Obermayer [2006] as detailed in Algorithm (2) works faster and produces better results. The method was originally defined for vector-valued objective functions, but can easily be extended to matrix-valued functions if HECGD is iterated over the columns of the matrix $\mathbf{W}$.

Algorithm 2 *Joint Hyper-Elliptical Conjugate Gradient Descent*

```
init W ← N(0, ε²),  W ← W (W^⊤ B W)^{1/2}
repeat
    for ∀w_i ∈ W do
      • w_i^new ← HECGD(w_i, W_{¬i}, A, B) (column optimization)
    if ‖w_{[i,:]}‖ < τ_row
      • w_{[i,:]} ← 0^⊤                    (set row to zero)
until ‖w_i − w_i^new‖ < τ_ΔW  ∀i
```

As partial derivatives of the columns of $\mathbf{W}$ serve the columns of the gradient in Euclidean space given by:

$$\frac{\partial s + \lambda\rho}{\partial \mathbf{W}} = 2\left(\mathbf{A} + \lambda\xi \mathrm{dg}^{-\frac{1}{2}}\left(\mathbf{W}\mathbf{W}^\top\right) - \lambda\xi^2\mathbf{I}\right)\mathbf{W}, \quad \xi = \frac{\mathrm{tr}\sqrt{\mathbf{W}\mathbf{W}^\top}}{\mathrm{tr}(\mathbf{W}\mathbf{W}^\top)}$$

For the derivation of the gradient see Appendix A.1.1. An implementations is provided in the file `ksfmin2.m` (for details see Section 3.1.3)

Greedy approaches

There is a less fancy but very effective method to select a set of support vectors $\{\mathbf{z}_i\} \subset \mathbf{Z}$ after the statistics $\mathbf{A}$ and $\mathbf{B}$ have been computed. Let $\mathcal{I}$ denote the index set of the vectors chosen from $\mathbf{Z}$ such that $\{\mathbf{z}_i\} = \mathbf{Z}_\mathcal{I}$. Then the "reduced" matrices $\mathbf{A}_\mathcal{I}$ and $\mathbf{B}_\mathcal{I}$ are obtained by discarding the rows and columns from $\mathbf{A}$ and $\mathbf{B}$ whose indices are not in $\mathcal{I}$. By applying SFA to the reduced matrices $\mathbf{A}_\mathcal{I}$ and $\mathbf{B}_\mathcal{I}$ and extracting the first K slowest components, one obtains a slowness $s(\mathcal{I})$ depending on the support vectors chosen and producing a measure for comparing different index sets $\mathcal{I}$.

That means, a Greedy strategy of adding and deleting candidate support vector indices from $\mathcal{I}$ can be adopted.

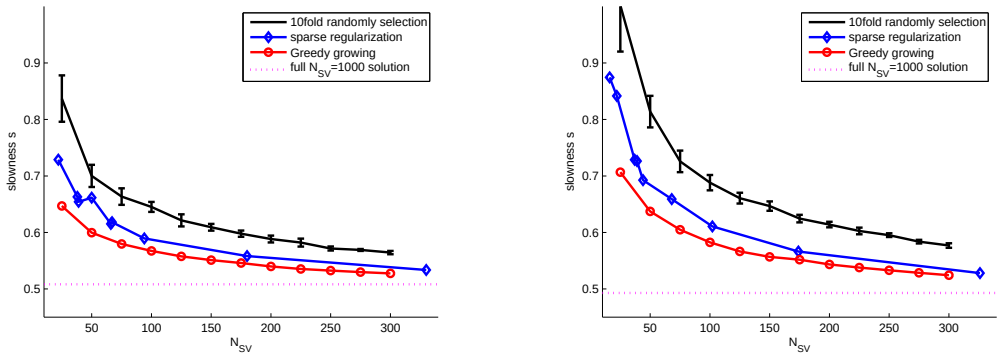
In Figure 4.10 the JHECGD regularization algorithm is compared to the Greedy selection approach. It turns out that both procedures perform better than randomly selected support vectors and that Greedy selection significantly outperforms the JHECGD algorithm. Furthermore, one can see that the FVS algorithm in combination with support vector selection afterwards produces worse results.

The final conclusion to draw is the following: In order to obtain sparse solutions, select via FVS with a small threshold τ a corpus of candidate support vectors prior to learning. Then, calculate the statistics $\mathbf{A}$ and $\mathbf{B}$ and apply Greedy support vector selection afterwards.

4.4 Results for Kernelized SFA

4.4.1 Polynomial kernels

Results with expanded SFA reported in the literature were obtained by applying an explicit expansion into the space of polynomials of degree two or



(a) Laplacian kernel $\tau = 10$ (b) Laplacian kernel $\tau = 200$

Figure 4.10: Initially, $N_{SV} = 1000$ support vectors (patches of size 10×10) were chosen according to Algorithm 1 with two different thresholds τ for the minimum distance between two support vectors.
Laplacian kernels of width $\sigma = 1200$ were used for the analysis. The figure shows results for three sparsification methods: Simple random removal of support vectors, regularization based support vector selection and Greedy support vector selection.

higher. Therefore, polynomial kernels are the most natural starting point for a comparative analysis. For equivalence considerations between explicitly expanded SFA and kernelized SFA see Appendix A.2.1.

There are two versions of widely used polynomial kernels, the homogeneous kernel

$$k(\mathbf{z}, \mathbf{x}) = \left(\frac{\mathbf{z}^\top \mathbf{x}}{a} \right)^c \tag{4.12}$$

containing monomials of order c and the inhomogeneous kernel

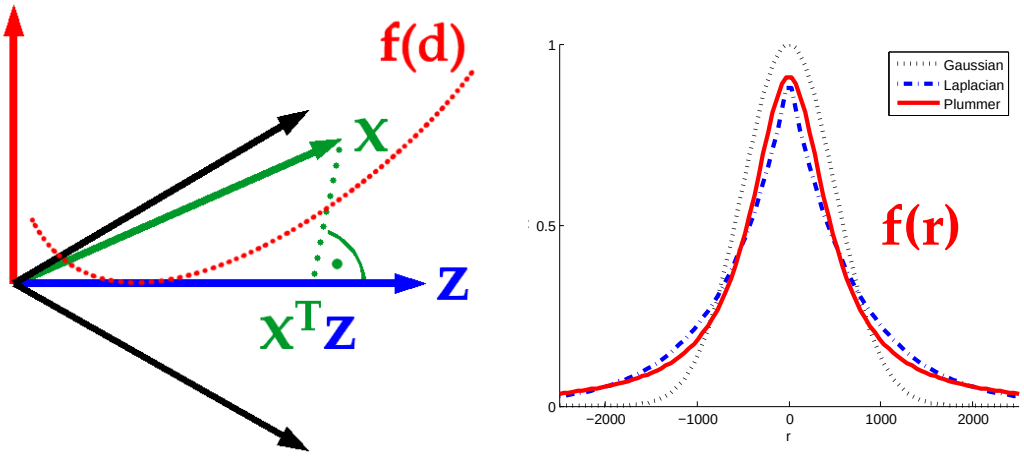
$$k(\mathbf{z}, \mathbf{x}) = \left(\frac{\mathbf{z}^\top \mathbf{x}}{a} + b \right)^c \tag{4.13}$$

containing monomials of order up to c. Geometrically, the polynomial kernel computes a projection and than applies a polynomial function along $\mathbf{z}$ as shown in Figure 4.11(a).

To get an intuition for the parameters a, b and c, some general considerations are made in the following. The offset b and the scale a have a similar effect, because the kernel can be equivalently rewritten as:

$$k(\mathbf{z}, \mathbf{x}) \propto \left(\frac{\mathbf{z}^\top \mathbf{x}}{ab} + 1 \right)^c \propto \left(\mathbf{z}^\top \mathbf{x} + ab \right)^c \tag{4.14}$$

In general, large values ab compared to the dot product $\mathbf{z}^\top \mathbf{x}$ lead in the limit to the constant value $k(\mathbf{z}, \mathbf{x}) \to b^c$ and small valued ab result in the homogeneous kernel. That means, the scale of ab determines in a certain sense how nonlinear the mapping is. The power c defines the maximum degree of the polynomials. If c is set too small, functions of low complexity with



(a) $k(\mathbf{z}, \mathbf{x}) = f(\mathbf{z}^\top \mathbf{x}) = f(d)$ (b) $k(\mathbf{z}, \mathbf{x}) = f(\|\mathbf{z} - \mathbf{x}\|) = f(r)$

Figure 4.11: Among the kernel functions $k(\mathbf{z}, \mathbf{x})$ there are two widely used families of special kernels: Dot product kernels depend only on a projection of $\mathbf{x}$ onto the support vector $\mathbf{z}$ whereas RBF kernels depend only on the distance of $\mathbf{x}$ to the support vector $\mathbf{z}$.
RBF kernels differ in the size of their support. Gaussian kernels are very local, Laplacian kernels have "heavier tails" and the Plummer kernel has the largest range of influence. All three kernels plotted here have a length scale parameter $\sigma = 500$.

tendency towards underfitting emerge and when c is too large, overfitting might occur. Furthermore, c takes as values only natural numbers to keep the kernel values real.

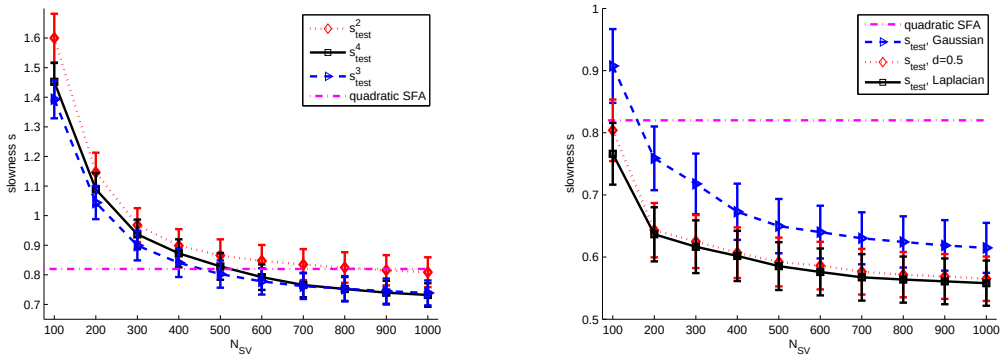
Figure 4.13 illustrates the influence of the kernel parameters and the number of support vectors on the slowness of the learned filters. Starting from a baseline experiment with $(a, b, c) = (500, 1, 3)$ and 100 support vectors, these four parameters were altered. Larger scale a and bigger offset b lead to slower responses. For only a few support vectors, quadratic kernels perform best and more support vectors lead to smaller slowness values. Slowness will increase again for larger values of ab.

As depicted in Figure 4.12(a), in the case of patches of size 10×10 pixels lying in the 8bit gray value range of $[0, 255]$, suitable kernel parameters are $(a, b, c) = (500, 5, 3)$. Around $N_{SV} = 500$ support drawn from $\mathcal{N}(\mathbf{0}, \mathbf{I})$ lead to slowness performance as good as quadratically expanded SFA.

Different heuristics for the choice of support vectors were considered. For details see Section 4.3.1.

4.4.2 Other dot product kernels

One can think of many pointwise nonlinearities $f(\cdot)$ applied to a dot product and yielding a reasonable kernel $k(\mathbf{z}, \mathbf{x}) = f(\mathbf{z}^\top \mathbf{x})$. In the following two short sections, numerical results for two more dot product kernels are pre-



(a) Polynomial kernel $s(N_{SV})$ (b) RBF kernel $s(N_{SV})$

Figure 4.12: 10-fold cross-validation on 10×10 *patches using polynomial and RBF kernels is used to compare slowness performance of the different kernels in the case of a growing number of support vectors. For reference, the slowness value of quadratically expanded SFA is given in both cases. Kernel parameters for polynomial kernels were* $(a, b) = (500, 5)$ *and* $c \in \{2, 3, 4\}$*. Kernelized SFA with quadratic kernels reaches the quadratical expansion bound for around* $N_{SV} = 700$ *support vectors. Cubical and fourth-order kernelized SFA show similar asymptotic behavior.*
RBF kernels worked with $\sigma = 1200$ *for* $d \in \{\frac{1}{2}, 1\}$ *and* $\sigma = 500$ *for* $d = 2$*. Both sub-Gaussian kernels show equivalent asymptotic behavior.*

sented, even though both of them are not positive definite.

