Non-linear latent factor models for revealing structure in high-dimensional data

by

Roland Memisevic

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Non-linear latent factor models for

revealing structure in high-dimensional data

Roland Memisevic

Doctor of Philosophy

Graduate Department of Computer Science

University of Toronto

2007

Real world data is not random: The variability in the data-sets that arise in computer vision, signal processing and other areas is often highly constrained and governed by a number of degrees of freedom that is much smaller than the superficial dimensionality of the data. Unsupervised learning methods can be used to automatically discover the "true", underlying structure in such data-sets and are therefore a central component in many systems that deal with high-dimensional data.

In this thesis we develop several new approaches to modeling the low-dimensional structure in data. We introduce a new non-parametric framework for latent variable modelling, that in contrast to previous methods generalizes learned embeddings beyond the training data and its latent representatives. We show that the computational complexity for learning and applying the model is much smaller than that of existing methods, and we illustrate its applicability on several problems.

We also show how we can introduce supervision signals into latent variable models using conditioning. Supervision signals make it possible to attach "meaning" to the axes of a latent representation and to untangle the factors that contribute to the variability in the data. We develop a model that uses conditional latent variables to extract rich distributed representations of image transformations, and we describe a new model for learning transformation features in structured supervised learning problems.

Acknowledgements

The most important acknowledgement and thanks goes to Geoff Hinton for being an amazing teacher and inspiration for me over the last few years, and for his great support and his belief in my abilities throughout the time I spent at the University of Toronto.

Also, I thank my committee members and associated faculty, Yoshua Bengio, David Fleet, Brendan Frey, Radford Neal, Sam Roweis and Rich Zemel for their valuable feedback and support. For the many interesting and helpful discussions at the University of Toronto I thank the members of the machine learning group, including Xuming He, Jenn Listgarten, Ben Marlin, Ted Meeds, Andriy Mnih, Iain Murray, Vinod Nair, Rama Natarajan, Simon Osindero, David Ross, Ruslan Salakhutdinov, Horst Samulowitz, Nati Srebro, Liam Stewart, Ilya Sutskever, Graham Taylor and Tijmen Tieleman.

I thank my parents and my family for their constant support. And I thank my wife Tatiana for being with me and for giving me the motivation to see this thesis through to completion.

My work has been financially supported by a Government of Canada Award.

Contents

Chapter 1

Introduction

1.1 Unsupervised learning

High-dimensional data occurs in a wide range of domains, and many common tasks involve processing this kind of data. Often, real-world data either has a natural representation as a set of high-dimensional points (images, for example, can be represented as vectors of pixel-intensities for use in classification), or they can be forced into this representation while retaining much of the information that is necessary for the task at hand (documents, for example, can be represented as vectors of word counts for use in information retrieval, or songs as vectors of melodic descriptors for use in genre classification). In yet other applications, such as motion tracking or document understanding, data is given by a *sequence* of vectors, rather than a set of independent observations. In all these problems, the basic building blocks for representing real-world data are high-dimensional vectors of observations.

Unsupervised learning is the sub-area of machine learning that uncovers structure, that can be hidden in such data-sets. Looking for structure has several motivations, all related to well-known problems with high-dimensionality. The most obvious and simple is that saving and handling very large vectors can consume a lot of space and computation time. If there is some form of *redundancy* in the data, finding it makes it possible to reduce the size and

thereby the burden associated with saving and handling the data.

Unsupervised learning can have additional benefits: Processing sequences of numbers is not what human brains have been designed for or are good at. This makes it difficult to discover trends or other interesting properties in high-dimensional data. Reducing the dimensionality of data, *without loosing the essential information*, can be a useful prelude to further analysis. In particular, reducing the dimensionality to 2 or 3 makes it possible to get a visual understanding of data, since it allows us to generate plots in a 2- or 3-dimensional space.

A deeper reason for looking for structure is that the ability to capture structure lies at the heart of intelligence and creativity itself. Being able to *compress* data, by extracting its essence and discarding the rest, is a necessary condition for *understanding* the data. Models that can grasp the "meaning" of data often play an important role in systems that can process data intelligently.

The task of "making sense" of data can be defined formally as bringing it into a new representation in which meaning is made explicit. Consider, for example, a high-dimensional retinal image of a visual scene showing objects of different sizes and configurations. The true, low-dimensional causes of variability in this and in similar data-sets are represented implicitly as subtle and complex interactions of pixels. Mapping high-dimensional vectors of pixel values into a low-dimensional space can make it possible to obtain a more explicit representation of the low-dimensional causes of variability. Dimensionality reduction methods therefore can be an important part of artificial intelligence systems that try to perceive and, in some restricted sense, "understand" real-world data.

Research in dimensionality reduction (aka. "data embedding") tries to develop criteria that allow us to define such a mapping in reasonable and useful ways. Since performing a mapping into a low-dimensional space entails loss of information, reducing dimensionality amounts to making a decision about what is the *signal* and what is the *noise* in a given data-set. The signal is what is being preserved and the noise is what is being discarded.

Developing an embedding method thus usually comes down to defining "signal" and "noise" in ways that are useful for applications, and at the same time ensuring feasibility of the resulting computational procedures. The most well-known embedding method is principal components analysis (PCA) (see [Jolliffe, 1986]), which is based on the objective of preserving variance, so that the signal is the projection of the data into the subspace with the largest variance. Other criteria include probabilistic considerations [Hinton and Roweis, 2003] or the preservation of inter-point similarities [Roweis and Saul, 2000].

Most dimensionality reduction methods come with an appealing geometric intuition: Data is distributed along a low-dimensional *manifold* lying in the much higher-dimensional data-space (see Figure 2.1 for an illustration). Variability along the manifold is the signal, variability orthogonal to it is noise. "Manifold learning" is a common alternative term for embedding. A low-dimensional manifold can be parameterized using a low-dimensional set of parameters, and embedding amounts to projecting the observed data into this parameter-space. The low-dimensional parameters are often referred to as *latent variables*.

1.2 Untangling variability

Data embedding methods have been used in a huge number of applications, which shows that the idea of extracting the signal from noisy data is a useful metaphor in practice. However, it is also well-known that the separation of data variability into signal and noise is actually quite simplistic and does often not do justice to the complex and subtle ways in which real world data can be structured [Tenenbaum and Freeman, 2000], [Vasilescu and Terzopoulos, 2002], [Bowling et al., 2005], [Lee and Elgammal, 2005].

As an example, consider the scene mentioned above. If the variability in the data would be caused by the two factors, "lighting" and "viewpoint", a standard embedding method would compute a latent representation that preserves exactly these two factors. Depending on the application, however, we might be interested *only* in lighting or *only* in viewpoint variation.

In other words, what we regard as the signal and what we regard as noise is often application dependent rather than generic.

In other cases it might a subtle factor that does not contribute much to the overall variance that is the most important to a particular application. A related issue is, that in many real world problems the knowledge that is available about data is not just the set of high-dimensional vectors itself, but there might be additional information available. Sometimes a data-set can be partitioned naturally into classes, or specific factors such as lighting variation can be easily extracted beforehand. Standard embedding methods are blind to any type of extra information, and the standard embedding metaphor does not provide the space for extensions to include these and similar situations. In these and similar cases, applications can profit from regarding embedding as a more complex process than the simple projection of data onto a low-dimensional manifold (see *eg.* , [Tenenbaum and Freeman, 2000]).

Technically, embedding data with standard methods usually comes down to modelling pair-wise interactions – between observed variables and latent variables. Introducing any form of extra information inevitably comes with the requirement of using *multi-way* interactions – between observations, latent variables and the available extra-information. One of the challenges of performing data embedding in non-standard ways is to model higher-order interactions efficiently. Example approaches include [Vasilescu and Terzopoulos, 2002] [Tenenbaum and Freeman, 2000].

In this thesis we develop a range of novel approaches that allow us to perform embedding in semi-supervised ways. We develop methods that combine multiple sources of information efficiently in order to attach "meaning" to the axes of a latent space. We show in particular, that often we can use *conditioning* in a probabilistic framework as a way to include multiple sources of information. Formally, many embedding methods can be expressed as models of a joint probability density $p(\boldsymbol{y}, \boldsymbol{z})$ $(= p(\boldsymbol{y}|\boldsymbol{z})p(\boldsymbol{z}))$ over observed data $\boldsymbol{y}$ and low-dimensional latent variables $\boldsymbol{z}$. Methods that are of this form include PCA [Jolliffe, 1986], ICA [Hyvärinen, 1999], more recent non-parametric models, such as Gaussian process latent

variable models [Lawrence, 2004], and many others. One of the (many) justifications for this probabilistic approach is that maximizing the log-probability of the observations under this type of model is under certain conditions equivalent to maximizing an estimate of the *mutual information* ([Cover and Thomas, 1990]) between the latent variable $\boldsymbol{z}$ and the observed data $\boldsymbol{y}$ (see *eg.* , [Baldi and Hornik, 1995]) – We obtain latent factors that capture as much of the information contained in the observed data as possible.

In order to obtain *informed* embeddings within the probabilistic framework, we suggest introducing an additional random variable $\boldsymbol{x}$ that encodes available extra information, and model the *conditional density* $p(\boldsymbol{y}, \boldsymbol{z}|\boldsymbol{x})$ $(= p(\boldsymbol{y}|\boldsymbol{z}, \boldsymbol{x})p(\boldsymbol{z}|\boldsymbol{x}))$, over observed data $\boldsymbol{y}$ and latent variables $\boldsymbol{z}$. Embedding models now become functions of prior knowledge and they encode variability *in relation to* that prior knowledge. A conditional model can also be thought of as modeling data *transformations* (of the conditioning information $\boldsymbol{x}$, to obtain the data $\boldsymbol{y}$). By choosing the conditioning information appropriately, invariances can be encoded, and latent spaces can be constructed in a supervised way. An advantage of probabilistic conditioning is that it allows us to include more general types of information than with many existing methods. Furthermore, conditioning is possible for a variety of models, both parametric, and non-parametric, and can provide efficient solutions to the problem of modeling multi-way interactions.

1.3 Thesis Outline

The following chapter provides an overview of learning with latent variables and discusses in detail connections and differences between existing methods, such probablistic latent variable models, non-parametric methods and neural networks. That chapter also discusses the limitations of existing methods, and described how supervision signals can be used to make embedding methods more useful in applications. The chapter concludes by describing the related problem of prediction with high-dimensional outputs. The chapters that follow (Chapters 3

to 6) constitute the main part of the thesis, which is followed by concluding remarks (Chapter 7).

The four main chapters of the thesis can be divided into two main parts. The first part (Chapters 3 and 4) focusses on non-parametric embedding. Chapter 3 describes a general framework for non-linear embedding based on maximizing non-parametric estimates of mutual information. Side-information can be introduced within the framework by replacing mutual information with conditional mutual information. This leads to the notion of a shared latent space in which part of the available dimensions are used to encode side-information, while the other dimensions encode the remaining variability in a data-set. Chapter 4 takes the idea further by allowing a latent variable model *itself* to allocate latent space dimensions in order to encode both variability in the data and additional information within a single latent space.