Sine kernel

Besides algebraic polynomials serving as approximations for arbitrary functions, trigonometric polynomials can be used to approximate periodic functions. A more heuristic motivation comes from the theory of optimal free responses (see Section 2.2.3) where trigonometric functions are found to be the slowest possible functions. The sine kernel

$$k(\mathbf{z}, \mathbf{x}) = \sin(\frac{\mathbf{z}^\top \mathbf{x}}{a} + \phi) \tag{4.15}$$

implements periodic waves along the direction of $\mathbf{z}$ characterized by a wavelength $\lambda = 2\pi \left|a\right| / \left\|\mathbf{z}\right\|$ and an offset from the origin $\upsilon = \left|a\right| \phi / \left\|\mathbf{z}\right\|$. Fixing the length of the support vectors to $\|\mathbf{z}\| = 1$ allows for convenient modeling of λ with the help of the parameter a and for adjustment of ν by means of choosing the parameter $\phi \in [0, 2\pi)$.

As shown in Figure 4.14, the results are a bit slower than in the polynomial case but do not reach RBF performance. One can clearly see a symmetry in the shift parameter ϕ. The support vectors were randomly chosen from $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and the same shift ϕ was used for all support vectors.

A drawback of trigonometric functions could be their non-locality. It might happen that beating or interference effects occur when many waves

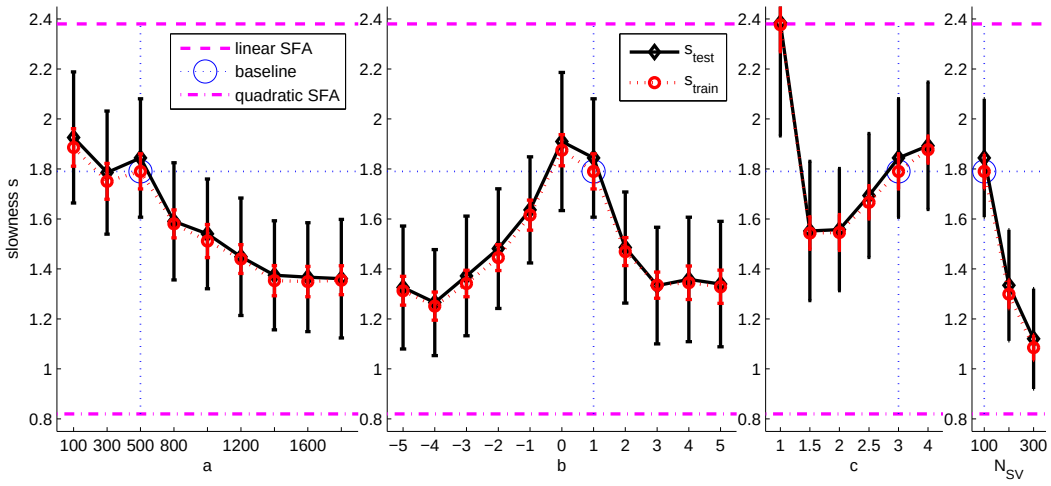


Figure 4.13: Starting from a baseline experiment ($a = 500$, $b = 1$, $c = 3$, $N_{SV} = 100$) the changes in the slowness function are explored for these four kernel parameters. Concluding from this 10-fold cross-validation experiment using 10×10 patches, one can say that more support vectors and higher values for both a and b produce slower results. In the case of only a few support vectors, quadratic kernels lead to the slowest outputs. Support vectors were sampled from a standardized Gaussian distribution.

are linearly combined which makes a slowness function unstable in the sense that removal of one single support vector contribution would significantly change the resulting function.

Neural net or sigmoid kernel

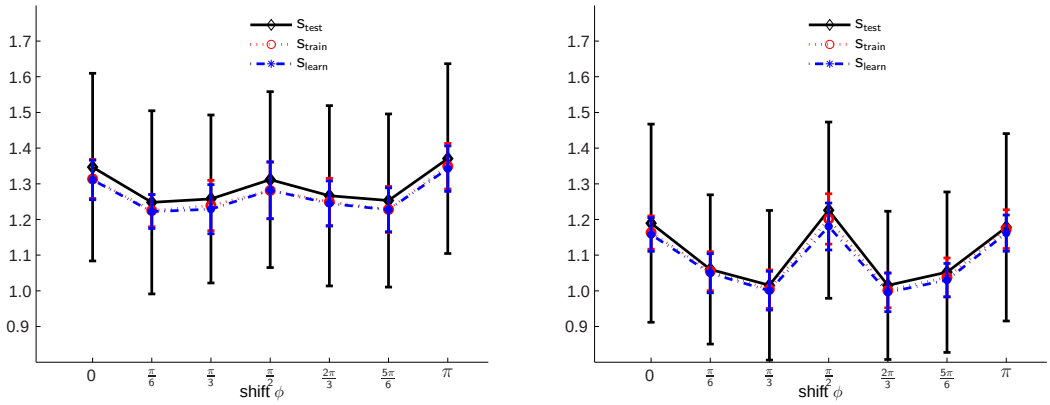
For reasons of curiosity and motivated by an equivalence of a special neural network with sigmoid activation function, the neural net kernel

$$k(\mathbf{z}, \mathbf{x}) = \tanh(\frac{\mathbf{z}^\top \mathbf{x}}{a} + b) = \tanh(\kappa \mathbf{z}^\top \mathbf{x} + \theta) \tag{4.16}$$

is also considered. Here, a is the scaling parameter, also known as gain κ and b is the offset, sometimes referred to as threshold θ. The superposition of two sigmoid functions in one dimension yields a localized function as shown in Appendix A.1.2. There are also results in the literature reporting similar performance of neural net kernels and RBF kernels in the context of support vector machines. To make it short, Figure 4.15 visualizes the slowness for two different scaling parameters.

4.4.3 RBF kernels

The second large class of kernel functions considered are so called radial basis function (RBF) kernels. In a nutshell, kernels only depend on the distance between $d(\mathbf{z}, \mathbf{x})$ the two data points. In the following, Euclidean distances (4.2) and distances respecting the covariance structure of the data



(a) $a = 400$, $N_{SV} = 200$ (b) $a = 1900$, $N_{SV} = 200$

Figure 4.14: The plot shows the resulting slowness values of sine kernels for two different scaling parameters a obtained in a 10-fold cross-validation experiment with 10×10 patches. Larger scaling produces slower responses, support vectors were randomly sampled from a Gaussian standard normal distribution.

based on general inner products (4.3) are used. The kernel has the general form $k(\mathbf{z}, x) = f(\|\mathbf{x} - \mathbf{z}\|)$ where f is usually a rapidly decaying continuous function. Some typical examples also used for kernelized SFA are shown in Figure 4.11(b).

The most prominent representatives among the RBF kernels are the Gaussian ($d = 2$) and the Laplacian ($d = 1$) kernels given by:

$$k(\mathbf{z}, \mathbf{x}) = \exp\left(-\left(\frac{\|\mathbf{x} - \mathbf{z}\|}{\sqrt{2}\sigma}\right)^d\right) \tag{4.17}$$

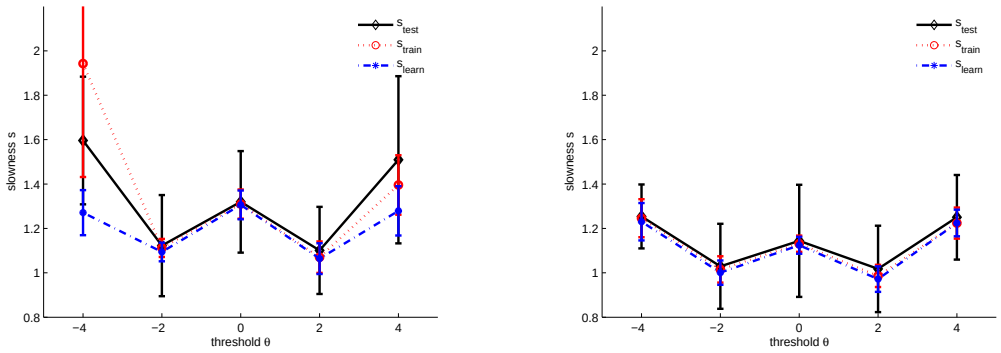
A visually similar kernel is a close relative to the Plummer kernel:

$$k(\mathbf{z}, \mathbf{x}) = \frac{\sigma^d}{\|\mathbf{x} - \mathbf{z}\|^d + \sigma^d} \tag{4.18}$$

The main advantage of the Plummer kernel is the fact that it can efficiently be calculated as the evaluation of the exponential is replaced by a simple division.

All three kernels of Figure 4.11(b) have the same range parameter σ - the major difference is the speed of decay and the differentiability at $\|\cdot\| = 0$.

The considered RBF kernels have only one parameter (if only some discrete values are assumed for d) signifying the length scale of the kernel. In Figure 4.16 the influence of the kernel width σ on the shape of the arising slowness function is investigated. Empirically successive widths for Gaussian and Laplacian kernels are examined in order to provide a feeling for the complexity of the slowness function in different directions of the data space. Subsets of only ten pixels allow for very smooth mappings only; along the hyperdiagonal more complex functions are possible f.i. double humps. The



(a) $1/\kappa = 400$, $N_{SV} = 200$ (b) $1/\kappa = 1600$, $N_{SV} = 200$

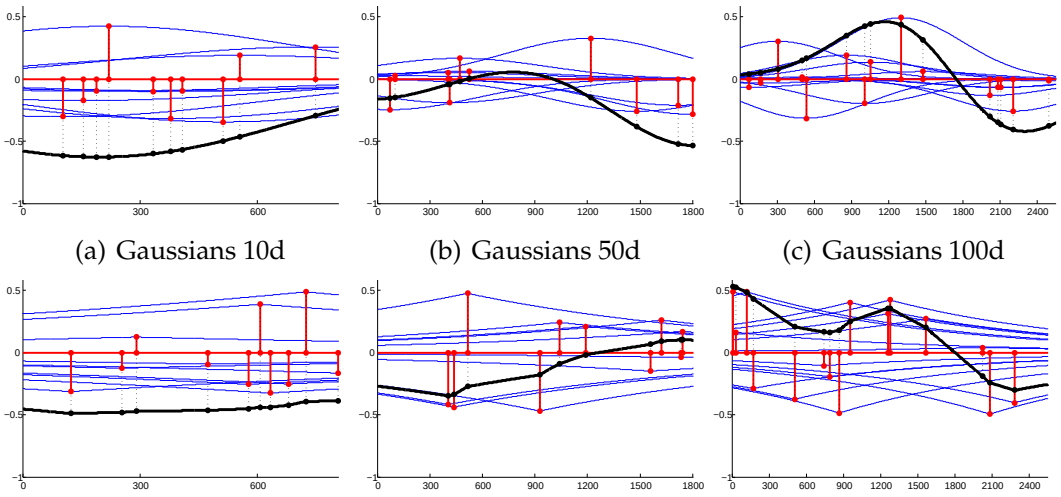
Figure 4.15: The plot shows the resulting slowness values of neural network kernels with hyperbolic tangent transfer function for two different scaling parameters (gain) κ obtained in a 10-fold cross-validation experiment with 10×10 patches. Larger gain produces slower responses, support vectors were randomly sampled from a Gaussian standard normal distribution.

sum of Gaussians is smooth, yet the sum of Laplacians visually looks like a locally linear interpolation.

Another important adjusting screw for model complexity is the number of support vectors. Patches of size 10×10 pixels live in a space of dimension $d = 100$. If one puts kernels of widths σ into that space, the number of kernels M – necessary for an entire covering – behaves like $M \propto (1/\sigma)^d$. For details, see Appendix A.2.2. This exponential scaling clearly shows that even relatively large numbers of kernels (say $10,000$) provide a very coarse covering of the data space ($\sqrt[100]{10,000} \approx 1.1$ kernels per dimension).

In order to derive an optimal set of parameters (σ, d) different simulations were carried out. The main results largely corresponding to intuition are summarized in Figures 4.12(b) and 4.17. One can clearly see that the degree d affects the resulting slowness; namely sub-Gaussian kernels yield the slowest responses and show similar behavior for the case of many support vectors. On the other hand, the kernel width has to be adjusted properly at least for the Gaussian case. Here one clearly gets an optimal width (in the range of $\sigma = 500$ for 10×10 patches). If kernels with a smaller width are used, very complex functions will be possible and overfitting problems will occur. The other way round, if kernels get too large, only very smooth functions are possible and underfitting will be the major problem. For heavy-tailed kernels such as the Laplacian kernel, the slowness performance is less sensitive to the exact value of the kernel width parameter. A broad range of values above a threshold of $\sigma = 800$ lead to good results. One also can see from Figure 4.12(b) that slownesses ($s_{test} \sim 0.6$) much smaller than in quadratically expanded SFA ($s_{test} \sim 0.8$) are possible. Note that, even if not plotted here, the Plummer kernel will produce equally slow output functions at smaller computational costs.

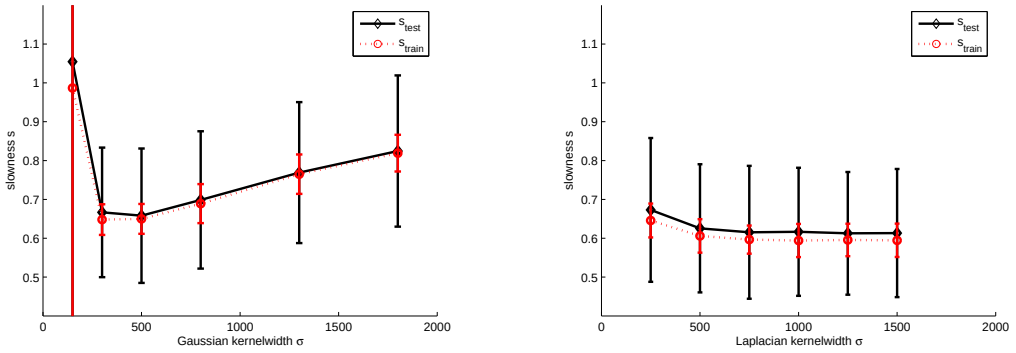
Finally, the question of finding the smallest slowness value for a single



(a) Gaussians 10d (b) Gaussians 50d (c) Gaussians 100d

(d) Laplacians 10d (e) Laplacians 50d (f) Laplacians 100d

Figure 4.16: To answer the question, what a typical RBF-kernel function in the space $[0, 255]^{10\times10}$ of image patches of size 10×10 looks like, Gaussian kernels with $\sigma = 500$ (a-c) and Laplacian kernels with $\sigma = 800$ (d-f) were linearly combined and plotted.
The space of patches $[0, 255]^{10\times10}$ is thus a 100-dimensional hypercube. The difference between the three plots is the number of directions (corresponding to single pixels) of the hypercube included. It can be seen, that along the longest possible line (the hyperdiagonal of length $\sqrt{100} \cdot 255$) the function can show more complex behavior than in only 10 dimensions ($\sqrt{10} \cdot 255$), where the kernel width is too large to allow arbitrarily complex behavior.



(a) Gaussian kernel $s(\sigma)$, $N_{SV} = 500$ (b) Laplacian kernel $s(\sigma)$, $N_{SV} = 200$

Figure 4.17: RBF kernels have a characteristic length scale which is given by the kernel width σ. The choice of the width determines the shape of possible functions. The figure shows 10-fold cross-validation experiments for Gaussian and Laplacian kernels using 10×10 patches. For Gaussian kernels there is a kind of optimal width, but in the case of Laplacian kernel the choice of length scale is not critical of above a certain value.

layer architecture was addressed. Taking larger patches clearly means a more complex model which translates as possibly slower output functions. In Figure 4.18 a Laplacian kernel is applied to patches of size 28×28. Optimization of the kernel width yields a value of about $\sigma = 2500$. For many support vectors, the slowness on the test set s_{test} does not decrease below a certain threshold even if more support vectors are selected. In contrast, the slowness of the training set s_{train} decreases further which corresponds to overfitting as the gap between s_{test} and s_{train} becomes larger. Optimal values are smaller than 0.2.

4.5 Understanding the filters

Using the results from 4.4 yielding the optimal kernel k and the corresponding support vectors $\mathbf{Z}$ one now possesses a nonlinear filter function $\mathbf{g}(\mathbf{x}_t) = \mathbf{W}^\top \mathbf{k}(\mathbf{Z}, \mathbf{x}_t)$ being optimal in terms of achieved temporal slowness. Up to now, this filter g is a black box. In order to shed some light on the properties of the nonlinear mapping $\mathbf{g} : \mathbb{R}^N \to \mathbb{R}^K$, the following analysis is carried out: Filtered images are studied and visually characterized, patches producing maximal and minimal responses are determined, interesting projections in the space of filtered images are found via ICA and trajectories of artificially generated patches are explored.

4.5.1 Filter output

Two selected gray scale images (240×320 pixels) from the robot lab were filtered by patchwise (10×10 pixels) application of the filters found by kernelized SFA. The ten slowest filtered images (231×311 pixels) are shown in Figure 4.19. They do not share the same color coded scale of intensity but the color for zero is equal. First of all, there are two classes of filters. Three filters produce responses all over the image and seven filters respond only locally to horizontal structures. For further interpretation, one may also remark that five filters exhibit unipolar responses even though the responses have zero mean and unit variance and are mutually decorrelated. Further analysis of the first three filters $k = 1..3$ reveals that they act like a simple low-pass filter $\mathbf{f}$ on the image $\mathbf{I}$ (written as $\mathbf{I}^{\text{filt}} = \mathbf{I} \star \mathbf{f}$) combined with a gray value transfer function ϕ_k operating in a pixelwise way: $\mathbf{I}^{\text{slow}}_{ij} = \phi_k(\mathbf{I}^{\text{filt}}_{ij})$

The last seven filters extract horizontal edges in different phases which guarantees decorrelatedness of the responses. The fact that horizontal edges are the most prominent features detected can be further confirmed and refined by plotting the length of the feature vectors in the slowness space as done in Figure 4.20. The overlay of the color coded length of the first ten responses and the initial gray value image confirms the striking dominance of horizontal structures like the sharp edge of the table or the almost black-white transition near the seat of the chair.

Having a closer look at the video data obtained by navigating the robot

in the lab and in the floor in front of the lab reveals an idea why horizontal structure is so dominant. The robot has two degrees of freedom – it can move forward and it can rotate. In the image plane, a rotation of the robot induces mainly horizontal movement (see Figure 3.2(b)) and also a step forward causes movement in horizontal direction (see Figure 3.2(a)). Finally, one can intuitively summarize the strategy implemented by the filters as follows: To respect the unit variance constraint, the filter has to have a nonzero output somewhere. On the other hand, it is clear that horizontal structure changes very little under horizontal translation which means that those structures are a priori very likely to change only a little bit in the future. The filter is able to correct for the "little bit" by invariant response to slight changes.

4.5.2 Scatter plot analysis and ICA

The pairwise scatter plots of Figure 4.21 show in the upper right triangular part the positions of image patches (taken from the images of Figure 4.20) in slowness space. Each of the six slow components is centered while the six components are mutually decorrelated and scaled to have unit variance. But instead of showing a spherical structure, the plots show prominent directions reaching away from the origin which means large overall response. These prominent directions are considered to be interesting.

Let $\mathbf{X} \in \mathbb{R}^{N\times T}$ denote the data matrix with containing T data points $\mathbf{x}_t \in \mathbb{R}^N$ with zero mean $\langle \mathbf{x}_t \rangle = \mathbf{0}$ and covariance $\mathbf{C} = \left\langle \mathbf{x}\mathbf{x}^\top \right\rangle$ and let f be a contrast function. The task of finding M projections $\mathbf{W} \in \mathbb{R}^{N\times M}$ maximizing f is efficiently solved by the FastICA[1] fixed point algorithm.

The case of $f(t) = t^2$ corresponds to skewness or asymmetry maximization and $f(t) = t^3$ matches curtosis maximization. Postprocessing the data with ICA does not change the slowness of the output (2.32) because the transform is non-singular and it acts only on the subspace spanned by the slow components.

***Algorithm 3** FastICA fixed point algorithm*

`init` $\mathbf{W} \leftarrow \mathcal{N}(0, \epsilon^2)$
`iterate`
- $\mathbf{W}_{\text{new}} \leftarrow \mathbf{C}^{-1}\mathbf{X} f(\mathbf{X}^\top \mathbf{W})$ `(optimization)`
- $\mathbf{W}_{\text{new}} \leftarrow \mathbf{W}_{\text{new}} \left(\mathbf{W}_{\text{new}}^\top \mathbf{C} \mathbf{W}_{\text{new}}\right)^{-\frac{1}{2}}$ `(orthogonalization)`

`until` $\|\mathbf{W}_{\text{new}} - \mathbf{W}\|_2 < \gamma$

The new axes obtained by curtosis maximization with FastICA show quite a good alignment as depicted in the lower left triangular part of Figure 4.21. A qualitative difference can also be observed in the filtered im-

[1] The FastICA package for Matlab® implementing the fast fixed-point algorithm for ICA can be downloaded from `http://www.cis.hut.fi/projects/ica/fastica/`.

ages shown in Figure 4.22. The border of the table leads to a double response which is clearly separated by ICA. It seems easier to assign a special meaning to the learned filters. For example, the first component responds to white regions, the second one extracts double edges (bright-dark-bright or dark-bright-dark) and the third component implements a segmentation based on the mean color into regions of extreme color and regions of intermediate color.

4.5.3 Scatter plot analysis and clustering

Coming back to the scatter plots, in order to extract the branches of interest one can apply a thresholding operation. Due to the unit variance constraint the axes have the same scale, thus a simple spherical threshold on the distance to the origin is a reasonable criterion.

In Figure 4.23(a), the initial scatter plot of patches projected into slowness space and a threshold including 99 per cent of the data is shown. Figure 4.23(b) shows the same data after thresholding and projected onto the unit sphere. Visual inspection shows a clear cluster structure at least in the two dimensional projections and motivates the use of standard clustering algorithms like k-means under the constraints of unit length.

The unit sphere $\|\mathbf{x}\| = 1$ embedded into $\mathbb{R}^N$ has a local dimension of $N-1$ and inner product, angle and induced metric can be related via:

$$\|\mathbf{x}-\mathbf{y}\|^2 = 2 - 2\langle \mathbf{x}, \mathbf{y}\rangle = 2 - 2\cos(\alpha)$$

That means, the usual K-means algorithm can be rewritten using dot products and rescaling operations leading to Algorithm 4.

Algorithm 4 *K-means Clustering on a Hypersphere*

`init` $\beta \leftarrow \beta_0$, $\eta > 1$, $\mathbf{w}_q^{\text{old}} \leftarrow \mathcal{N}(0, \epsilon^2)$
`while` $\beta < \beta_{\text{end}}$ `do`

- $P(t \to q) \leftarrow \exp \beta \langle \mathbf{x}_t, \mathbf{w}_q^{\text{old}} \rangle / \sum_p \exp \beta \langle \mathbf{x}_t, \mathbf{w}_p^{\text{old}} \rangle$ `(classes)`
- $\mathbf{w}_q^{\text{new}} \leftarrow \sum_t \mathbf{x}_t P(t \to q) / \sum_t P(t \to q)$ `(new centers)`
- $\mathbf{w}_q^{\text{new}} \leftarrow \mathbf{w}_q^{\text{new}} / \|\mathbf{w}_q^{\text{new}}\|$ `(normalization)`
- $\beta \leftarrow \eta\beta$ `(annealing)`

`until` $\|\mathbf{w}_q^{\text{new}} - \mathbf{w}_q^{\text{old}}\| < \gamma$

For our data set, k-means detects two major clusters of two subclusters each. The upper cluster consists of sharp horizontal edges with a dark lower part and the lower one contains horizontal edges where the light comes from below. All in all, the analysis reveals not only which patches give rise to large responses but also how these patches are structured.

4.5.4 Trajectories in slowness space

The last experiment, depicted in Figure 4.24, goes further into the question how transformations of patches in the input space translate into slowness

space. Invariance with respect to a certain transformation means that all transformed patches are mapped to the same point in slowness space. Four sets of artificially generated patches are considered

4.6 Multi-layer results

Computation times, already quite large in the single-layer case, become really huge for multi-layer architectures because the filters of previous layers have to be applied to generate the inputs to the last layer. In this thesis, three multi-layer architectures were compared to each other and to a single-layer approach – all have an overall receptive field of 28×28 pixels:

- 28×28 receptive field in the first layer
- 10×10 receptive field in the first layer and
  19×19 receptive field in the second layer
- 19×19 receptive field in the first layer and
  10×10 receptive field in the second layer
- 10×10 receptive field in the first, second and third layer

Simulation results for multi-layer SFA with Gaussian and Laplacian RBF kernels are summarized in Figure 4.25. Comparing the absolute values to single-layer slownesses of Figure 4.18, one can draw the following conclusions. Gaussian kernels are more suitable in higher layers. Slownesses obtained are not significantly different and the differences between test and training values have the same size. That means slowness performance and generalization abilities are very similar but computation times in terms of learning and filtering are significantly larger.