The second major part of the thesis (Chapters 5 and 6) discusses conditional parametric models. Chapter 5 shows how conditioning a parametric model can be achieved by modulating the parameters of a model using supervision signals and describes an application to the task of learning image transformations. Chapter 6 describes a non-probabilistic approach and discusses how a conditional non-linear manifold can be used in structured supervised learning tasks.

Chapter 2

Background and related work

2.1 Dimensionality reduction

Figure 2.1 illustrates the basic dimensionality reduction, or embedding, problem. We are given a set of high-dimensional vectors (elements of $\mathbb{R}^d$, where d is large). In the illustration the vectors are stacked pixel-intensities of grayscale images. We will denote high-dimensional data vectors using the variable $\boldsymbol{y}$ throughout. Although the superficial dimensionality of the data (the number of pixels in each image) is high, the data-set has an *intrinsic dimensionality* that is much lower: The variability in this example data-set is caused only by rotation, a single parameter. So a single number should suffice to describe how any two images differ from each other.

Geometrically, we can think of the data as being distributed along a smooth low-dimensional manifold in the high-dimensional space. The intrinsic dimensionality of that manifold is equal to the number of "true" degrees of freedom, which in the example is one (the rotation angle). In reality, the data-generating process is often not perfect, so there could be noise, or some additional degrees of freedom that we do not care about. Therefore the data that we observe is distributed only *around* some ideal manifold, effectively "thickening it out".

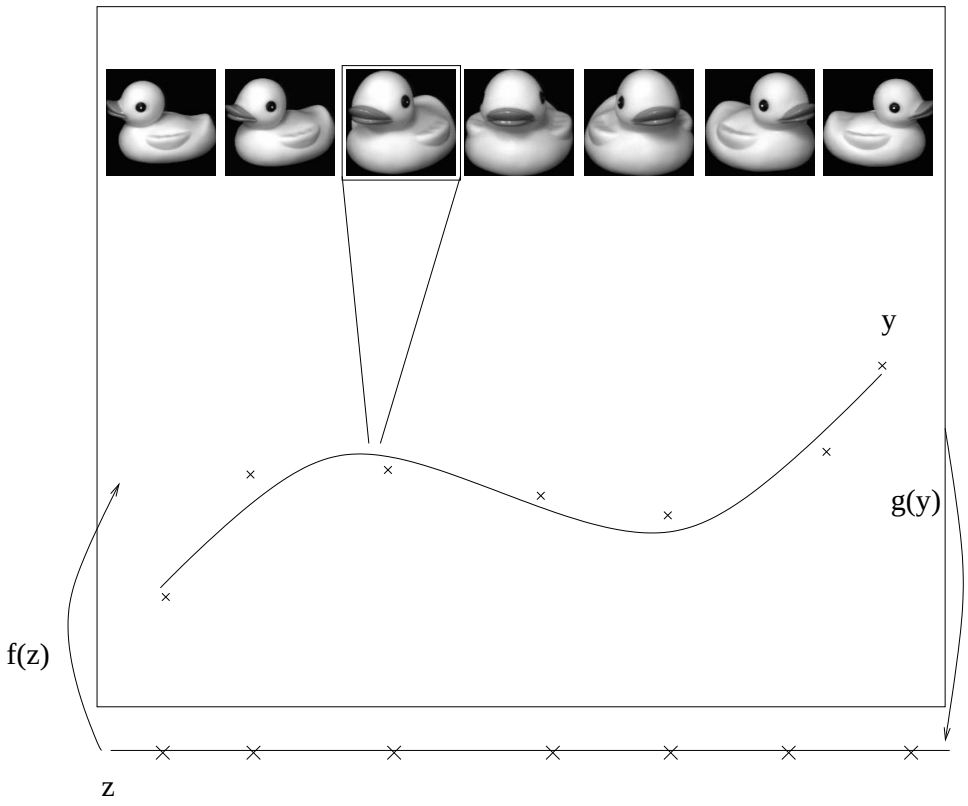

Figure 2.1: Illustration of dimensionality reduction. Data is distributed around some ideal low-dimensional manifold in the high-dimensional data-space. The high-dimensional space is represented by two dimensions in the illustration. The task of data embedding consists of three sub-tasks: (a) finding a low-dimensional representative $\boldsymbol{z}$, for each data-case $\boldsymbol{y}$ (b) finding a mapping f from the low-dimensional to the high-dimensional space, (c) finding a mapping g from the high-dimensional to the low-dimensional space.

The goal of learning is to identify this manifold. Depending on the application, "identifying the manifold" can mean several different things: In many problems, it is sufficient to obtain *low-dimensional representatives* (or "codes") for the data-points. These low-dimensional representatives can be thought of encoding the true degrees of freedom and in some sense capturing the "essence" of the observed data. In the example, one-dimensional representatives would allow us to re-express the images using their rotation angle, and would constitute a highly compressed representation of the data. The low-dimensional codes can also be thought

of also as hidden *causes* of the data and the task of extracting the hidden causes as an *inverse problem* (see, for example, [Palmer, 1999]). Solving this problem is important in many artificial intelligence applications that involve a form of perception, such as computer vision. Extracting the "true" underlying causes of variability is the first step in transforming raw data into *meaning* and in gaining an understanding of the data.

Formally, we can think of a *latent*, or *hidden*, variable $\boldsymbol{z}$, whose dimensionality is much lower than that of the original data, and which represents the underlying causes of the variability in the data. Our goal is to find an instantiation $\boldsymbol{z}^\alpha \in \mathbb{R}^q$ for every high-dimensional point $\boldsymbol{y}^\alpha$. Another common application is data visualization: If the dimensionality of $\boldsymbol{z}$ is not more than 3, it is possible to look at the data by producing a scatter plot of the latent representatives[1].

In other applications, finding low-dimensional representatives is not sufficient. In pattern recognition tasks, for example, it can be useful to have a *mapping* $g(\boldsymbol{y})$ in addition, which can be applied to data-cases $\boldsymbol{y}$ not part of the training set. Since low-dimensional codes represent the "true" variability in the data, they are often cleaner representatives of the variability in data than the original data-points. Applications such as classification therefore often work better when operating in the low-dimensional space. However, in that case we require low-dimensional codes for *test-cases*, that were not seen during training. That is, we need a mapping $g(\boldsymbol{y})$, sometimes referred to as *"backward-mapping"*. The necessity for a backward-mapping is also referred to as "out-of-sample" problem [Bengio et al., 2004].

Other applications require a *forward-mapping* $f(\boldsymbol{z})$ that can map low-dimensional codes into the data-space. The image of this function is the (ideal) manifold in the data-space. Applying it allows us to generate "fantasy"-data, *ie.* data-points that are lying on the manifold but that are not part of the training data. One application that requires a forward-mapping

[1]Reducing the dimensionality to 3 or less at first sight seems to make sense only if the actual number of degrees of freedom is actually 3 or less. However in practice it is common to reduce the dimensionality in this way, even if the number of degrees of freedom is much larger. Capturing "as much of the variability as possible" still allows us to obtain a visual impression of the data that is often useful, even if the true intrinsic dimensionality is much larger.

is lossy compression. Reducing the dimensionality of data, while preserving most of the variability, allows us to encode data compactly without loosing much information. However, to be able to reconstruct the original data from the low-dimensional codes, we need a mapping from the code-space to the data-space.

Another example problem is *de-noising* (see *eg.* , [Kwok and Tsang, 2003]): If the observed data is noisy, it can be thought of as being distributed around the ideal manifold, effectively thickening it, as mentioned above. In that case, $f(\boldsymbol{z}^{\boldsymbol{\alpha}})$ can give us a de-noised version of data-point α: It is the point *on* the ideal manifold corresponding to the ideal low-dimensional descriptor for the data-points. If we learn both a forward- and a backward-mapping from the training data, we can also de-noise previously unseen test-points y^{test}, by computing $f(g(\boldsymbol{y}^{\mathbf{test}}))$. More generally, all applications that involves any kind of *generation* of patterns in the data-space requires the forward-mapping $f(\boldsymbol{z})$ (see [Memisevic, 2003] for some examples).

2.2 Parametric methods

2.2.1 Optimizing reconstructions

A very simple, but very effective and common embedding method can be derived from the assumption that the manifold is a *linear* subspace of the data-space. In this case, every data-point $\boldsymbol{y}$, lying on the (ideal) manifold can be viewed as the projection $W\boldsymbol{z}$ of a low-dimensional code-vector $\boldsymbol{z}$ into the data-space, where W is a $d \times q$ matrix. Because of noise, as mentioned above, the actually observed data-points $\boldsymbol{y}^{\alpha}$ are lying only *near* the ideal manifold. So fitting the manifold can be achieved by minimizing the average squared reconstruction error. In contrast to standard (supervised) regression, we obtain an optimization problem in both, the model parameters W and the latent variable realizations $\boldsymbol{z}^{\alpha}$. Given a data-set $\{\boldsymbol{y}^{\alpha}\}_{\alpha=1\ldots N}$,

we need to minimize:

$$E^{\mathrm{FWD}} = \sum_{\alpha} \|\boldsymbol{y}^{\alpha} - W\boldsymbol{z}^{\alpha}\|^2 = \|\boldsymbol{Y} - W\boldsymbol{Z}\|^2 \tag{2.1}$$

with respect[2] to W and the $\boldsymbol{z}^{\alpha}$, where $\boldsymbol{Y}$ denotes the matrix of all observed data-points (column-wise), and $\boldsymbol{Z}$ denotes the corresponding matrix of latent representatives.

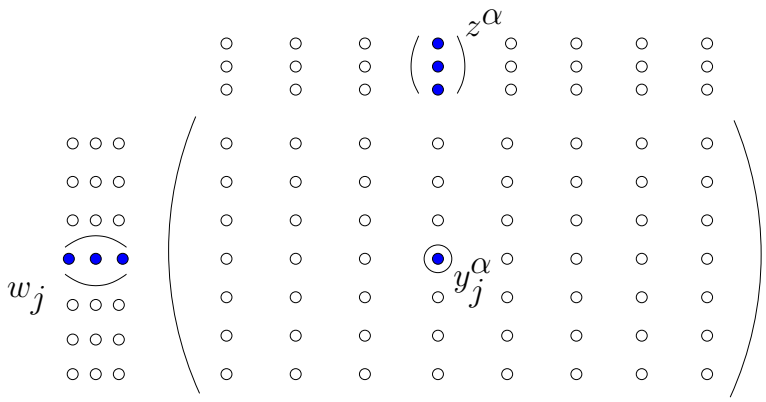

Figure 2.2: The simplest form of parametric embedding is matrix decomposition: Each component y_j^{α} of the data-matrix $\boldsymbol{Y}$ is approximated by the inner product of a row-feature w_j and a column-feature $\boldsymbol{z}^{\alpha}$. Row-features are model parameters, column-features are latent variables.