By means of Figure 4.26 and Figure 4.19(b) one can visually compare the filter outputs for different architectures. The first four slow components are visually equivalent. One can hypothesize from Figure 4.26 that the size of the overall receptive field is more important than the sizes of the receptive fields of the several layers.

In the first layer, basically horizontal structure is extracted. Higher layers will consequently only be able to combine features of the previous layers nonlinearly. Thus, it is a good idea to include more than only ten slow components. However, vertical edges are prominent in slow components of indices around 100. Hence, very many slow components have to be kept in order to conserve information about vertical structure. This clearly limits the possible receptive field sizes in higher layers. Future experiments should increase the size of the receptive field very slowly and rather preserve many slow components.

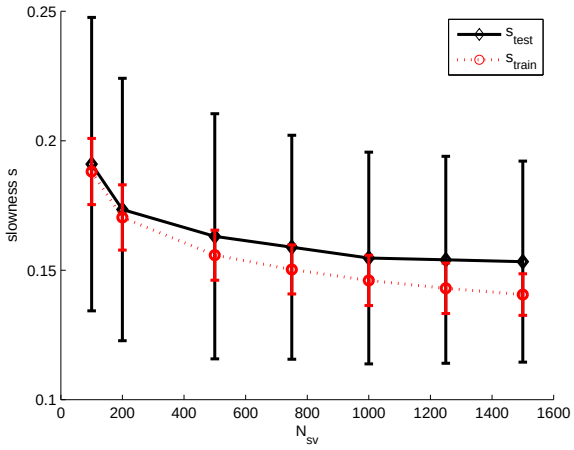
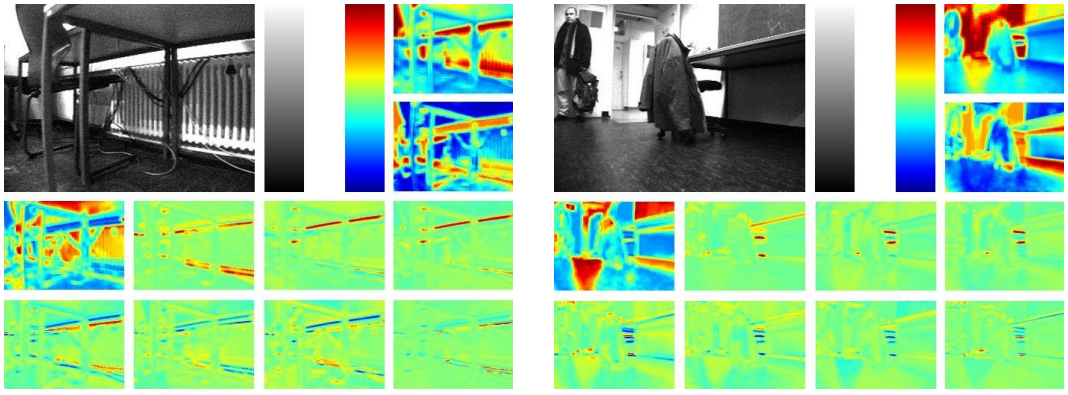


Figure 4.18: In order to find out the slowest possible architecture for the first layer, a Laplacian kernel of width $\sigma = 2500$ was used. The patch size was increased to 28×28 and more and more support vectors were used. The lowest possible test slowness value lies around 0.18. One can see that the test slowness s_{test} saturates whereas the training slowness s_{train} gets smaller and smaller. This can be interpreted as overfitting.



(a) (b)

Figure 4.19: Two images from the robot lab were filtered using a Laplacian kernel of width $\sigma = 800$ and $N_{SV} = 200$ support vectors. The first three slowest components smoothen the image and assign other color values, the other seven components extract horizontal edges in different phases. Among the slow components there are five components that show unipolar responses only. All ten images have the same color for zero activity but a different scale for absolute values.

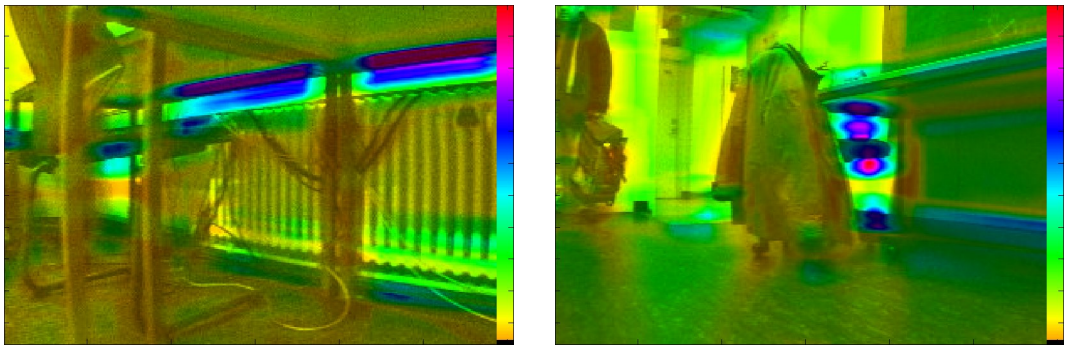


Figure 4.20: The images show the norm $\|\mathbf{y}\| = \|\mathbf{g}(\mathbf{x})\|$ of the first $K = 10$ slow components y_i from Figure 4.19. Colors encode the value of the norm but the gray values of the original image retained. The most striking features are horizontal structures like the edge of the table and the seat of the chair.

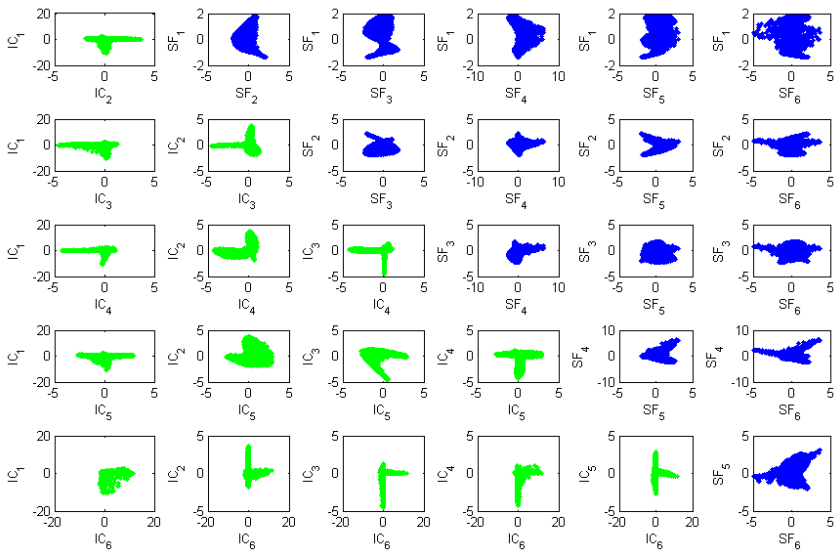


Figure 4.21: The scatter plots of the first six slow components (upper right triangle) of Figure 4.19 do not have spherical structure, even though they are decorrelated. In order to extract the prominent directions, one can apply ICA (lower left triangle). Note that this non-singular transformation does not change the slowness value because it acts in the subspace of the extracted slow features only.

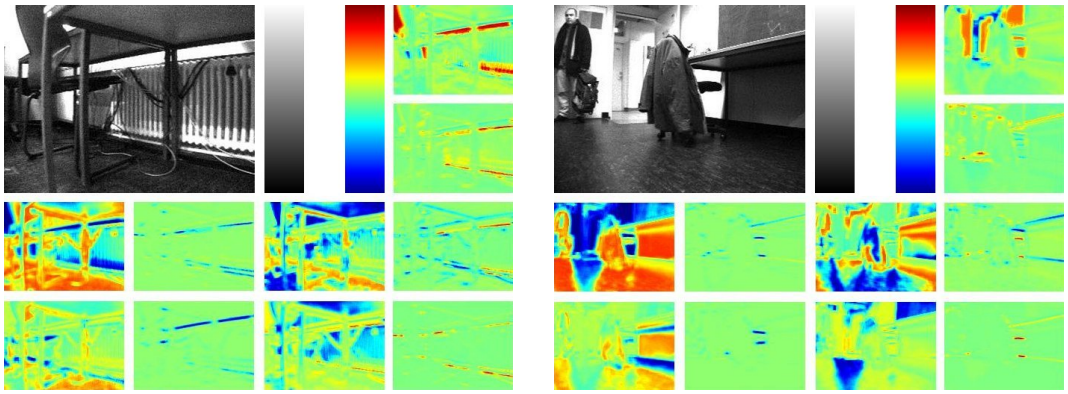
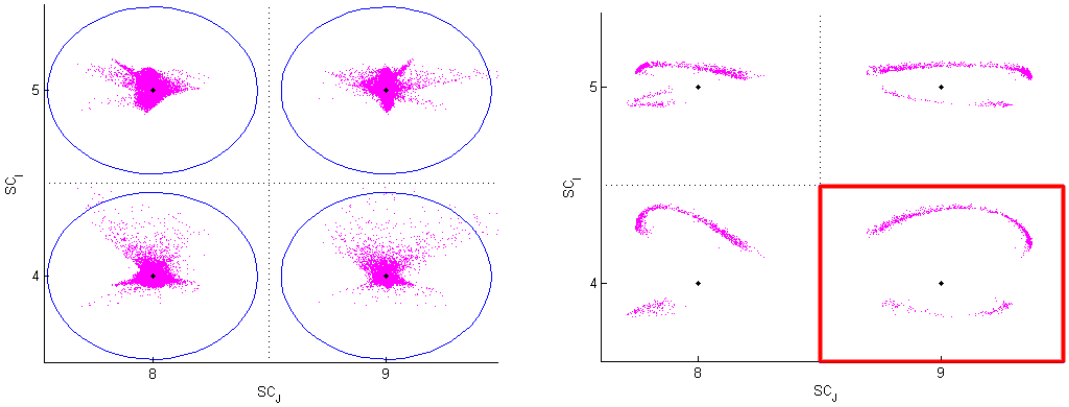
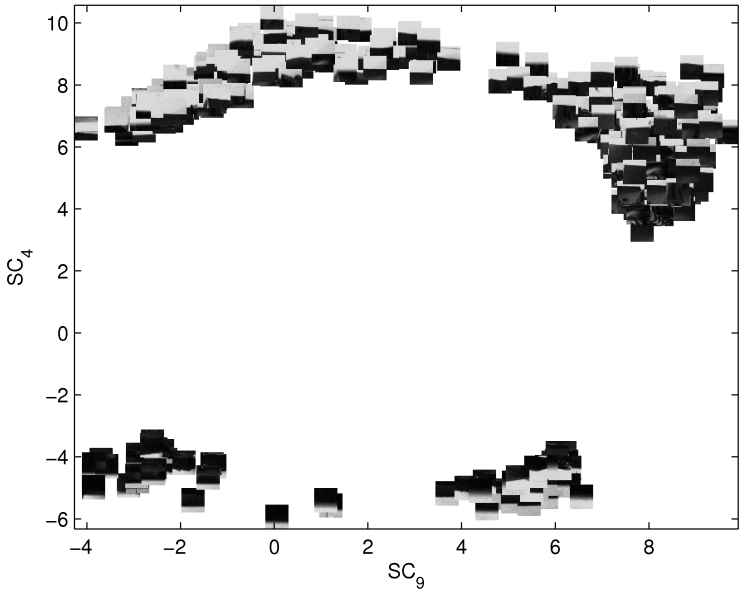


Figure 4.22: The two images from the robot lab were filtered by using a Laplacian kernel of width $\sigma = 800$ and $N_{SV} = 200$ support vectors. On top of that, ICA was applied. As a result one obtains three unipolar and two bipolar filters extracting horizontal edges. One filter simply responds to white regions in the input and the remaining four filters act on the image histogram. All ten images have the same color for zero activity but a different scale for absolute values.



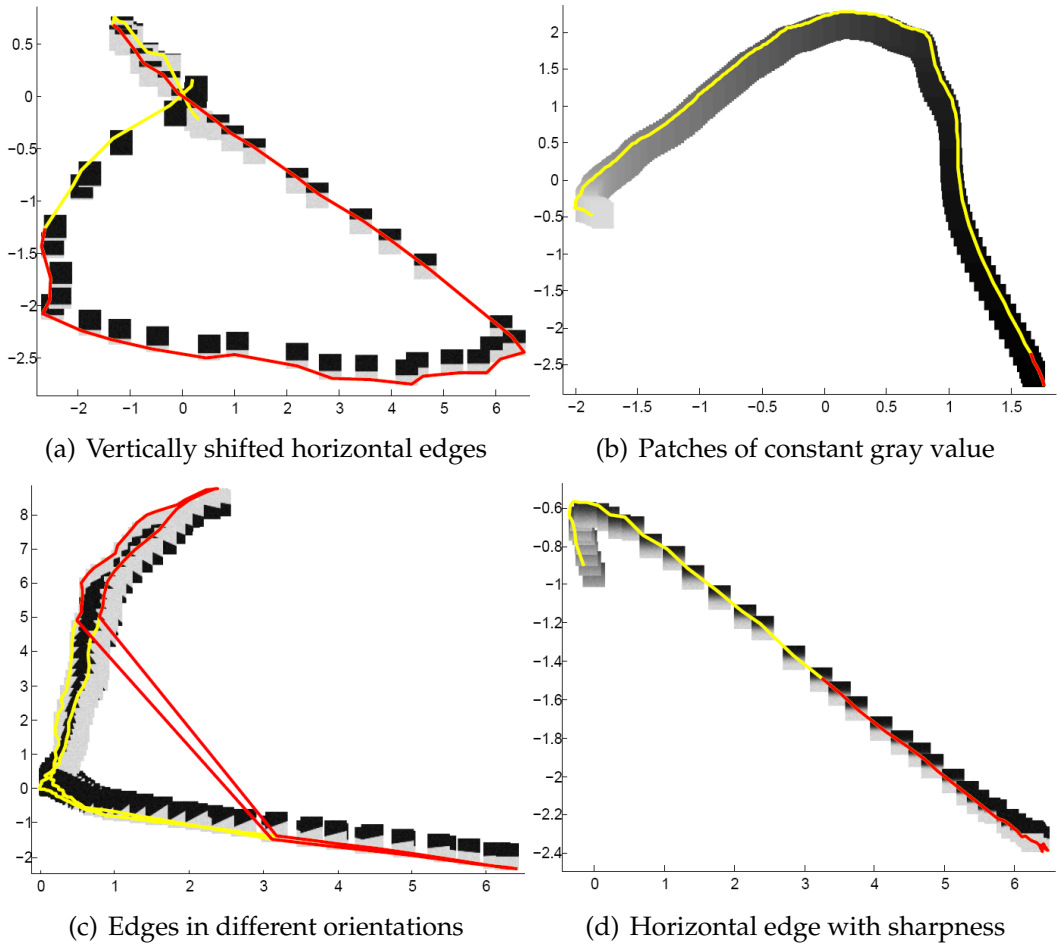
(a) Scatter plot before thresholding

(b) Scatter plot after thresholding



(c) Corresponding remaining patches

Figure 4.23: To capture the "branches" (i.e. the patches with the most striking response) of the data cloud (top left) reaching far away from the origin, a thresholding operation is applied. All data points outside some radius τ are kept, the rest is discarded. The threshold τ is chosen so that only 1 per cent of the data is kept. For convenience, the remaining data points are projected onto the unit sphere (top right).
Clustering (according to Algorithm 4) of the remaining data points reveals that there are two respectively four centers of mass. The corresponding patches are shown in the plot at the bottom. The two main clusters are horizontal edges with a lower bright part and a lower dark part. These clusters can be split in sub-clusters characterized by the dominance of either the lower or the upper part in terms of width.



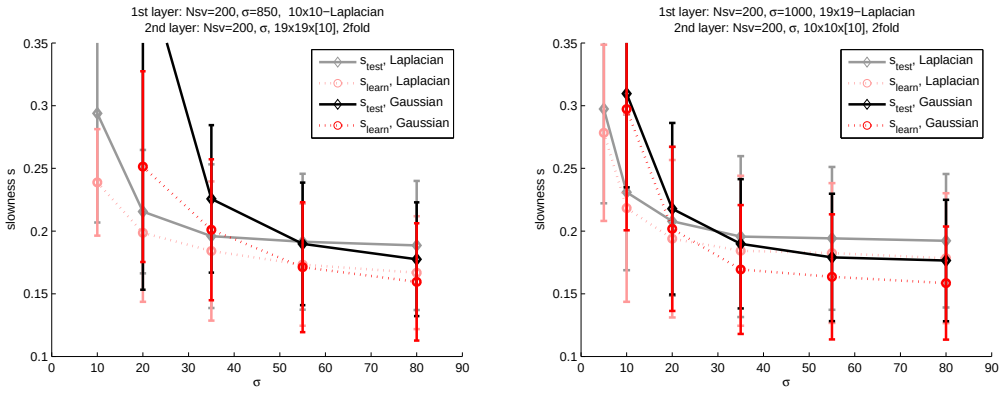
(a) Vertically shifted horizontal edges

(b) Patches of constant gray value

(c) Edges in different orientations

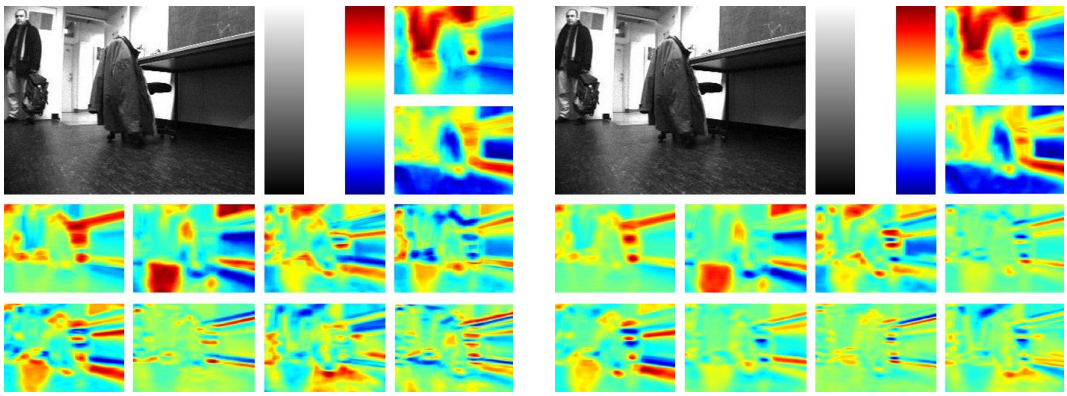
(d) Horizontal edge with sharpness

Figure 4.24: Artificially generated edge images do have characteristic trajectories in the space of the filter outputs. Here, edges of variable width, edges in different orientations, edges with varying sharpness and constant image patches are considered.
The yellow line connects successive image patches of the artificial sequence, and the red line illustrates patches whose filter response is larger than that of 95 percent of the corpus.



(a) $10 \times 10 \rightarrow 19 \times 19 \times (10)$ (b) $19 \times 19 \rightarrow 10 \times 10 \times (10)$

Figure 4.25: A second layer is learned based on the 10 slowest output signals of the first layer (Laplacian kernel, 200 support vectors, optimal σ). Figure (a) shows results for a smaller receptive field in the first layer and Figure (b) shows results for a larger receptive field in the first layer. Gaussian and Laplacian kernels with 200 support vectors each were used in the second layer; Gaussian kernels produce slower outputs.



(a) $10 \times 10 \rightarrow 19 \times 19 \times (10)$ (b) $19 \times 19 \rightarrow 10 \times 10 \times (10)$

Figure 4.26: The ten slowest output signals of the first layer (Laplacian kernel, 200 support vectors, optimal σ) are used to learn a second layer with 200 support vectors and Gaussian kernels of width $\sigma = 80$. Interestingly, the filtered images look very similar to each other. (The sign of some components was flipped for visualization.)

Chapter 5

Conclusion

5.1 Summary

SFA is able to extract interpretable features from natural videos. Temporal smoothness and decorrelation is achieved either by mappings into another color-space after a low-pass filtering step or by the extraction of horizontal edges in different phases. Nonlinear networks obtained by kernelization achieve much slower outputs than explicit expansion into a feature space, e.g. polynomial while having less computational complexity. Unfortunately, kernelized networks are far more difficult to interpret or to analytically predict than general quadratic functions.

Extensive parameter explorations show how multi-layer networks should be constructed to have slowly varying outputs. In kernelized networks it turned out that RBF kernels, especially Laplacian kernels, yield best performance when the range of influence σ is adjusted as suggested. In general, a larger receptive field, i.e. larger image patches, increases the complexity and leads to slower results. The same holds true for more support vectors, even though additional support vectors have a smaller effect when many support vectors are already present.

Efficient filtering methods in the frequency domain suitable for large data sets are composed and implemented by rewriting the filters as linear operations. A major source for computational complexity is, besides the receptive field size of the filter, the number of support vectors being used. Therefore, methods of support vector selection prior to learning and by sparse optimization are studied and enhanced, in a way that the user is able to achieve optimally slow outputs for a given number of support vectors.

Finally, slow functions in the true video do not necessarily have to be slow in the simulator videos. Slow functions of simulator videos are usually slow in true videos but less slow than functions directly learned from natural videos. This clearly suggests that the simulator suits only in a very limited way for off-line learning.

5.2 Discussion

It is clear that SFA can only become invariant to transformations present in the input data. Hence, extracted horizontal structure signifies invariance to horizontal translations in the image plane. Unfortunately, vertical movements are less prominent in the robot's videos and the decorrelation constraint in SFA does not lead to simultaneous emergence of feature detectors invariant to both horizontal and vertical translations in the first few slow components.

Networks composed of RBF kernels of width σ are universal function approximators of resolution σ as long as the whole space is adequately covered. One can object that a hundred dimensional space is only very coarsely covered by only thousand kernels, even when they originate from the true data distribution. Consequently, not enough support vectors might be computationally tractable to work in the really interesting regimes. More support vectors will lead to slower output functions even if they are randomly selected. There are two possible explanations: On the one hand, one can hypothesize more "good" vectors to lie in a larger set of candidates. On the other hand, the sheer number of support vectors can indeed lead to slower results, no matter what their exact value is. In general, RBF functions are local approximators, but Laplacian kernels have heavy tails allowing for a small influence in larger distances which is obviously an advantage in the first layer. The locality property implies features coded in advance by the choice of the support vectors.

Another criticism might aim at the biological plausibility of the learned functions. At least, they are hierarchically organized in layers. No recurrency is present and learning only happens layer by layer. Processing speed is also far away from real time capability. But this is an effect of the von Neumann bottleneck. Parallel hardware, more adapted to the inherently parallel structure of the filtering, as the pixel pipelines of modern graphics cards, can account for that.

During learning, patches are randomly clipped from the video, regardless of their importance and structure. As a result, most of the patches might be uninteresting because objects or important structure are likely to occur very seldom in the video.

5.3 Outlook

The major remaining task is to construct an application taking advantage of the preprocessing step learned by the principle of temporal coherence. Another direction of further research is the analysis of the invariances implemented by the slow features. This could be done by starting at the image patch leading to the strongest response and following the direction of smallest change in the filter function.

Some multi-layer architectures were simulated, but only shallow settings with at most three layers were studied. It would be interesting to investigate

which patterns remain stable in higher layers and how deeper networks influence the features extracted and the generalization behavior. A possible way to overcome computational limitations consists of slowly growing receptive fields while keeping many slow components for higher layers.

One can also try to preprocess the video by standard edge extraction filters or complex cell filters to explicitly code the first layer and focus on high-level combinations of these features. Additionally one can extend the input data to additional channels like colors or stereo information.