Note that in the column-wise representation, a row corresponds to a dimension and a column to a data-point. Solving the optimization problem gives us not only low-dimensional data representatives $\boldsymbol{z}^{\alpha}$, but also the forward-mapping, which is simply the linear transformation W itself. Figure 2.2 illustrates this approach: The result of minimizing E^{FWD} is that the data-matrix $\boldsymbol{Y}$ gets de-composed into the product of the two rectangular matrices W and $\boldsymbol{Z}$. Each component y_j^{α} is approximated by the inner product of a "row-feature" vector w_j (a row of W) and a "column-feature" vector $\boldsymbol{z}^{\alpha}$ (a column of $\boldsymbol{Z}$).

[2]Note that there is a degeneracy in this solution: We can, for example, apply any invertible linear transformation to $\boldsymbol{z}$ and the inverse of that transformation to W without affecting the result. One common way to deal with this degeneracy in practice is to ignore it. Below we discuss other solutions to this degeneracy.

The most simple way to train this model is with gradient based optimization of W and $\boldsymbol{Z}$. An alternative is to impose orthogonality constraints on W and $\boldsymbol{Z}$, which turn the problem into a singular value decomposition [Horn and Johnson, 1994], and allow us to use specialized optimization routines. We review this idea briefly below. However, a notable benefit of gradient descent is that it allows us to deal with missing data, simply by ignoring missing entries in the data-matrix during the optimization.

After training, reconstructions can be computed as the inner product $w_j^T \boldsymbol{z}^\alpha$ (which also implies that missing data can be filled in this way, if desired). The computational complexity for training is $O(Ndq)$. Online gradient descent (aka. stochastic gradient descent, see [Bottou, 2004] for an overview) can be used to train the model online, which allows it to deal with very large data-sets.

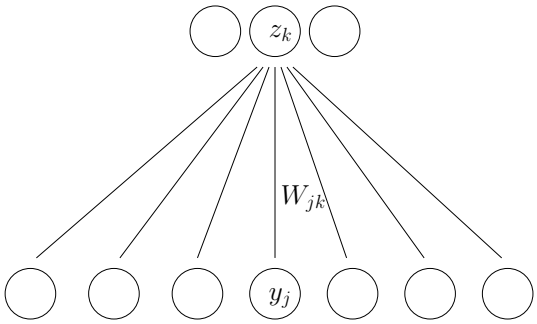

Figure 2.3: Data embedding with a bi-partite network. Model parameters connect latent variables to observed variables, but there are no connections between latent variables or between observed variables.

If the observed data is not mean-centered we need to include means in the approximation by replacing the linear function W with an affine function. That is, in practice we approximate $\boldsymbol{Y}$ with $W\boldsymbol{Z} + B$, where B is a matrix whose rows are constant. The additive terms are usually called biases. In what follows we will drop the biases formally as we did above to simplify the exposition, keeping in mind that the constant terms do get included in practice. Alternatively we can think of the input data as being mean-centered.

Figure 2.3 shows an alternative interpretation of the linear parametric model. Each component W_{jk} of the parameter-matrix can be thought of as connecting a hidden variable z_k to the observed variable y_j. The data and the hidden variables (column-features) are computational units, that are connected to each other via the parameters (row-features). We will use the terms "unit", "component" and "variable" interchangibly in the following. As the illustration shows, the connectivity between hidden units and data-variables forms a *bi-partite* network: Data is reconstructed by using inner products between row-features w and column-features $\boldsymbol{z}$, so there are no multiplicative couplings between any of the components in $\boldsymbol{z}$ or the components in $\boldsymbol{y}$. As a result the computation of the gradients that are needed for the optimization – and of other quantities, too, as we discuss below – is very simple. In the various extensions of this model, that we will discuss below, bi-partite structure will therefore remain a key aspect.

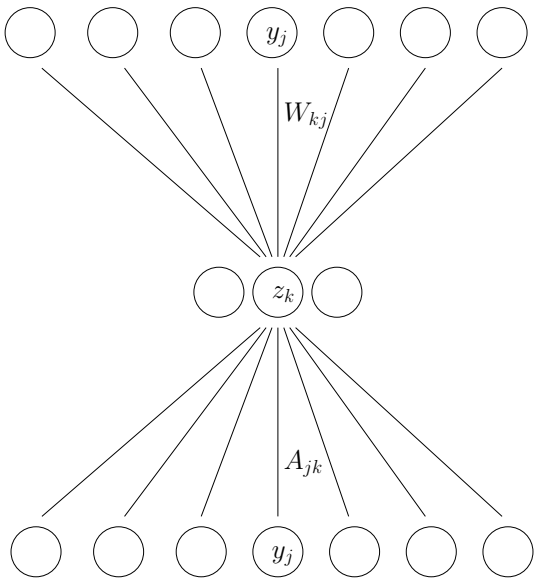

Figure 2.4: An auto-encoder.

Minimizing the objective in Eq. 2.1 gives us the low-dimensional codes $\boldsymbol{Z}$ and the forward mapping W, which is linear. However, it does not give us a backward mapping. Many applications (such as filling in missing data) do not require a backward mapping. Other

applications, in particular those that involve an inverse problem, such as in perceptual tasks, do require this mapping: After training, the running system needs to extract the codes for new, previously unseen, stimuli. One way to obtain the backward mapping in addition to $\boldsymbol{Z}$ and W is by modelling it explicitly and learning it along with the other parameters. If we define the backward-mapping as a linear function $A : \mathbb{R}^d \rightarrow \mathbb{R}^q$, we can use the objective:

$$E^{\mathrm{LAE}} = \sum_{\alpha} \|\boldsymbol{y}^{\alpha} - W A \boldsymbol{y}^{\alpha}\|^2 \tag{2.2}$$

This approach is the well-known *linear auto-encoder*. It can be viewed as an application of a two-layer linear feed-forward neural network ([Bishop, 1995]): The first layer computes the codes $\boldsymbol{z}$ from the data $\boldsymbol{y}$, the second layer projects these back into the data-space to reconstruct the data. Note that, in contrast to previously we do not explicitly optimize the low-dimensional codes in this approach. But the backward-mapping A defines the codes $\boldsymbol{z}^{\alpha}$ for each data-case $\boldsymbol{y}^{\alpha}$ *implicitly*, as $A\boldsymbol{y}^{\alpha}$. This approach is illustrated in figure 2.4. Note again the bi-partite structure in each pair of layers. Here, this structure simplifies the computation of gradients. Training the model is usually done with gradient descent.

Instead of using linear functions for the backward- and forward-mappings, we can use any non-linear ones, as long as they are differentiable, so that the optimization is tractable. One way is to replace the linear mappings A and W by multilayer neural networks. For the resulting overall mapping to be non-linear, the mappings themselves need to contain non-linear hidden layers. A common way to turn a linear layer into a non-linear one is by applying the sigmoid $\sigma(o_i) = \frac{1}{1+\exp(-o_i)}$ to each output-component o_i of a linear mapping. Defining the model this way allows us to maintain the bi-partite, layered structure, which in turn makes it possible to use back-propagation [Rumelhart et al., 1986] to compute gradients.

An alternative variation of the linear approach consists in constraining the columns of W to be orthonormal. We can then define the backward-mapping $g(\boldsymbol{y})$ as the pseudo-inverse of the forward mapping W, which in that case is simply given by W^{T} $\left(= (W^{\mathrm{T}}W)^{-1}W^{\mathrm{T}}\right)$.

Therefore, the objective function can then be re-written as:

$$E^{\mathrm{PCA}} = \|\boldsymbol{Y} - WW^{\mathrm{T}}\boldsymbol{Y}\|^2 \tag{2.3}$$

$$= \mathrm{tr}(\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}}) - \mathrm{tr}(W^{\mathrm{T}}\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}}W). \tag{2.4}$$

After discarding terms that do not depend on W, we can therefore formulate the training problem as a constrained optimization problem:

$$\min_{W} -\mathrm{tr}(W^{\mathrm{T}}MW) \quad \text{subject to} \quad \mathrm{tr}(W^{\mathrm{T}}W) = I, \tag{2.5}$$

with $M = \boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}}$. The solution to problems of the form 2.5 is by the Rayleigh-Ritz-theorem [Horn and Johnson, 1994] given by the leading eigenvectors of M. This fact is sometimes considered beneficial, and is sometimes put forth as a reason for the orthogonality constraint in the first place. It is this spectral solution that is usually referred to as principal components analysis (PCA) (see, for example, [Jolliffe, 1986] or [Hastie et al., 2001]). It should be noted, however, that efficiently computing an eigen-decomposition is not trivial, and can involve iterative optimization procedures itself. For many applications that make use of an eigen-decomposition it is not clear whether a simple gradient-based optimization of the unconstrained problem could not have been used instead.

Nevertheless, PCA has many additional interpretations beyond the one given above, which can be beneficial to certain applications. One well-known property of the eigen-solution is that it also maximizes the variance of the projections $WW^{\mathrm{T}}\boldsymbol{Y}$. In this sense PCA finds the linear subspace that captures most of the variability in the data. There are many further interpretations for PCA, that can be useful, or at least comforting, in some applications. See [Baldi and Hornik, 1995] for a detailed discussion of the relation between linear auto-encoders and PCA.

Note that the matrix M in the PCA eigen-problem (Problem 2.5) is a $d \times d$-matrix, which can be prohibitively large if the data is very high-dimensional. In that case it is possible, and common, to perform an eigen-decomposition of the inner product matrix $\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Y}$ instead. This is possible, because of a simple duality property of the eigen-decomposition: If v is

an eigenvector of $\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}}$ and λ the corresponding eigenvalue, then we have (by definition) $\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}}v = \lambda v$. But multiplying with $\boldsymbol{Y}^{\mathrm{T}}$ from the left shows that in that case we must also have: $\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}}v = \lambda \boldsymbol{Y}^{\mathrm{T}}v$. In other words $\boldsymbol{Y}^{\mathrm{T}}v$ must be an eigenvector of $\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Y}$. But we can also write $\boldsymbol{Z}^{\mathrm{T}} = \boldsymbol{Y}^{\mathrm{T}}V$, where V contains the eigenvectors columnwise. Therefore, if instead of solving Problem 2.5 we perform an eigen-decomposition of $\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Y}$, we directly obtain the latent representatives. This observation is also the basis for kernel PCA [Schölkopf et al., 1998], which we describe in more detail below.

We have left open the choice of the latent space dimensionality for now (the number of latent variables). This number depends on what we believe to be the "true" number of degrees of freedom in the data (up to noise). In many applications, this number is not known beforehand, but it can often be guessed roughly. The behavior of systems that use dimensionality reduction as one of their components is usually not very sensitive to this choice. If necessary, it is also possible to choose the dimensionality by cross-validation.