Appendix A

Additional material

A.1 Gradients

A.1.1 Joint sparseness

In the following, the gradient of the regularized objective function $s(\mathbf{W}) + \lambda\rho(\mathbf{W})$ is derived by using the matrix valued functions $\sqrt{\cdot}$ (element-wise square root $\sqrt{[a_{ij}]} := [\sqrt{a_{ij}}]$) and $\mathrm{dg}^{-\frac{1}{2}}(\cdot)$ (inverse square root of diagonal elements, off-diagonal elements are set to zero $\mathrm{dg}^{-\frac{1}{2}}([a_{ij}]) := [\delta_{ij}\sqrt{a_{ij}^{-1}}]$).

$$\begin{aligned} s(\mathbf{W}) &= \mathrm{tr}(\mathbf{W}^\top \mathbf{A}\mathbf{W}) \\ \rho(\mathbf{W}) &= \frac{\left(\sum_i \sqrt{\sum_{j=1}^n w_{ij}^2}\right)^2}{\sum_i \sum_{j=1}^n w_{ij}^2} = \frac{\mathrm{tr}^2\sqrt{\mathbf{W}\mathbf{W}^\top}}{\mathrm{tr}(\mathbf{W}\mathbf{W}^\top)} =: \frac{f(\mathbf{W})}{g(\mathbf{W})} \end{aligned}$$

An auxiliary calculation that will be helpful:

$$\begin{aligned} \frac{\partial}{\partial w_{kl}}\mathrm{tr}\sqrt{\mathbf{W}\mathbf{W}^\top} &= \frac{\partial}{\partial w_{kl}} \sum_i \sqrt{\sum_j w_{ij}^2} = \frac{1}{2}\sum_i \left(\sum_j w_{ij}^2\right)^{-\frac{1}{2}} \frac{\partial}{\partial w_{kl}} \sum_j w_{ij}^2 \\ &= \sum_i \left(\sum_j w_{ij}^2\right)^{-\frac{1}{2}} \sum_j \delta_{ik} w_{il} = \left(\sum_j w_{kj}^2\right)^{-\frac{1}{2}} w_{kl} \quad \text{(A.1)} \\ \frac{\partial}{\partial \mathbf{W}}\mathrm{tr}\sqrt{\mathbf{W}\mathbf{W}^\top} &\overset{(A.1)}{=} \mathrm{dg}^{-\frac{1}{2}}\left(\mathbf{W}\mathbf{W}^\top\right)\mathbf{W} \quad \text{(A.2)} \end{aligned}$$

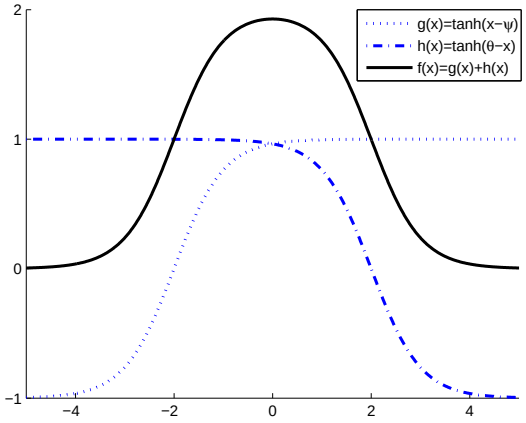
The gradients of the numerator and the denominator are calculated separately and then combined by using the quotient rule.

$$\begin{aligned}
\frac{\partial f}{\partial \mathbf{W}} &\overset{(A.2)}{=} 2\mathrm{tr}\sqrt{\mathbf{W}\mathbf{W}^\top}\mathrm{dg}^{-\frac{1}{2}}\left(\mathbf{W}\mathbf{W}^\top\right)\mathbf{W} && \text{(A.3)}\\
\frac{\partial g}{\partial \mathbf{W}} &= 2\mathbf{W}\\
\frac{\partial \rho}{\partial \mathbf{W}} &= \frac{\partial}{\partial \mathbf{W}}\frac{f}{g} = \frac{1}{g^2}\left(g\frac{\partial f}{\partial \mathbf{W}} - f\frac{\partial g}{\partial \mathbf{W}}\right) = \frac{1}{g}\frac{\partial f}{\partial \mathbf{W}} - \frac{f}{g^2}\frac{\partial g}{\partial \mathbf{W}} && \text{(A.4)}\\
&\overset{(A.3+A.4)}{=} 2\frac{\mathrm{tr}\sqrt{\mathbf{W}\mathbf{W}^\top}}{\mathrm{tr}(\mathbf{W}\mathbf{W}^\top)}\mathrm{dg}^{-\frac{1}{2}}\left(\mathbf{W}\mathbf{W}^T\right)\mathbf{W} - 2\left(\frac{\mathrm{tr}\sqrt{\mathbf{W}\mathbf{W}^\top}}{\mathrm{tr}(\mathbf{W}\mathbf{W}^\top)}\right)^2\mathbf{W}\\
\frac{\partial s}{\partial \mathbf{W}} &= (\mathbf{A}+\mathbf{A}^\top)\mathbf{W} \overset{\text{sym.}}{=} 2\mathbf{A}\mathbf{W}
\end{aligned}$$

Finally, the gradient takes the following form:

$$\frac{\partial s + \lambda\tau}{\partial \mathbf{W}} = 2\left(\mathbf{A} + \lambda\frac{\mathrm{tr}\sqrt{\mathbf{W}\mathbf{W}^\top}}{\mathrm{tr}(\mathbf{W}\mathbf{W}^\top)}\mathrm{dg}^{-\frac{1}{2}}\left(\mathbf{W}\mathbf{W}^\top\right) - \lambda\left(\frac{\mathrm{tr}\sqrt{\mathbf{W}\mathbf{W}^\top}}{\mathrm{tr}(\mathbf{W}\mathbf{W}^\top)}\right)^2\mathbf{I}\right)\mathbf{W} \tag{A.5}$$

A.1.2 Superposition of two sigmoid functions



$$\begin{aligned}
f(x) &= \tanh(x-\psi) - \tanh(x-\theta) = \frac{e^{x-\psi} - e^{\psi-x}}{e^{x-\psi} + e^{\psi-x}} - \frac{e^{x-\theta} - e^{\theta-x}}{e^{x-\theta} + e^{\theta-x}}\\
&= \frac{\left(e^{x-\psi} - e^{\psi-x}\right)\left(e^{x-\theta} + e^{\theta-x}\right) - \left(e^{x-\psi} + e^{\psi-x}\right)\left(e^{x-\theta} - e^{\theta-x}\right)}{\left(e^{x-\psi} + e^{\psi-x}\right)\left(e^{x-\theta} + e^{\theta-x}\right)}\\
&= \frac{\left[e^{2x-\psi-\theta} + e^{\theta-\psi} - e^{\psi-\theta} - e^{\psi+\theta-2x}\right] - \left[e^{2x-\psi-\theta} - e^{\theta-\psi} + e^{\psi-\theta} - e^{\psi+\theta-2x}\right]}{e^{2x-\psi-\theta} + e^{\psi-\theta} + e^{\theta-\psi} + e^{\psi+\theta-2x}}\\
&= \frac{2e^{\theta-\psi} - 2e^{\psi-\theta}}{e^{2x-\psi-\theta} + e^{\psi-\theta} + e^{\theta-\psi} + e^{\psi+\theta-2x}}
\end{aligned}$$

By setting $\theta = -\psi = \frac{\lambda}{2} > 0$ one obtains

$$f(x) = \frac{2e^{\lambda} - 2e^{-\lambda}}{e^{2x} + e^{\lambda} + e^{-\lambda} + e^{-2x}} = f(-x)$$

and one can deduce the limit

$$\lim_{x \to \infty} f(x) = 0,$$

the derivative

$$f'(x) = 2\frac{2e^{\lambda} - 2e^{-\lambda}}{\left(e^{2x} + e^{\lambda} + e^{-\lambda} + e^{-2x}\right)^2} \left(e^{-2x} - e^{2x}\right) \stackrel{!}{=} 0$$

and the maximum value $x_{\max} = 0$.

A.2 Kernel considerations

A.2.1 Equivalence of eSFA and kSFA for polynomials

The set of polynomials up to degree D of $\mathbf{x} \in \mathbb{R}^N$is given as

$$\mathbb{P}^D_{\mathbf{x}} = \mathcal{L}\left\{\prod_{\sum_{i=1}^N d_i \leq D} x_i^{d_i}\right\}$$

where polynomials $p(\mathbf{x}) \in \mathbb{P}^D$ are functions $p : \mathbb{R}^N \to \mathbb{R}$ and $\mathcal{L}\{\cdot\}$ is the convex hull. As a result, one knows that $\dim \mathbb{P}^D_{\mathbf{x}} = \sum_{d=1}^D \binom{N+d-1}{d}$. The space of polynomials is equipped with an addition and a multiplication with a scalar.

On the other hand, the set of all polynomial kernel functions can be written as

$$\begin{aligned}\mathbb{F}^D_{\mathbf{x}} &= \mathcal{L}\left\{k(\mathbf{z}, \mathbf{x})\right\} = \mathcal{L}\left\{\left(\frac{\mathbf{z}^\top \mathbf{x}}{a} + b\right)^D\right\} = \mathcal{L}\left\{b^D \left(\frac{\mathbf{z}}{ab}^\top \mathbf{x} + 1\right)^D\right\} \\ &= \mathcal{L}\left\{(\tilde{\mathbf{z}}^\top \mathbf{x} + 1)^D\right\}\end{aligned}$$

It is sufficient to consider the parameter D only, because b is consumed by the convex hull and scaling by a can be achieved by scaling the support vectors. One can rewrite the simplified polynomial kernel using the multinomial theorem as

$$\begin{aligned}\left(\mathbf{z}^\top \mathbf{x} + 1\right)^D &= \sum_{d=0}^D \binom{D}{d} \left(\sum_{i=1}^N z_i x_i\right)^d \\ &= \sum_{d=0}^D \binom{D}{d} \sum_{\sum_{i=1}^N d_i = d} \frac{d!}{\prod_{i=1}^N d_i!} \prod_{i=1}^N z_i^{d_i} x_i^{d_i} \qquad \text{(A.6)} \\ &= \sum_{\sum_{i=1}^N d_i = d=1}^D \alpha_{d_1 \ldots d_N} \prod_{i=1}^N x_i^{d_i} \in \mathbb{P}^D_{\mathbf{x}}\end{aligned}$$

which shows that $\mathbb{F}_{\mathbf{x}}^D \subseteq \mathbb{P}_{\mathbf{x}}^D$. The equivalence of the two sets for $D = 2$ is shown later in the text by construction of a set of support vectors of size $M = \frac{1}{2}N^2 + \frac{3}{2}N + 1$ equal to the dimension of $\mathbb{P}_{\mathbf{x}}^2$. One can suspect that similar constructions are possible for $D > 2$.

Additionally, one can see the feature map of $\left(\mathbf{z}^\top \mathbf{x} + 1\right)^D$ by inspection of A.6 to equal

$$\mathbf{\Phi}_N^D(\mathbf{x}) = \left(\sqrt{\binom{D}{d} \frac{d!}{\prod_{i=1}^N d_i!}} \prod_{i=1}^N x_i^{d_i} \right)_{\sum_{i=1}^N d_i = d \leq D}$$

in general for polynomial kernels of degree D in N dimensions and especially:

$$\begin{aligned}
\mathbf{\Phi}_2^2(\mathbf{x}) &= \left(1,\ \sqrt{2}x_1,\ \sqrt{2}x_2,\ x_1^2,\ \sqrt{2}x_1x_2,\ x_2^2\right) \\
\mathbf{\Phi}_3^2(\mathbf{x}) &= \left(1,\ \sqrt{2}\left\{x_1, x_2, x_3\right\},\ \sqrt{2}\left\{x_1x_2, x_1x_3, x_2x_3\right\},\ \left\{x_1^2, x_2^2, x_3^2\right\}\right) \\
\mathbf{\Phi}_2^3(\mathbf{x}) &= \left(1,\ \sqrt{3}\left\{x_1, x_2\right\},\ \sqrt{6}x_1x_2,\ \sqrt{3}\left\{x_1^2, x_2^2\right\},\ \sqrt{3}\left\{x_1^2x_2, x_1^1x_2^2\right\},\ \left\{x_1^3, x_2^3\right\}\right)
\end{aligned}$$

How to choose a small subset of support vectors, such that the space of polynomials is covered in an optimal way, remains an open question. Experimental evidence suggests that the choice might not be critical. A good idea, however would be to choose support vectors $\mathbf{z}_i \in \mathbb{R}^N$ that approximately span at least the space $\mathbb{R}^N$.