Another issue that we did not discuss in detail is the choice of loss function. We used squared error in all of the objective functions above. While this is a computational convenience (for example, derivatives are easy to calculate in this case), other cost functions are more appropriate in some situations, for example when dealing with discrete data. Many cost-functions can be derived conveniently from probabilistic modelling, a viewpoint that we describe in more detail below.

2.2.2 Generative models

The previous section discussed dimensionality reduction from the perspective of approximating the data-matrix by minimizing reconstruction error. A common alternative is the perspective of probabilistic *generative models*. According to this view, we can learn about the structure in data by building a model of the stochastic process that generated the data [Jordan, 2008]. Similarly as in the previous section, latent variables can be part of the model to pick up structure that is hidden in the variability of the data.

To model data $\boldsymbol{y}$ using latent variables $\boldsymbol{z}$ it is common to build a model of the joint distribution $p(\boldsymbol{y}, \boldsymbol{z})$, and then to *marginalize* over $\boldsymbol{z}$ to get the probability distribution of the data as

$$p(\boldsymbol{y}) = \int p(\boldsymbol{y}, \boldsymbol{z}) \, \mathrm{d}\boldsymbol{z} \tag{2.6}$$

It is common in embedding tasks to decompose the joint distribution as

$$p(\boldsymbol{y}, \boldsymbol{z}) = p(\boldsymbol{y}|\boldsymbol{z})p(\boldsymbol{z}) \tag{2.7}$$

The conditional $p(\boldsymbol{y}|\boldsymbol{z})$ is usually referred to as "observation model", because it encodes how, given instantiations of the latent variables, the observed data is produced stochastically. Note that the use of probabilistic latent variables requires that we adopt an interpretation of probabilities as encoding beliefs rather than relative frequencies of repeatable events. In other words, generative models are *Bayesian* with respect to the latent variables. In contrast to the models in the previous section, here it is necessary to define a *prior* on the latent variables $p(\boldsymbol{z})$. This can be an advantage, since it provides an opportunity to encode prior beliefs. It can also be a burden, since it also *requires* that we encode prior beliefs.

We can think of the observation model $p(\boldsymbol{y}|\boldsymbol{z})$ as the probabilistic version of a forward mapping. From Bayes' rule we get the corresponding, probabilistic backward mapping as:

$$p(\boldsymbol{z}|\boldsymbol{y}) = \frac{p(\boldsymbol{y}|\boldsymbol{z})p(\boldsymbol{z})}{p(\boldsymbol{y})} \tag{2.8}$$

This definition of a backward mapping is obviously more complicated than the simple linear, or non-linear, function, defined in the previous section. It is not uncommon to choose the prior, such that the resulting backward model has a convenient analytical form, rather than actually encoding a strong prior belief. The common claim justifying probabilistic over optimization-based models is that probabilities allow us to be explicit about uncertainties. However the actual benefit of generative models over simple optimization is not uncontroversial [LeCun and Huang, 2005], and in many applications that make use of generative models the actual importance of using probabilities is unknown. Another potential advantage of this

class of models, however, is that it allows us to take a fully Bayesian perspective and consider the model parameters themselves as random variables. Fully Bayesian modelling is beyond the scope of this thesis.

Typically, the model $p(\boldsymbol{y}, \boldsymbol{z})$ contains parameters W, that are optimized by maximizing the average log probability of the data: $\frac{1}{N}\sum_\alpha \log p(\boldsymbol{y}^\alpha)$. Similarly as in the previous section, bi-partite structure can be useful to simplify the models. In particular, we obtain a probabilistic version of PCA when using a linear dependency of $\boldsymbol{y}$ and a Gaussian prior on $\boldsymbol{z}$ plus Gaussian noise [Tipping and Bishop, 1999]. For more general non-linear, non-Gaussian models, estimating model parameters and evaluating the backward-mapping can be complicated or intractable. In that case the much simpler error minimization approach described above is often more convenient.

Note that, if the number of variables is large, specifying a model and doing computations in it can become difficult, because the number of possible joint instantiations of the variables is exponential in the number of variables. To overcome this issue we can make independence assumptions, so that the involved distributions decompose into products of distributions over only subsets of the variables [Jordan, 2008]. Probabilistic models that can be defined as a product of conditional probabilities can be represented conveniently as directed graphs. They are therefore known as directed graphical models. (There are also undirected models, which we will describe in the next section.)

A common model is the hidden Markov model, in which the dependency of the latent variable prior takes the form of a chain, so that we have $p(\boldsymbol{z}) = \prod_\alpha p(\boldsymbol{z}^\alpha|\boldsymbol{z}^{\alpha-1})$. Another common model, especially in vision problems, is the Markov random field (MRF), in which the components of the latent variables are arranged in a grid. MRFs are usually modelled as undirected models. We review these in Section 2.2.3.

Generative models vs. manifold learning

The graphical model approach to building complex systems is based on a slightly different modelling philosophy than manifold learning. (It is important to keep in mind, however, that many manifold learning methods, when cast in probabilistic terms, can be viewed as – extremely simple – types of graphical models themselves.) Graphical models make independence assumptions for tractability and thereby *fix* the dependency structure at the outset. Fixed, hand-coded features are often used as the data representation on which these models operate. Manifold learning, in contrast, does not making any assumptions on the dependency structure in the data, but rather tries to *discover* this structure through learning. It is the low dimensionality itself that is used to reduce computational complexity. Manifold learning methods often have a strong focus on *feature extraction*: Learning a good *representation* of the raw data is the main problem that manifold learning tries to solve. The weights in the bi-partite network (ref. Figure 2.2) and in similar models, when trained on images, for example, can be viewed as optimal adaptive *filters* [Bell and Sejnowski, 1997] [Olshausen and Field, 1996]. It is interesting to note in this context, that the *representation* of data can be argued to be the most important aspect of solutions to problems involving data. Finding the right representation is often at the heart of creative solutions to problems [Hofstadter, 1984], and with the right representation, the rest often follows automatically.

2.2.3 Restricted Boltzmann machines

A family of parametric methods that could be viewed as hybrids between optimization based approaches (Section 2.2.1) and generative models (Section 2.2.2) can be derived by considering *undirected graphical models*.

A probability distribution over a random vector can be modelled using an undirected graph as follows [Jordan, 2008]: We first define a graph whose nodes correspond to entries of the random vector. We then define a positive function $\Psi()$ on each clique (fully connected

subset of nodes) of the graph. Such a function can be viewed as a score that models the "goodness", or desirability, of joint instantiations of the nodes within a clique. Based on these score functions, we can define the probability over joint instantiation of *all* nodes in the graph by multiplying these functions together and normalizing. Because of positivity, we obtain a probability distribution over joint instantiations of all nodes (variables) in the graph. Normalizing simply amounts to dividing by the sum over all possible joint instantiations. Modelling a distribution this way amounts to making a set of independent judgements for each clique (each $\Psi()$). (The difference between directed and undirected models is that the first are normalized locally, so that the factors from which the joint distribution is composed are conditional probabilities.)

A common way of obtaining positive valued scores $\Psi()$ on the cliques from arbitrary, *ie.* not necessarily positive, score functions $S()$ is by exponentiating: $\Psi(\boldsymbol{y}) = \exp(S(\boldsymbol{y}))$. An arbitrary score function can be easier to model, so the "exp" is used to ensure positivity. The initial, non-positive score can be thought of as modelling a set of independent constraints on the cliques, which by exponentiating and normalizing gets turned into a set of independent probabilities.

In many applications, undirected models that have a bi-partite structure are of special interest. Bi-partite structure can, similarly as in non-probabilistic models, greatly simplify computations as we show below. To parameterize a bi-partite graphical model for *embedding*, we can use a score function of the form:

$$S(\boldsymbol{y}, \boldsymbol{z}) = \sum_{jk} W_{jk} y_j z_k \tag{2.9}$$

It is instructional to think of $\boldsymbol{y}$ and $\boldsymbol{z}$ as binary vectors for now. As mentioned above, given this score we can obtain the joint distribution as:

$$p(\boldsymbol{y}, \boldsymbol{z}) = \frac{1}{Z} \exp(S(\boldsymbol{y}, \boldsymbol{z})), \tag{2.10}$$

where

$$Z = \sum_{\boldsymbol{y}, \boldsymbol{z}} \exp(S(\boldsymbol{y}, \boldsymbol{z})) \tag{2.11}$$

The constant Z is usually known as *partition function*[3] with reference to statistical physics.

This model is known as Restricted Boltzmann machine (RBM). The restriction lies in the fact that, similarly as the non-probabilistic methods discussed above, there are no connections between hidden units or between observed units. The only connections that are present are between hidden and observed variables. So the model takes the same form as in figure 2.3. This bi-partite structure translates into convenient independence properties: From Eq. 2.10 it follows easily (by conditioning and then marginalizing) that the *conditional* distributions over latent variables, given observations, and over observations, given the latent variables, are products of element-wise independent distributions:

$$p(z_k = 1|\boldsymbol{y}) \quad = \quad \frac{1}{1 + \exp(-\sum_j W_{jk} y_j)} \tag{2.12}$$

for every latent variable z_k, and:

$$p(y_j = 1|\boldsymbol{z}) \quad = \quad \frac{1}{1 + \exp(-\sum_k W_{jk} z_k)} \tag{2.13}$$

for every observed variable y_j.

Eqs. 2.12 and 2.13 show that both mappings take a log-linear form, with linear parts that are *symmetric* versions of each other. The whole model therefore has the same structure like an auto-encoder (figure 2.4), with the differences that here, the linear mappings are transposes of each other, and all units are binary and stochastic. Similarly as in the models above, we need to add biases in practice to model affine, rather than linear, structure. (However, doing so does not affect the bi-partite structure.)

To train the model, we can proceed as with directed graphical models, and maximize the log-probability of the observed data, or equivalently minimize the negative log-probability, which gives us the objective function:

$$E^{\mathrm{RBM}} = -\sum_{\alpha} \log p(\boldsymbol{y}^{\alpha}) = -\sum_{\alpha} \log \sum_{\boldsymbol{h}} p(\boldsymbol{y}^{\alpha}, \boldsymbol{z}). \tag{2.14}$$

[3]For notational convenience and to comply with the historic conventions we will overload the symbol Z to denote both, the partition function, and later a random variable from which latent representatives can be drawn. It will be clear from the context which meaning is intended.

The gradient with respect to the parameters is given by:

$$\frac{\partial E^{\mathrm{RBM}}}{\partial W} = -\sum_{\alpha}\left[\sum_{\boldsymbol{z}} p(\boldsymbol{z}|\boldsymbol{y}^{\boldsymbol{\alpha}})\frac{\partial S(\boldsymbol{z},\boldsymbol{y}^{\boldsymbol{\alpha}})}{\partial W} - \sum_{\boldsymbol{z},\boldsymbol{y}} p(\boldsymbol{y},\boldsymbol{z})\frac{\partial S(\boldsymbol{z},\boldsymbol{y})}{\partial W}\right] \tag{2.15}$$

Derivatives of the score function 2.9 are simply pair-wise products:

$$\frac{\partial S(\boldsymbol{y},\boldsymbol{z})}{\partial W_{jk}} = y_j z_k. \tag{2.16}$$

By marginalizing Eq. 2.15 we can write the element-wise derivatives also as:

$$\frac{\partial E}{\partial W_{jk}} = \sum_{\alpha} y_j^{\alpha} z_j^{\alpha} - \sum_{y_j,z_k} p(y_j,z_k) y_j z_k, \tag{2.17}$$

where we define $z_j^{\alpha} := p(z_j = 1|\boldsymbol{y}^{\alpha})$. (Recall, that we use binary units for now.)

Both the objective function (Eq. 2.14) and the derivatives (Eqs. 2.15 and 2.17) are intractable, because they contain sums over joint instantiations of $\boldsymbol{y}$ and $\boldsymbol{h}$. However, because of the bi-partite structure, we can train these models similarly as the auto-encoder by performing restricted updates *locally*:

To obtain an efficient learning method, we can proceed as follows. First, note that we could approximate the intractable sums using samples from the joint distribution $p(\boldsymbol{y},\boldsymbol{z})$. In this model, we could perform block Gibbs sampling [Neal, 1993] for this purpose. Because of the factorization of $p(\boldsymbol{z}|\boldsymbol{y})$ and $p(\boldsymbol{y}|\boldsymbol{z})$, sampling from these conditionals can be performed component by component. When using binary units, for example, we just need to sample from scalar Bernoullis (Eqs. 2.12 and 2.13).

To draw samples from the joint $p(\boldsymbol{z},\boldsymbol{y})$ we could thus initialize randomly, and perform block Gibbs updates by repeatedly drawing samples from the conditionals $p(\boldsymbol{z}|\boldsymbol{y})$ and $p(\boldsymbol{y}|\boldsymbol{z})$. Finally we could approximate the sums in Eqs. 2.14 and 2.15 using these samples. While this procedure is possible in principle, sampling in high-dimensional spaces is very time-consuming and in practical problems it could take too long for the Gibbs sampler to converge.

However, note that a single iteration of this sampling procedure is strikingly similar to the procedure of computing reconstructions in an auto-encoder network with sigmoid units.

The only difference is that here, activities of the units are sampled instead of being fixed, and the weights in this model are symmetric (*ie.* $W = A^{\mathrm{T}}$ in the auto-encoder terminology, ref. Section 2.2.1). This similarity suggests using a locally constrained learning procedure, similarly as for the auto-encoder, instead of the global procedure based on proper sampling in the whole data-space. Contrastive divergence (CD) [Hinton, 2002] is a learning approach that turns this observation into practice: In order to approximate the intractable sum in the gradient, we initialize the observable units with the *training data* itself, and then use as samples simply the one-step-reconstructions. In other words, we replace the intractable gradient (Eqs. 2.15, 2.16) by

$$D_{jk}^{\mathrm{CD}} = \sum_{\alpha} y_j^{\alpha} z_j^{\alpha} - z_j^{\alpha,1} y_j^{\alpha,1} \tag{2.18}$$

where z_j^{α} is defined as above and $z_j^{\alpha,1}$ and $y_j^{\alpha,1}$ are the reconstructions computed using one iteration of Gibbs sampling. In contrast to using an autoencoder, computing gradients here is simpler than with back-propagation, and we obtain a probabilistic model. CD has been applied in [Welling et al., 2005], [Gehler et al., 2006] [He et al., 2004], for example.

Note, that the derivations above hold for binary $\boldsymbol{z}$ and $\boldsymbol{y}$. However, it is straightforward to extend this reasoning to any other exponential family distribution for either $\boldsymbol{z}$ or $\boldsymbol{y}$ [Welling et al., 2005]. We will use similar arguments as described in [Welling et al., 2005] in Section 5 to extend conditional variations of RBMs into the exponential family.

Also note that the manifold that we can model with binary hidden units is non-linear. An RBM with binary hidden units is in a sense the analog of an auto-encoder with sigmoid hidden units, because the hidden variable activations are computed using the same non-linearity (Eq. 2.12).

Deterministic manifold learning methods and RBMs trained using contrastive divergence are based on the same modelling philosophy: They assume that the data distribution is concentrated in a highly constrained region of the data space and try to build a valid model of the data *only in this region*. In contrast to proper probabilistic models trained using maximum

likelihood, these methods *do not* try to build a valid model of the data in regions that were not seen during training, *ie.* in regions where no training data resides. The parameters of these models do *define* a model globally (we could plug in test-cases from anywhere in the data-space and evaluate their probabilities according to Eq. 2.10, for example). But in areas other than near the training data the result would not be reasonable.

The motivation for this modelling approach is that we do not *expect* to ever see data from these regions at test-time. Trying to build a globally valid model in a very high-dimensional space would be wasteful and useless, if we will not encounter test-cases to which we could apply this model. More importantly, building a global model is often not even an option in practice. If the data-space is very high-dimensional, "covering" it with a correct model can be difficult, since proper sampling will take too long to be tractable. Restricting the model, such that integrations can be performed analytically is in many applications not an option, either, since it would be too restrictive to be useful.

This local modelling approach is also related to the framework of non-probabilistic, energy-based learning [LeCun and Huang, 2005], which also avoids the effort of building globally normalized, probabilistic model in cases where they are simply not needed. Learning in this framework amounts to "sculpting" an energy-landscape. From the perspective of this framework, CD does not attempt to sculpt the energy surface correctly in areas far from where the data resides.

2.3 Non-parametric methods

The parametric embedding methods that we discussed in the previous section have been useful in many problems, but they do have a few limitations. Parametric models lack flexibility, because they make the assumption that the manifold belongs to a specific parametric family, such as linear in the case of PCA. In many real world problems there is no reason, a priori, to make such an assumption.

Training a highly non-linear manifold can be time-consuming and difficult, especially when using a relatively flexible non-linear model, such as a multi-layer auto-encoder. Training parametric models is especially difficult when dealing with very high-dimensional data. Probabilistic latent variable models are even more problematic in terms of tractability. Therefore, there has been a strong bias towards linear models, especially among the fully probabilistic approaches, even though linearity does often not hold in real world problems, as we will discuss later.

Because of these and related issues, a new class of non-parametric models has emerged in the last few years. These models take a "dual" perspective, in which scalar-valued, pairwise similarities between all data-cases are the basis for computing low-dimensional embeddings. These methods can be thought of as non-linear generalizations of Multidimensional Scaling (MDS) (see [Mardia et al., 1980]), but many of these modern methods add more than just non-linearity to MDS as we shall show. Non-parametric models do not assume a fixed functional relationship between latent variables and data, so we can think of them as "universal approximators".

Modern embedding methods in particular side-step the issues associated with probabilistic latent variable models. While some of the methods do come with certain probabilistic interpretations (for example, [Lawrence, 2004] [Hinton and Roweis, 2003]), they are non-probabilistic in a fundamental sense: They do not compute distributions over latent representatives, but only point-estimates. That is, non-parametric methods are not Bayesian with respect to $\boldsymbol{Z}$. The fact that these methods do not encode uncertainties in their estimates of the latent data representation could be considered a disadvantage, but it is the basis for their computational feasibility and efficiency.

A drawback of many early non-parametric methods is that they do not provide estimates of the mappings $f(\boldsymbol{z})$ and $g(\boldsymbol{y})$. To solve the problem, several heuristics have been proposed (see *eg.* [Bengio et al., 2004] for backward-, and [Kwok and Tsang, 2003] for forward-mappings). Some recent non-parametric models do provide either of these mappings, as we show below. In

Section 3 we describe a non-parametric method that provides both the forward- and backward-mapping.

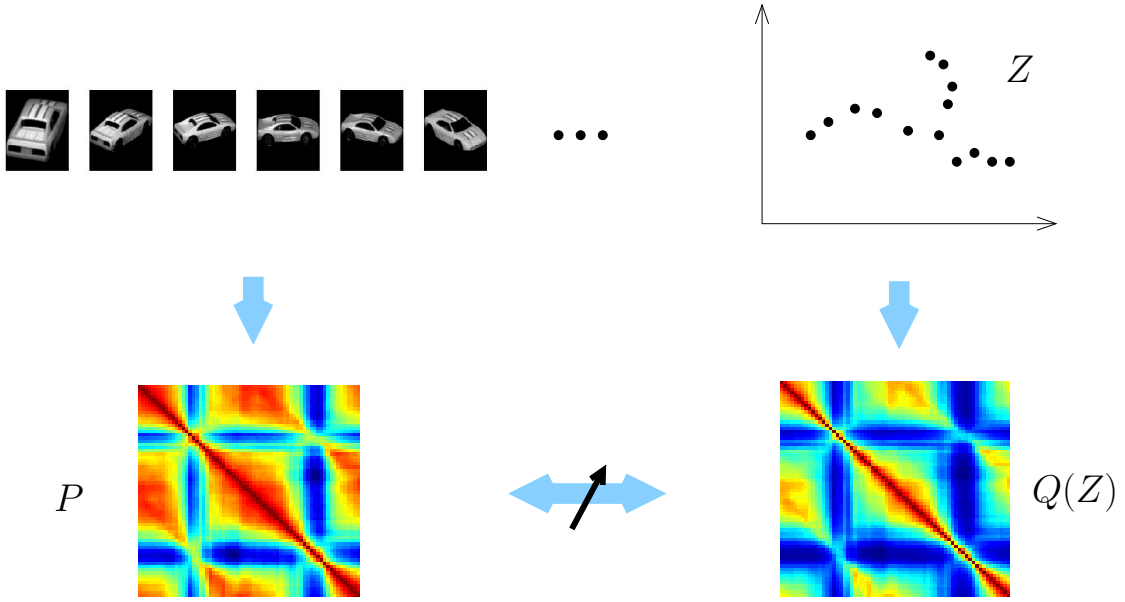

Figure 2.5: Illustration of non-parametric dimensionality reduction: We compute a similarity-matrix P from the input data-set (a set of car-images in the example), and a similarity matrix $Q(\boldsymbol{Z})$ as a function the of the latent data representatives $\boldsymbol{Z}$. In order to perform dimensionality reduction, we arrange the latent representatives, such that a measure of the mismatch between P and $Q(\boldsymbol{Z})$ is minimized. This leads to an optimization problem in the latent matrix $\boldsymbol{Z}$.