Quadratically expanded SFA

In SFA with quadratic expansion and additional linear and constant part, every possible output signal $y_{j,t}$ takes the following form:

$$y_{j,t} = g_j(\mathbf{x}_t) = \mathbf{x}_t^\top \mathbf{H}_j \mathbf{x}_t + \mathbf{f}_j^\top \mathbf{x}_t + c_j \tag{A.7}$$

where $\mathbf{H}_j$ is a symmetric matrix summarizing the coefficients for the mixed (quadratic) terms $x_i x_l$, $\mathbf{f}_j$ is a vector containing the coefficients for the linear terms x_i and c_j is just a constant offset. Alternatively, the expression can be written as a single quadratic form:

$$g(\mathbf{x}_t) = \left[\mathbf{x}_t^\top 1\right] \begin{bmatrix} \mathbf{H} & \mathbf{f} \\ \mathbf{0}^\top & c \end{bmatrix} \begin{bmatrix} \mathbf{x}_t \\ 1 \end{bmatrix} = \tilde{\mathbf{x}}_t^\top \mathbf{A} \tilde{\mathbf{x}}_t \tag{A.8}$$

The matrix $\mathbf{A} \in \mathbb{R}^{N+1 \times N+1}$ has $\frac{N(N+1)}{2} + N + 1 = \frac{1}{2}N^2 + \frac{3}{2}N + 1$ degrees of freedom from $\mathbf{H}$, $\mathbf{f}$ and c.

Polynomial SFA with kernel of degree 2

When using a polynomial kernel of degree 2 and support vectors $\{\mathbf{z}_i\}_{i=1..M}$ the output is written as

$$
\begin{aligned}
g(\mathbf{x}_t) &= \sum_{i=1}^{M} w_i k(\mathbf{z}_i, \mathbf{x}_t) = \sum_{i=1}^{M} w_i \left\{ \mathbf{z}_i^\top \mathbf{x}_t + 1 \right\}^2 \\
&= \sum_{i=1}^{M} w_i \left\{ \mathbf{x}_t^\top \left(\mathbf{z}_i \mathbf{z}_i^\top \right) \mathbf{x}_t + (2\mathbf{z}_i)\, \mathbf{x}_t + 1 \right\} \\
&= \sum_{i=1}^{M} w_i \left\{ \tilde{\mathbf{x}}_t^\top \begin{bmatrix} \mathbf{z}_i \mathbf{z}_i^\top & 2\mathbf{z}_i \\ \mathbf{0}^\top & 1 \end{bmatrix} \tilde{\mathbf{x}}_t \right\} = \tilde{\mathbf{x}}_t^\top \left\{ \sum_{i=1}^{M} w_i \begin{bmatrix} \mathbf{z}_i \mathbf{z}_i^\top & 2\mathbf{z}_i \\ \mathbf{0}^T & 1 \end{bmatrix} \right\} \tilde{\mathbf{x}}_t \\
&= \tilde{\mathbf{x}}_t^\top \left\{ \sum_{i=1}^{M} w_i \mathbf{A}_i \right\} \tilde{\mathbf{x}}_t \qquad \text{(A.9)}
\end{aligned}
$$

Now, in order to show equivalence between the two approaches, it is sufficient to construct a set of support vectors which can generate any matrix $\mathbf{A}$ respecting the constraints from A.8, namely as a linear combination of the support vector induced elementary matrices $\mathbf{A}_i$ with weights w_i.

Construction of the basis set

The (sparse) support vectors used for the construction are defined to have the form $\mathbf{z}_{uv}^{ab} = (0, \ldots, a, \ldots, b, \ldots, 0)^\top$ which means that component u of the support vector has the entry a and component number v has a value of b, whereas the remaining entries equal zero. The notation is ambiguous as $\mathbf{z}_{uv}^{ab} = \mathbf{z}_{vu}^{ba}$. Therefore, the constraint $u < v$ is imposed. Furthermore, $\mathbf{z}_{uv}^{a0}$, $\mathbf{z}_{uv}^{0a}$ and $\mathbf{z}_{uv}^{00}$ are simply written as $\mathbf{z}_u^a$, $\mathbf{z}_v^b$ and $\mathbf{z}^0$.

With $\mathbf{z}^0 := \mathbf{0}$ and $\mathbf{z}_u^1 := (0, \ldots, 1, \ldots, 0)^\top$ one has $1 + N$ support vectors yielding the following matrices:

$$
\mathbf{A}^0 = \left(\begin{array}{ccccc|c} \ddots & & & & & \vdots \\ & 0 & 0 & 0 & & 0 \\ & 0 & 0 & 0 & & 0 \\ & 0 & 0 & 0 & & 0 \\ & & & & \ddots & \vdots \\ \hline & & \mathbf{0}^\top & & & 1 \end{array} \right) \qquad \mathbf{A}_u^1 = \left(\begin{array}{ccccc|c} \ddots & & & & & \vdots \\ & 0 & 0 & 0 & & 0 \\ & 0 & 1 & 0 & & 2 \\ & 0 & 0 & 0 & & 0 \\ & & & & \ddots & \vdots \\ \hline & & \mathbf{0}^\top & & & 1 \end{array} \right)
$$

Furthermore, $\mathbf{z}_{uv}^{11} := (0, .., 1, .., 1, .., 0)^\top$ yields $\frac{N(N-1)}{2}$ additional support vectors and $\mathbf{z}_{1v}^{13} = (0, .., 1, .., 3, .., 0)^\top$ (together with $\mathbf{z}_{u1}^{31}$ to take into account the first column) contributes N additional support vectors and induce matrices:

$$
\mathbf{A}_{uv}^{11} = \left(\begin{array}{ccccc|c} \ddots & & 0 & & & \vdots \\ & 1 & \cdots & 1 & & 2 \\ 0 & \vdots & \ddots & \vdots & 0 & \vdots \\ & 1 & \cdots & 1 & & 2 \\ & & 0 & & \ddots & \vdots \\ \hline & & \mathbf{0}^\top & & & 1 \end{array} \right) \qquad \mathbf{A}_{1v}^{13} = \left(\begin{array}{ccccc|c} 1 & 0 & \cdots & 3 & & 2 \\ 0 & \ddots & & \vdots & & \vdots \\ \vdots & & 0 & 0 & & 0 \\ 3 & \cdots & 0 & 9 & & 6 \\ & & & & \ddots & \vdots \\ \hline & & \mathbf{0}^\top & & & 1 \end{array} \right)
$$

Here, $\{\cdots, \ddots, \vdots\}$ and empty parts are placeholders for zeros. From that definition can be seen that with

$$\mathbf{A}^{const} := \mathbf{A}^0 \qquad \mathbf{A}^{sym}_{uv} := \mathbf{A}^{11}_{uv} - \mathbf{A}^1_u - \mathbf{A}^1_v + \mathbf{A}^0$$
$$\mathbf{A}^{diag}_v := \tfrac{1}{6}\mathbf{A}^{13}_{1v} - \tfrac{1}{2}\mathbf{A}^{11}_{1v} + \tfrac{1}{3}\mathbf{A}^1_1 \qquad \mathbf{A}^{lin}_v := \mathbf{A}^1_v - \mathbf{A}^{diag}_v - \mathbf{A}^0$$

we get a basis the space of matrices $\mathbf{A}$.

$$\mathbf{A}^{const} = \left(\begin{array}{cccc|c} \ddots & & & & \vdots \\ & 0 & 0 & 0 & 0 \\ & 0 & 0 & 0 & 0 \\ & 0 & 0 & 0 & 0 \\ & & & \ddots & \vdots \\ \hline & & \mathbf{0}^\top & & 1 \end{array}\right) \qquad \mathbf{A}^{sym}_{uv} = \left(\begin{array}{ccccc|c} \ddots & & 0 & & & \vdots \\ & 0 & \cdots & 1 & & 0 \\ 0 & \vdots & \ddots & \vdots & 0 & \vdots \\ & 1 & \cdots & 0 & & 0 \\ & & 0 & & \ddots & \vdots \\ \hline & & & \mathbf{0}^\top & & 0 \end{array}\right)$$

$$\mathbf{A}^{diag}_v = \left(\begin{array}{cccc|c} \ddots & & & & \vdots \\ & 0 & 0 & 0 & 0 \\ & 0 & 1 & 0 & 0 \\ & 0 & 0 & 0 & 0 \\ & & & \ddots & \vdots \\ \hline & & \mathbf{0}^\top & & 0 \end{array}\right) \qquad \mathbf{A}^{lin}_v = \left(\begin{array}{cccc|c} \ddots & & & & \vdots \\ & 0 & 0 & 0 & 0 \\ & 0 & 0 & 0 & 1 \\ & 0 & 0 & 0 & 0 \\ & & & \ddots & \vdots \\ \hline & & \mathbf{0}^\top & & 0 \end{array}\right)$$

A.2.2 Estimation of covering numbers for RBF kernels

The aim is to cover the data manifold by balls of radius σ where the number M of balls needed for a covering is of special interest.

The volume of the data manifold can be approximated by the eigenvalues of its covariance matrix, that define the length of the axis of a d-dimensional ellipsoid $V_e \approx v_{d,1} \prod_{i=1}^{d} \sqrt{\lambda_i}$ where $v_{d,1}$ denotes the volume of a d-dimensional unit sphere and $V_s = v_{d,1}\sigma^d$ is the volume of a ball with radius σ in d dimensions.

The number M of balls with radius σ that is needed to entirely cover the ellipsoid can be bounded as follows. At least the volume of the ellipsoid is required in terms of balls. Therefore, a lower bound on M would be $V_e/V_s = \prod_{i=1}^{d} \frac{\sqrt{\lambda_i}}{\sigma} \leq M$.

On the other hand, one can construct a special covering in order to give an upper bound. The ellipsoid is completely covered by a parallelepiped of side lengths $2\sqrt{\lambda_i}$. If the parallelepiped is covered then the ellipsoid will be covered as well. A covering of the parallelepiped will be achieved if one assumes a grid of width δ_{grid} and a ball at each grid point. The distance between centers of neighboring balls on the grid is also given by δ_{grid}. Each hypercube defined by the grid has a longest diagonal of length $\delta_{grid}\sqrt{d}$. When coverage is achieved along each direction of the rectangular grid and along the longest diagonal than the whole grid is covered. In other words

when the centers of the hypercubes forming the grid are covered than the whole grid is covered. That is the case if one chooses $\delta_{grid} \leq \frac{\sigma}{\sqrt{d}}$. In direction of the principal axis i of the ellipsoid one needs with this covering $M_i \leq \lceil 2\sqrt{\lambda_i d}/\sigma \rceil$ balls. Overall, along all axes one finds a total number of $M \leq \prod_{i=1}^{d} \lceil 2\sqrt{\lambda_i d}/\sigma \rceil$.

$$M_{low} = \prod_{i=1}^{d} \frac{\sqrt{\lambda_i}}{\sigma} \leq M \leq \prod_{i=1}^{d} \left\lceil \frac{2}{\sigma}\sqrt{\lambda_i d} \right\rceil = M_{upp} \tag{A.10}$$

Clearly, the number of balls M scale exponential with the dimension $M \propto \sigma^{-d}$. Taking as example a patch of size 10×10 which means $d = 100$ of gray value pixels from $[0, 255]$.

The scatter plot of Figure 4.6 gives an idea of the value's order of magnitude. The 100-dimensional parallelepiped has edges of length 255 and a hyperdiagonal of length $255\sqrt{100} = 2,550$. The longest axis has the length $2\sqrt{\lambda_1} = 815$ and the shortest axis has a length of around $2\sqrt{\lambda_{100}} = 30$.

$$M_{low} = \prod_{i=1}^{d} \frac{\sqrt{\lambda_i}}{\sigma} \leq M \leq \prod_{i=1}^{d} \left\lceil \frac{20}{\sigma}\sqrt{\lambda_i} \right\rceil = M_{upp}$$

A (for a computational implementation) convenient lower bound for the number of balls is reached for different radii σ.