2.3.1 Optimizing mismatch

Practically all non-parametric embedding methods are based on the same principle, illustrated in figure 2.5: First, we construct an $N \times N$-matrix P that encodes all pairwise similarities between the observed data-points. We also define a corresponding matrix $Q(\boldsymbol{Z})$ that encodes

the similarities between the low-dimensional data representatives[4]. Letting this latent similarity matrix depend on the set $\boldsymbol{Z}$, allows us to perform dimensionality reduction simply by minimizing a measure of the *mismatch* between these two matrices wrt. $\boldsymbol{Z}$. We will denote the mismatch by $E(\boldsymbol{Z})$ in the following. In contrast to Section 2.2, here the objective function is only a function of $\boldsymbol{Z}$, since there are no parameters. The parameter matrix on the left of figure 2.2 simply gets dropped in non-parametric methods.

Most non-parametric embedding methods differ only in the definition of P, $Q(\boldsymbol{Z})$ and $E(\boldsymbol{Z})$. The motivations for the many possible ways to define the similarities and the mismatch differ greatly across methods and give rise to specific benefits and limitations.

One difference is with respect to optimization: If $E(\boldsymbol{Z})$ is a *quadratic* function of the latent variables, in other words, if we can write $E(\boldsymbol{Z}) = \mathrm{tr}(\boldsymbol{Z}M\boldsymbol{Z}^{\mathrm{T}})$, for some symmetric, positive definite matrix M, we can obtain a solution that can be found by an eigendecomposition by the Rayleigh-Ritz theorem, as mentioned above. However, it is necessary in that case to impose an orthogonality constraint on $\boldsymbol{Z}$. The dual PCA solution (Section 2.2.1) is an instantiation of this type of method. Many non-linear methods can be cast in this form, too (for example, Locally Linear Embedding [Roweis and Saul, 2000], ISOMAP [Tenenbaum et al., 2000], kernel PCA [Schölkopf et al., 1998]). We shall review some of these methods below.

If we cannot write $E(\boldsymbol{Z})$ as a quadratic form, or do not want to impose the orthogonality constraints, we can resort to iterative optimization. Methods that are based on iterative optimization include (we review these below, too): SNE [Hinton and Roweis, 2003], GPLVM [Lawrence, 2004], UKR [Meinicke et al., 2005] and others. Since the objective functions of these methods are not convex, we can only find local minima of their objective functions. However, the presence of local minima is usually not considered a major problem, because the local minima are often very good, and if not, there are methods for encouraging an optimizer

[4] P depends on $\boldsymbol{Y}$ and Q depends on $\boldsymbol{Z}$. We make the dependency of Q on $\boldsymbol{Z}$ explicit in the text, but not the dependency of P on $\boldsymbol{Y}$, because $\boldsymbol{Z}$ is a variable that we optimize, while $\boldsymbol{Y}$ is fixed and constant throughout the optimization.

to find good local minima. We briefly review a few common methods to provide an idea of the flavors and capabilities of the different approaches:

The **Sammon mapping** [Sammon, 1969] is probably the first non-linear embedding method and was introduced in the 1960's. It is based on a heuristic objective function for preserving point distances. It does not define a forward- or backward mapping.

Stochastic neighbor embedding (SNE) [Hinton and Roweis, 2003] defines each entry $P_{\alpha\beta}$ of P as the conditional probability that data-point α "picks" data-point β as its neighbor. It defines the matrix $Q(\boldsymbol{Z})$ analogously for the latent representatives. For training, SNE minimizes the sum of KL-divergences between the rows of P and the rows of $Q(\boldsymbol{Z})$ wrt. $\boldsymbol{Z}$ The probabilities are computed by centering Gaussian kernels on the data-points. The method does not provide forward- or backward mappings, but it has been fairly successful, because unlike many other methods, its intuitive probabilistic interpretation suggests a wealth of extensions (see, for example, [Goldberger et al., 2005], [Cook et al., 2007]).

Locally linear embedding [Roweis and Saul, 2000] computes weights to linearly reconstruct observed data-points from neighboring points. It then tries to arrange the latent representatives, such that they can be re-constructed from their neighbors with the same weights, which can be achieved by solving an eigen-problem. The method does not provide a forward- or backward-mapping.

Kernel PCA [Schölkopf et al., 1998] makes use of the duality property of the PCA eigen-problem (Section 2.2.1): Recall, that to perform PCA, we can decompose the inner product matrix $\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Y}$ to get the latent representatives $\boldsymbol{Z}$. By Mercer's theorem (see [Schölkopf and Smola, 2001]) a positive definite kernel function $k(\boldsymbol{y}^\alpha, \boldsymbol{y}^\beta)$ can be interpreted as computing an inner product between feature vectors $\phi(\boldsymbol{y}^\alpha)$ and $\phi(\boldsymbol{y}^\beta)$, that can be non-linearly related to $\boldsymbol{y}^\alpha$ and $\boldsymbol{y}^\beta$. Therefore, replacing $\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Y}$ by a kernel matrix, amounts to performing a non-linear transformation to the data and subsequently performing PCA in the feature space. The transformation is only implicit, that is we do not actually compute, or parameterize, it. A drawback of this formulation is that we only obtain the low-dimensional

representatives $\boldsymbol{Z}$ and the backward-mapping but not the forward-mapping. Below we will describe an "inverse" view of kernel PCA that provides the forward mapping instead.

Other non-parametric methods include Laplacian eigenmaps [Belkin and Niyogi, 2003], ISOMAP [Tenenbaum et al., 2000] and semi-definite embedding [Weinberger and Saul, 2006]. Recently, several non-parametric methods have been introduced from the perspective of **non-parametric regression**. In the next section we describe a non-parametric framework for PCA, and show that those methods are special cases within this framework.

2.3.2 Two non-parametric views of PCA

It is possible to obtain two non-parametric views of PCA by replacing the linear PCA-mappings with non-parametric functions. Replacing either the forward- or backward-mapping leads to two different types of non-parametric embedding method, each with specific advantages and limitations. Several recently proposed models are instantiations of one of these two views.

PCA as a non-parametric forward mapping

First, let us consider as the forward-mapping $f(\boldsymbol{z})$ the class of non-parametric functions of the following form: $f(\boldsymbol{z}) = \sum_\beta b(\boldsymbol{z}, \boldsymbol{z}^\beta)\boldsymbol{y}^\beta$, where $b(\boldsymbol{z}, \boldsymbol{z}^\beta)$ are some (arbitrary) basis functions defined in the latent space. Functions that define the output as a linear combination of training cases, are generally referred to as *linear smoothers*. They are common as supervised non-parametric regression models.

Using this form as the forward mapping for dimensionality reduction means that the mapping depends on the latent representatives themselves. So at first sight it seems that this definition makes sense only, if we had the representatives already. However, note that, given this forward-mapping, training can be achieved similarly as in a parametric model, by

minimizing the reconstruction error:

$$E(\boldsymbol{Z}) = \sum_{\alpha} \|\boldsymbol{y}^{\alpha} - f(\boldsymbol{z})\|^2 = \|\boldsymbol{y}^{\alpha} - \sum_{\beta} b(\boldsymbol{z}^{\alpha}, \boldsymbol{z}^{\beta})\boldsymbol{y}^{\beta}\|^2 \tag{2.19}$$

Since the forward-mapping is a *function of the latent representatives themselves*, minimizing this objective gives rise to both the latent representatives and the forward mapping simultaneously.

Several special cases of this framework were recently derived independently from each other from probabilistic viewpoints. [Meinicke et al., 2005], [Memisevic, 2003] used the Nadaraya-Watson kernel regression model [Bishop, 1995] as the non-parametric forward function. The model is known as unsupervised kernel regression (UKR). Later, [Lawrence, 2004] used a Gaussian process regression model as the forward function and called the model a "Gaussian process latent variable model" (GPLVM). It is important to note that these special cases are direct generalizations of PCA. To see this, consider the use of *linear* basis functions $b(\boldsymbol{z}, \boldsymbol{z}^{\beta}) = \boldsymbol{z}^{\mathrm{T}}\boldsymbol{z}^{\beta}$. Under this choice, we recover PCA by re-writing the objective as:

$$\begin{aligned} E(\boldsymbol{Z}) &= \sum_{\alpha} \|\boldsymbol{y}^{\alpha} - \sum_{\beta} \boldsymbol{y}^{\beta}\boldsymbol{z}^{\alpha\mathrm{T}}\boldsymbol{z}^{\beta}\|^2 && (2.20)\\ &= \|\boldsymbol{Y}(I - \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z})\|^2 && (2.21)\\ &= \mathrm{tr}(\boldsymbol{Y}(I - \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z})^2\boldsymbol{Y}^{\mathrm{T}}) && (2.22)\\ &= -\mathrm{tr}(\boldsymbol{Z}\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Y}\boldsymbol{Z}^{\mathrm{T}}) + \Omega, && (2.23) \end{aligned}$$

where Ω denotes terms that do not depend on $\boldsymbol{Z}$. This problem takes exactly the form of the dual PCA eigen-problem (Section 2.2.1) if we impose the constraint $\mathrm{tr}(\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z}) = I$. Using non-linear basis functions amounts obtaining non-linear embeddings, at the cost of giving up on the convenience of a spectral solution. But we do retain the advantage of PCA that we learn a forward mapping along with the latent representatives. We therefore do not have a pre-image problem [Kwok and Tsang, 2003] in this type of model.

PCA as a non-parametric backward mapping

Let us now consider a non-parametric model for the *backward-mapping* $g(\boldsymbol{y})$. Similarly as above we can use the form $g(\boldsymbol{y}) = \sum_\beta b(\boldsymbol{y}, \boldsymbol{y}^\alpha)\boldsymbol{z}^\beta$, where $b(\boldsymbol{y}, \boldsymbol{y}^\alpha)$ are now basis functions defined in the data-space. Learning the backward-mapping, rather than the forward-mapping, suggests optimizing the *latent space reconstruction error*, which results in the objective function:

$$
\begin{aligned}
E(\boldsymbol{Z}) &= \sum_\alpha \|\boldsymbol{z}^\alpha - g(\boldsymbol{y}^\alpha)\|^2 && (2.24)\\
&= \sum_\alpha \|\boldsymbol{z}^\alpha - \sum_\beta b(\boldsymbol{y}^\alpha, \boldsymbol{y}^\beta)\boldsymbol{z}^\beta\|^2 && (2.25)\\
&= \|\boldsymbol{Z}(I - B(\boldsymbol{Y}))\|^2 && (2.26)\\
&= \mathrm{tr}(\boldsymbol{Z}(I - B(\boldsymbol{Y}))(I - B(\boldsymbol{Y}))^{\mathrm{T}}\boldsymbol{Z}^{\mathrm{T}}), && (2.27)
\end{aligned}
$$

where we use the matrix $(B(\boldsymbol{Y}))_{\alpha\beta} = b(\boldsymbol{y}^\alpha, \boldsymbol{y}^\beta)$ of basis function evaluations. We obtain an eigen-problem again. If $B(\boldsymbol{Y})$ is symmetric, we have $(I - B(\boldsymbol{Y}))(I - B(\boldsymbol{Y}))^{\mathrm{T}} = (I - B(\boldsymbol{Y}))^2$, who have the same eigenvectors. Instead of keeping the leading eigenvectors of $(I - B(\boldsymbol{Y}))$ we can equivalently keep the eigenvectors corresponding to the *smallest* eigenvalues of $B(\boldsymbol{Y})$. So, if we use linear basis functions, we again recover the dual PCA eigen-problem. However, in contrast to above, if we use non-linear basis functions in order to obtain non-linear embeddings, we still retain the spectral solution. In this approach we implicitly learn the backward-mapping in addition to the latent representatives.

A special case of this approach, where the basis functions are derived from kernel regression, was proposed initially by [Meinicke, 2002] and is discussed in detail in [Memisevic, 2003]. Another special case is kernel PCA [Schölkopf et al., 1998], which we obtain by using Mercer kernels as the basis functions.

Deriving kernel PCA as a special case of a non-parametric backward mapping shows why the backward-mapping, but not the forward mapping, comes for free in kernel PCA. While kernel PCA relies on symmetric Mercer kernels as the basis functions, the view from non-parametric regression shows that symmetry is not actually required, and we can use arbitrary

basis functions, such as the probabilistic ones described in [Memisevic, 2003]. Many further possibilities exist. We can, for example, use a Gaussian process mean function as the backward mapping to obtain spectral Gaussian process embeddings, which are the spectral counterparts of the model described by [Lawrence, 2004].

2.4 Categorization of embedding methods

It is clear from the previous sections that many data embedding methods have been proposed in the past. They can be categorized along several dimensions. The most important ones are:

Parametric vs. non-parametric

Parametric models (Section 2.2) operate in the original data-space, non-parametric models operate in the "dual" representation of pairwise similarities. Non-parametric models are typically non-linear. PCA is the link between parametric and non-parametric models, as it can be formulated in both ways.

Provide latent representatives; provide f; provide g

As mentioned above, the embedding problem is three-fold: Learn latent representatives $\boldsymbol{Z}$; learn a forward mapping $f(\boldsymbol{z})$; learn a backward mapping $g(\boldsymbol{y})$. Methods differ with respect to which of the three problems they solve. Parametric approaches usually model at least one of the involved mappings, non-parametric approaches models these only when defined as regression models (Section 2.3.2).

Linear vs non-linear

The assumption of a linear manifold is often too restrictive in practical applications. Non-parametric models are naturally non-linear, parametric models require the presence of non-linear mappings, such as neural networks, to be non-linear.

Probabilistic vs. non-probabilistic

Some embedding methods provide distributions over the latent representation to encode uncertainties. That is, they are *Bayesian* w.r.t the latent variables. Non-parametric methods are not probabilistic in this sense. Many of them do come with certain probabilistic interpretations [Hinton and Roweis, 2003], [Lawrence, 2004], [Meinicke et al., 2005], but they do not represent uncertainties in the latent representations.

Optimization

An important difference between parametric models and non-parametric models is that the first can easily be trained online, the latter require the whole data-set to be around during training. The computational complexity for training usually scales linearly with the number of cases for parametric models and quadratically (even cubically for gplvm [Lawrence, 2004]) for non-parametric models. For these reasons, parametric models can be trained on much larger data-sets than non-parametric models.

The training complexity of parametric methods and most non-parametric methods (in particular those that come with a forward-mapping) scales linearly with the number of dimensions of the output space. This can be problematic in very high-dimensional spaces. In section 3 we will introduce a non-parametric model whose optimization complexity does not scale with the data dimensionality, but which nevertheless defines a forward and backward mapping.

Some embedding methods can be trained by optimizing a quadratic or convex optimization problem, others require non-linear, non-convex optimization. To obtain a convenient optimization problem, such as an eigen-problem, we usually need to impose a constraint on the latent variables, such as uncorrelatedness. [Memisevic and Hinton, 2005] suggest a way of combining the benefits of spectral and general, iterative methods by modulating the gradient of a non-quadratic objective function with information extracted from the eigenvalues of an affinity matrix.

2.5 Side-information

Most dimensionality reduction methods are unsupervised, so there is no way of guiding the methods towards modes of variability that are of particular interest to the user. There is also no way of providing hints, when the true underlying factors are too subtle to be discovered by generic criteria such as the maximization of modeled variance in PCA, or the preservation of local geometry in LLE, for example. To overcome these issues, several approaches have been suggested to include side-information in order to improve the embeddings.

As an example of the issue consider images of faces. Nonlinear methods have been shown to find embeddings that nicely reflect the variability in the data caused by variation in face identity, pose, position, or lighting effects. However, it is not possible to tell these methods to extract a particular single factor for the purpose of, say intelligent image manipulation or pose identification, because the extracted factors are intermingled and may be represented simultaneously across all latent space dimensions.

Recently there have been several attempts to provide *side-information* in the hope of alleviating some of these difficulties. The idea is to allow the user to provide more information than just the raw data points, or a single set of pairwise similarities between data points. These approaches turn the unsupervised problem of dimensionality reduction into a semi-unsupervised one. [Cohn, 2003], for example, describe a way to inform a PCA model by encouraging it to preserve a user-defined grouping structure; [Tenenbaum and Freeman, 2000] consider the problem of extracting exactly two different kinds of factor, which they denote "style" and "content", by using bilinear models. This idea has been extended to more general multilinear models by [Vasilescu and Terzopoulos, 2002]. [Xing et al., 2003] and [Tijl De Bie and Cristianini, 2003] took a quite different approach to informing a model. They suggest pre-processing the input data by learning a metric in input space that makes the data respect user defined grouping constraints. In all these methods, side-information is encoded in the form of *equivalence constraints*. That is, a user needs to define a grouping structure for the input data by informing the model which data-points belong together.

In general, the problem of informing latent variable models with some kind of supervision signal is quite new, and it is the subject of current research. No framework or common viewpoint has emerged yet. It is worth noting, that non-parametric models provide a specifically convenient way to encode latent-variable constraints. Since the latent data matrix is modelled directly in these models, it is convenient to simply add constraints as a penalty term to the objective function of these models. See, for example, [Memisevic, 2003] [Urtasun et al., 2006] for applications of this idea. Note, that this possibility disappears, if we use spectral methods, which dictate the latent variable constraints in order to obtain an eigen-decomposition as the solution.

2.6 Structured discriminative learning

An important application of dimensionality reduction methods is in prediction problems, where the outputs are high-dimensional vectors. In this section we briefly review this type of problem and the solutions that have been proposed. We will discuss the problem in more detail in Section 6.

2.6.1 Predicting high-dimensional outputs

Recall that one motivation for data embedding is solving an *inverse problem*, in order to model perceptual capabilities, for example, (see Section 2.1). We use latent variables to model the true, underlying degrees of freedom of some data to capture the "meaning" inherent in the set of observations, and to get an understanding of some observed scene. From a probabilistic perspective (Section 2.2.2) solving the inverse problem at test time then amounts to invoking Bayes' rule (Eq. 2.8) to extract the meaning $p(\boldsymbol{z}|\boldsymbol{y})$ from the data $\boldsymbol{y}$.

In practice, it is often the case that the desired "percept" $\boldsymbol{z}$ can be made available for training a system by hand-labelling the data before training ([He et al., 2004], [Lafferty et al., 2001], [Kumar and Hebert, 2004]). In those cases learning turns into a su-

pervised problem: The training data consists of (input, output)-pairs. For example, in [He et al., 2004] the inputs are images, and the pixel labelings. In other applications, the inputs could be sequences of words, and the outputs grammatical categories for the words.

Unlike a traditional supervised learning problem, this problem contains outputs that can be high-dimensional vectors, possibly with varying sizes. However, most typical supervised learning methods have been developed for scalar outputs only, and adapting the developed machinery to vector-valued, and in particular high-dimensional, outputs is not trivial. We can solve the issue naively by building a set of independent predictors (one for each output-dimension), however since this solution fails to capture correlations in the outputs, it usually performs extremely poorly in practice. Modelling all possible correlations among the output-variables on the other hand would amount to considering all possible *joint* instantiations of the components of $\boldsymbol{y}$, which leads to a combinatorial explosion, and makes standard supervised models intractable [Memisevic, 2006b].

It has therefore been common to use unsupervised methods even for these essentially supervised problems in order to overcome these issues. In particular, using graphical models to model the "percept" $\boldsymbol{z}$ allows us to capture correlations tractably and makes it possible to perform inference using belief propagation or variational methods. In a generative model, where the correct latent variable instantiations have been provided for a training set through labelling, $\boldsymbol{z}$ is no longer unobserved. For training in this supervised setting, therefore, instead of marginalizing (Eq. 2.6), we can maximize the *joint likelihood* $p(\boldsymbol{z}, \boldsymbol{y})$, using a training set $\{(\boldsymbol{z}^\alpha, \boldsymbol{y}^\alpha)\}$. At test time, Bayes' rule, can then be used to perform inference.

2.6.2 Structure prediction using conditional models

Recently, an alternative modelling approach has become popular [LeCun et al., 1998], [LeCun and Huang, 2005], [Lafferty et al., 2001], [Kumar and Hebert, 2004], (for an overview see, for example, [Memisevic, 2006b]), which retains the "learning from example" philosophy from supervised learning, but solves the combinatorial problem using techniques

that have been popular in unsupervised learning. The idea is to use a graphical model on the outputs, and to make the model parameters *functions* of the inputs. Inference in these models is a simple application of the trained forward model. Since the outputs are modelled using a graphical model, inference is performed using dynamic programming or belief propagation. However, in contrast to unsupervised methods, it does not involve the inversion of a generative forward-mapping using Bayes' rule. [LeCun and Huang, 2005] discuss a generalization of this idea to non-probabilistic loss-functions and coined the term "end-to-end" learning for these types of model. We will describe these models in more detail is Section 6.

2.6.3 Structure prediction with flexible correlations

Most structure prediction models operate, as is typical for graphical models (see Section 2.2.2), on fixed, pre-defined feature functions, and assume a fixed independence structure for the components of the outputs $\boldsymbol{y}$.

To address the drawbacks of fixed feature functions, some approaches were suggested recently, that try to *model* the correlational structure of the outputs in structured prediction tasks [He et al., 2004] [Stewart, 2005]. The idea of these approaches is to build a latent variable model on the high-dimensional outputs. In Section 5 we describe an approach for learning data transformations by conditioning a latent variable model on input data. Supervised learning of structured predictions can also be viewed as an application of that framework. In section 6 we extend the model to a more general non-probabilistic framework, in which manifold learning can be viewed as a way to learn the correlational structure in the outputs of a supervised learning problem.

Chapter 3

Kernel information embeddings

In this chapter we derive a probabilistic framework for dimensionality reduction based on kernel density estimation. The framework combines the benefits of generative models with the geometric view-point of non-parametric methods. We show how within this framework we can compute non-linear embeddings that come with both the forward- and the backward-mapping and we show how we can obtain informed embeddings using conditioning.