σ	21	22	23	24	σ	700	800	900	1000	1200	2000
M_{low}	1.8M	17k	206	3	M_{upp}	7.1M	101k	12k	5184	224	20

Abbreviations

BSS	Blind Source Separation, page 13
CCD	Charge-Coupled Device, page 4
CPU	Central Processing Unit, page 35
DFG	Deutsche Forschungsgemeinschaft (German Research Foundation), page 2
eSFA	Expanded Slow Feature Analysis, page III
flop	Floating Point Operation, page 35
flops	Floating Point Operations per Second, page 35
FVS	Feature Vector Selection, page 30
GPS	Global Positioning System, page 3
HECGD	Hyper-Elliptical Conjugate Gradient Descent, page 52
ICA	Independent Component Analysis, page 7
IEEE	Institute of Electrical and Electronics Engineers, page 4
JHECGD	Joint Hyper-Elliptical Conjugate Gradient Descent, page 55
kSFA	Kernelized Slow Feature Analysis, page III
MNIST	National Institute of Standards and Technology (dataset), page 17
OOP	Object Oriented Programming, page 28
PC	Personal Computer, page 3
PCA	Principal Component Analysis, page 7
SFA	Slow Feature Analysis, page 7
SVM	Support Vector Machine, page 47
TDSEP	Time-Delayed Separation, page 14

Bibliography

Shun-ichi Amari and Hiroshi Nagaoka. *Methods of Information Geometry*. Oxford University Press, 2000.

Gaston Baudat and Fatiha Anouar. Kernel-based methods and function approximation. In *IJCNN*, 2001. URL `http://www.kernel-machines.org/papers/upload_8504_JCNNKernel.pdf`. 49

Pietro Berkes. Pattern recognition with slow feature analysis. Cognitive Sciences EPrint Archive (CogPrints) 4104,, February 2005a. URL `http://cogprints.org/4104/01/Berkes2005a-preprint.pdf`. 17

Pietro Berkes. *Temporal slowness as an unsupervised learning principle: self-organization of complex-cell receptive fields and application to pattern recognition*. PhD thesis, Humboldt-Universität, Berlin, 2005b. URL `http://edoc.hu-berlin.de/dissertationen/berkes-pietro-2005-06-23/PDF/berkes.pdf`. 7

Pietro Berkes and Laurenz Wiskott. Slow feature analysis yields a rich repertoire of complex cell properties. *Journal of Vision*, 5(6): 579–602, July 2005. URL `http://journalofvision.org/5/6/9/Berkes-2005-jov-5-6-9.pdf`. 16

Pietro Berkes and Laurenz Wiskott. On the analysis and interpretation of inhomogeneous quadratic forms as receptive fields. *Neural Computation*, 18(8):1868–1895, 2006. URL `http://cogprints.org/4081/01/quadratic.pdf`. 16, 29

Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, November 1995.

Tobias Blaschke. *Independent Component Analysis and Slow Feature Analysis: Relations and Combination*. PhD thesis, Humboldt-Universität, Berlin, 2005. URL `http://edoc.hu-berlin.de/dissertationen/blaschke-tobias-2005-02-02/PDF/Blaschke.pdf`. 7, 14

Tobias Blaschke and Laurenz Wiskott. Independent slow feature analysis and nonlinear blind source separation. In *Proc. of the 5th Int. Conf. on Independent Component Analysis and Blind Signal Separation ICA'04, Granada*, Lecture Notes in Computer Science, Granada, Spain, September 2004.

Springer. URL http://itb.biologie.hu-berlin.de/~wiskott/Publications/BlasWisk2004b-ISFA-ICA2004.pdf. 14

Alistair Bray and Dominique Martinez. Kernel-based extraction of slow features: Complex cells learn disparity and translation invariance from natural images. *NIPS*, 15:253–260, 2002. URL http://books.nips.cc/papers/files/nips15/NS20.pdf. 17, 18, 49

I. N. Bronstein, K. A. Semendjajew, G. Musiol, and H. Mühlig. *Taschenbuch der Mathematik*. Verlag Harri Deutsch, 5th edition, June 2000.

Christoph Drösser. Blitzrechnen ohne Geist. *DIE ZEIT*, 29:33, July 2006. URL http://www.zeit.de/2006/29/T-Intelligenz. 1

Richard O. Duda. *Pattern Classification*. Wiley-Interscience, 2nd edition, January 2001.

Alan Edelman, Tomás A. Arias, and Steven T. Smith. The geometry of algorithms with orthogonality constraints. *SIAM Journal on Matrix Analysis and Applications*, 20(2):303–353, 1999. URL http://epubs.siam.org/fulltext/SIMAX/volume-20/29095.pdf. 54

Wolfgang Einhäuser, Jörg Hipp, Julian Eggert, Edgar Körner, and Peter König. Learning viewpoint invariant object representations using a temporal coherence principle. *Biological Cybernetics*, 93:79–90, July 2005. URL http://www.klab.caltech.edu/~wet/BC2005.pdf. 7

Simone Fiori. Unsupervised neural learning on lie group. *International Journal of Neural Systems*, 12(3-4):219–246, 2002. URL http://www.unipg.it/~sfr/publications/LLG.pdf. 54

Peter Földiák. Learning invariance from transformation sequences. *Neural Computation*, 3(2):194–200, 1991. URL http://psy.st-andrews.ac.uk/foldiak/pub.html. 10

Gene H. Golub and Charles F. van Loan. *Matrix Computations*. John Hopkins University Press, 3rd edition, 1996.

J. H. van Hateren and Daniel L. Ruderman. Independent component analysis of natural image sequences yields spatio-temporal filters similar to simple cells in primary visual cortex. In *Proc. R. Soc. Lond. B*, volume 265, pages 2315–2320, 1998. URL http://hlab.phys.rug.nl/papers/van-hateren_ruderman_98.pdf. 7

Simon Haykin. *Neural Networks - A Comprehensive Foundation*. Prentice Hall, 2nd edition, July 1998.

Jarmo Hurri. *Computational Models Relating Properties of Visual Neurons to Natural Stimulus Statistics*. PhD thesis, Helsinki University of Technology (Espoo, Finland), 2003. URL http://lib.tkk.fi/Diss/2003/isbn951226823X/. 6

Aapo Hyvärinen. Survey on independent component analysis. *Neural Computing Surveys*, 2:94–128, 1999. URL http://www.cis.hut.fi/~aapo/ps/NCS99.pdf. 7

Aapo Hyvärinen and Patrik Hoyer. Emergence of phase- and shift-invariant features by decomposition of natural images into independent feature subspaces. *Neural Computation*, 12:1705–1720, 2000. URL http://www.cs.helsinki.fi/u/phoyer/papers/pdf/NC00_complex.pdf. 7

Aapo Hyvärinen, Jarmo Hurri, and Jaakko Väyrynen. Bubbles: a unifying framework for low-level statistical properties of natural image sequences. *Journal of the Optical Society of America*, 20(7):1237–1252, 2003. URL http://lib.tkk.fi/Diss/2003/isbn951226823X/article7.pdf. 8

Bernd Jähne. *Digitale Bildverarbeitung*. Springer, 6th edition, January 2005.

Christoph Kayser, Wolfgang Einhäuser, Olaf Dümmer, Peter König, and Konrad Körding. Extracting slow subspaces from natural videos leads to complex cells. In *Artificial Neural Networks - ICANN*, pages 1075–1080. Springer, 2001. URL http://www.koerding.com/pubs/Kayseretal2001.pdf. 10

Helmut Lütkepohl. *Handbook of Matrices*. Wiley, December 1997.

Lutz Molgedey and Heinz G. Schuster. Separation of a mixture of independent signals using time delayed correlations. *Phys. Rev. Lett.*, 72(23):3634–3637, June 1994. URL http://www.theo-physik.uni-kiel.de/thesis/molgedey94.ps.gz. 14

Marek Musial, Günter Hommel, and Klaus Obermayer. Neuronale biologisch inspirierte Steuerungsarchitektur für einen mobilen Roboter. Neuantrag auf Gewährung einer Sachbeihilfe bei der Deutschen Forschungsgemeinschaft, July 2004. 2

Bruno A. Olshausen and David J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 13:607–9, June 1996. 7

Mark D. Plumbley. Lie group methods for optimization with orthogonality constraints, 2004. URL http://www.elec.qmul.ac.uk/staffinfo/markp/2004/Plumbley04-ica-geometry.pdf. 54

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 2nd edition, February 1993. 30

Daniel L. Ruderman and William Bialek. Statistics of natural images: Scaling in the woods. *Physical Review Letters*, 73(6):814–817, August 1994. URL http://prola.aps.org/pdf/PRL/v73/i6/p814_1. 6

Arjen van der Schaaf. *Natural Image Statistics and Visual Processing*. PhD thesis, Graduate School for Behavioural and Cognitive Neurosciences, Rijksuniversiteit Groningen, 1998. URL http://dissertations.ub.rug.nl/faculties/science/1998/a.v.d.schaaf/. 6

Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels*. The MIT Press, 2002.

Bernhard Schölkopf, Alexander J. Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998. URL http://www.kernel-machines.org/papers/nlpca.ps.gz. 17

Alexander J. Smola, Olvi L. Mangasarian, and Bernhard Schölkopf. Sparse kernel feature analysis, 1999. URL http://www.kernel-machines.org/papers/SmoManSch99.ps.gz. 51

Anuj Srivastava, Ann B. Lee, Eero P. Simoncelli, and Song-Chun Zhu. On advances in statistical modeling of natural images. *Journal of Mathematical Imaging*, 18(1):17–33, January 2003. URL http://www.cavis.fsu.edu/pdf/journal/4-On_Advances_in_Statistical_Modeling_of_Natural_Images.pdf. 6

James V. Stone. Learning perceptually salient visual parameters using spatiotemporal smoothness constraints. *Neural Computation*, 8:1463–1492, 1996. URL ftp://ftp.shef.ac.uk/pub/misc/personal/pc1jvs/papers/ti_nc96.ps.gz. 10, 17

James V. Stone and Alistair Bray. A learning rule for extracting spatio-temporal invariances. *Network*, 3(2):1–8, 1995. URL ftp://ftp.shef.ac.uk/pub/misc/personal/pc1jvs/papers/network_final_web.ps.gz. 10

Michael E. Tipping. Sparse kernel principal component analysis. In *NIPS*, pages 633–639, 2000. URL ftp://ftp.research.microsoft.com/users/mtipping/skpca_nips.ps.gz. 51

Antonio Torralba and Aude Oliva. Statistics of natural image categories. In *Network: computation in neural systems*, volume 14, pages 391–412, 2003. URL http://web.mit.edu/torralba/www/ne3302.pdf. 6

Richard Turner and Maneesh Sahani. A maximum-likelihood interpretation for slow feature analysis. *Neural Computation*, accepted, 2006. URL http://www.gatsby.ucl.ac.uk/~turner/SFA/TSNCOMP2006v9.pdf. 15

Roland Vollgraf and Klaus Obermayer. Sparse optimization for second order kernel methods. In *International Joint Conference on Neural Networks (IJCNN)*, 2006. URL ftp://ftp.cs.tu-berlin.de/pub/local/ni/papers/Vollgraf_2005_SigProc.pdf. 29, 51, 54

Roland Vollgraf, Michael Scholz, I. A. Meinertzhagen, and Klaus Obermayer. Nonlinear filtering of electron micrographs by means of support vector regression. In *Advances in Neural Information Processing Systems, NIPS*, pages 717–724, 2004. URL ftp://ftp.cs.tu-berlin.de/pub/local/ni/papers/vro_04_nips.pdf. 39

Laurenz Wiskott. How does our visual system achieve shift and size invariance? Cognitive Sciences EPrint Archive (CogPrints) 3321, December 2003a. URL http://itb1.biologie.hu-berlin.de/~wiskott/Publications/Wisk2003a-Invariances-CogPrints.pdf. 6

Laurenz Wiskott. Slow feature analysis: Theoretical analysis of optimal free responses. *Neural Computation*, 15(9):2147–2177, September 2003b. URL http://itb.biologie.hu-berlin.de/~wiskott/Publications/Wisk2003b-SFATheoryFree-NC.pdf. 12

Laurenz Wiskott. Estimating driving forces of nonstationary time series with slow feature analysis. arXiv.org e-Print archive, December 2003c. URL http://itb.biologie.hu-berlin.de/~wiskott/Publications/Wisk2003c-SFA-Application-arXiv.pdf. 16

Laurenz Wiskott and Terrence Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural Computation*, 14(4):715–770, 2002. URL http://itb.biologie.hu-berlin.de/~wiskott/Publications/WisSej2002-LearningInvariances-NC.ps.gz. 7, 10, 11, 13

Andreas Ziehe and Klaus-Robert Müller. TDSEP - an efficient algorithm for blind separation using time structure. In *ICANN 1998*, 1998. URL http://wwwold.first.gmd.de/persons/Mueller.Klaus-Robert/ICANN_tdsep.ps.gz. 14