3.1 An information theoretic framework for dimensionality reduction

Following the notation introduced in Section 2, we define dimensionality reduction as the task of finding a set of low-dimensional representatives $\{\boldsymbol{z}^\alpha\}_{\alpha=1\ldots N}$ for a set of high-dimensional input data-points $\{\boldsymbol{y}^\alpha\}_{\alpha=1\ldots N}$. To solve this problem non-parametrically, we can define an objective function that captures the preservation of inter-point similarities and optimize this function wrt. the whole set of latent representatives $\boldsymbol{z}_i$ themselves, as discussed is Section 2.3. Recall, that if we stack the $\boldsymbol{z}^\alpha$ column-wise in a matrix, we obtain an optimization problem whose domain is the set of real matrices of size $q \times N$. For convenience, we will also think of the domain as $\mathbb{R}^{Nq}$, which we obtain by "vectorizing" this matrix.

Section 2.3 discussed how stating the problem in terms of the latent representatives themselves allows us to replace the traditional probabilistic view of embedding with a geometric one: Non-parametric methods, such as LLE or kernel PCA, measure inter-point similarities in the data-space to get an idea of the geometric configuration of the whole point-set. Arranging the low-dimensional representatives, such that they mimic this configuration, yields a lower-dimensional representation that captures the similarity structure in the original data-set, and thereby its underlying "structure".

Arguing in this "dual" domain of points and their similarities, rather than in the original domain of the random vectors themselves, can have several advantages, as discussed in detail in the previous section. But it can also have some disadvantages, mainly because geometric reasoning usually does away with probabilities. Probabilities can at times be useful as a measure of confidence that a model has in a certain solution, and *conditioning* in the probabilistic sense can be a natural operation for including side-information as we describe below.

A related issue is that the mappings $f(\boldsymbol{z})$ and $g(\boldsymbol{y})$ often fall out naturally when adopting a probabilistic viewpoint. A full model of the joint distribution $p(\boldsymbol{z}, \boldsymbol{y})$ contains all the information about the latent and observed variables and their dependencies. In other words, a model of the joint distribution contains all the information we can hope to have. From the joint it is usually straightforward to derive the (probabilistic) mappings (as in 2.8, for example).

In this chapter we describe a framework that allows us to combine the probabilistic view of generative models with the geometric reasoning of modern embedding methods. The tool that we use for this purpose is the well-known *kernel density estimate* (KDE): Given a random sample of points $\boldsymbol{y}^\alpha,\ \alpha = 1, \ldots, N$, we can obtain an estimate of the density from which the data was drawn by placing a positive, zero-mean kernel function on each data-point. We can define the estimate as the sum of the kernel function evaluations [Scott, 1992]:

$$\hat{p}(\boldsymbol{y}) = \frac{1}{N} \sum_{\alpha} \bar{k}(\boldsymbol{y}, \boldsymbol{y}^\alpha). \tag{3.1}$$

An advantage of this estimate is that we do not need to assume the distribution belongs to any

particular parametric family. The only assumption that we make on the density is smoothness. For $\hat{p}(\boldsymbol{y})$ to be a valid density estimate, the kernel functions $\bar{k}(\boldsymbol{y}, \cdot)$ need to be positive and integrate to one. A common way to derive a normalized kernel function $\bar{k}(\boldsymbol{y}, \boldsymbol{y}^\alpha)$ is by starting with some unnormalized positive kernel $k(\boldsymbol{y}, \boldsymbol{y}^\alpha)$, and defining $\bar{k}(\boldsymbol{y}, \boldsymbol{y}^\alpha) = \frac{1}{\mathcal{C}} k(\boldsymbol{y}, \boldsymbol{y}^\alpha)$ where $\mathcal{C} = \int k(\boldsymbol{y}, \boldsymbol{y}^\alpha) \, \mathrm{d}\boldsymbol{y}$.

To perform non-parametric embedding, we obtain an estimate of the distribution over the *joint* data/embedding space. As mentioned above, this estimate contains all the information that we can wish to have about the observable and latent variables and their relations. The kernel density estimate will allow us to cast the probabilistic approach in geometric terms similarly as in common non-parametric embedding methods. In other words, we obtain the advantages of non-parametric methods, while retaining the convenient properties of many parametric models.

3.1.1 Kernel information embedding

Let us assume that the observed data is drawn iid. from a vector-valued random variable Y. Our goal is to find a lower-dimensional variable Z that captures most of the variability in Y (*ie.* , the "signal").

Assume we had already found a such a variable Z. Then an intuitively appealing criterion for measuring how well Z captures the information present in the data Y would be the *mutual information* $I(Y; Z)$ between Y and Z. Mutual information (MI) can be defined as

$$I(Y; Z) = \int p(\boldsymbol{y}, \boldsymbol{z}) \log \frac{p(\boldsymbol{y}, \boldsymbol{z})}{p(\boldsymbol{y}) p(\boldsymbol{z})} \, \mathrm{d}\boldsymbol{y} \, \mathrm{d}\boldsymbol{z} \tag{3.2}$$

with $p(\boldsymbol{y}), p(\boldsymbol{z})$ the data- and latent space-densities, respectively (see, for example, [Cover and Thomas, 1990]). MI is equal to the KL-divergence between the actual joint distribution $p(\boldsymbol{y}, \boldsymbol{z})$ and a *factorized* joint distribution $p_{\mathrm{fact}}(\boldsymbol{y}, \boldsymbol{z}) = p(\boldsymbol{y}) p(\boldsymbol{z})$. Since the factorized distribution corresponds to a situation where $\boldsymbol{z}$ and $\boldsymbol{y}$ are statistical independent, MI can be viewed as a measure of the dependence between Y and Z.

Intuitively, MI (which is symmetric in Y and Z) answers the question: "What, on average, can the latent representation tell us about the data, and vice versa?" In other words, MI quantifies how much information would be preserved if we would use the low-dimensional data representatives as surrogates for the original data. Maximizing it amounts to obtaining latent variables that contain as much information of the original data as possible. A simple calculation shows that we can express MI in terms of entropies as [Cover and Thomas, 1990]

$$I(Y;Z) = H(Y) + H(Z) - H(Y,Z), \tag{3.3}$$

where $H(\cdot)$ denotes (differential) Shannon entropy.

In practice, MI and other entropy based quantities are not often used for embedding, because they are hard to evaluate for all but very specific distributions, such as the Gaussian. (It is interesting to note, however, that PCA can be viewed as maximizing mutual information between latent variables and observed data under the restriction of Gaussian distributions and linear mappings [Baldi and Hornik, 1995].)

Instead of trying to evaluate the involved high-dimensional integrals directly, we use an estimate of the entropies, based on a kernel estimate of the underlying densities: Using some spherical kernel functions $\bar{k}(\boldsymbol{z}^\alpha, \boldsymbol{z}^\beta)$ and $\bar{k}(\boldsymbol{y}^\alpha, \boldsymbol{y}^\beta)$, we can define the kernel density estimate of the *joint* density over the input and latent space as suggested above as:

$$\hat{p}(Y = \boldsymbol{y}, Z = \boldsymbol{z}) = \frac{1}{N} \sum_\alpha \bar{k}(\boldsymbol{z}, \boldsymbol{z}^\alpha) \bar{k}(\boldsymbol{y}, \boldsymbol{y}^\alpha) \tag{3.4}$$

Under this estimate for the joint, the marginals are given by

$$\hat{p}(Z = \boldsymbol{z}) = \frac{1}{N} \sum_\alpha \bar{k}(\boldsymbol{z}, \boldsymbol{z}^\alpha) = \frac{1}{N\mathcal{C}_Z} \sum_\alpha k(\boldsymbol{z}, \boldsymbol{z}^\alpha), \tag{3.5}$$

$$\hat{p}(Y = \boldsymbol{y}) = \frac{1}{N} \sum_\alpha \bar{k}(\boldsymbol{y}, \boldsymbol{y}^\alpha) = \frac{1}{N\mathcal{C}_Y} \sum_\alpha k(\boldsymbol{y}, \boldsymbol{y}^\alpha), \tag{3.6}$$

where $\mathcal{C}_Y$ and $\mathcal{C}_Z$ denote the observable space and latent space normalizing constants, re-

spectively. Now, using these density estimates we can estimate the *entropy* $H(Z)$ as follows:

$$\begin{aligned} H(Z) &= -\int p(\boldsymbol{z}) \log p(\boldsymbol{z}) \, \mathrm{d}\boldsymbol{z} && (3.7) \\ &\approx -\frac{1}{N} \sum_{\alpha} \log p(\boldsymbol{z}^{\alpha}) && (3.8) \\ &\approx -\frac{1}{N} \sum_{\alpha} \log \sum_{\beta} k(\boldsymbol{z}^{\alpha}, \boldsymbol{z}^{\beta}) + \log(N\mathcal{C}_Z) && (3.9) \\ &=: \hat{H}(\boldsymbol{Z}) && (3.10) \end{aligned}$$

We are making two approximations: First, using the law of large numbers we approximate the entropy by the negative log-likelihood of a sample (Eq. 3.8). Secondly, we plug in our kernel estimate $\hat{p}(\boldsymbol{z})$ for the true probability density function $p(\boldsymbol{z})$ (eq. 3.9). Note, that we use bold font for the argument to $\hat{H}()$ to stress the dependency on the latent matrix $\boldsymbol{Z}$.

Analogously, we can obtain estimates $\hat{H}(\boldsymbol{Y})$ and $\hat{H}(\boldsymbol{Y}, \boldsymbol{Z})$ for the entropy in the data-space and the joint entropy of the latent/observable space, respectively. We have:

$$\hat{H}(\boldsymbol{Y}) = -\frac{1}{N} \sum_{\alpha} \log \sum_{j} k(\boldsymbol{y}^{\alpha}, \boldsymbol{y}^{\alpha}) + \log(N\mathcal{C}_Y), \tag{3.11}$$

and

$$\hat{H}(\boldsymbol{Y}, \boldsymbol{Z}) = -\frac{1}{N} \sum_{\alpha} \log \sum_{\beta} k(\boldsymbol{y}^{\alpha}, \boldsymbol{y}^{\beta}) k(\boldsymbol{z}^{\alpha}, \boldsymbol{z}^{\beta}) + \log(N\mathcal{C}_Y\mathcal{C}_Z). \tag{3.12}$$

Now, plugging the entropy estimates into equation 3.3, we obtain an estimate of the *mutual information* as follows:

$$\begin{aligned} \hat{I}(\boldsymbol{Y}, \boldsymbol{Z}) &= \hat{H}(Y) + \hat{H}(Z) - \hat{H}(Y, Z) && (3.13) \\ &= \Omega_0 - \frac{1}{N} \sum_{\alpha} \log \sum_{\beta} k(\boldsymbol{z}^{\alpha}, \boldsymbol{z}^{\beta}) + \frac{1}{N} \sum_{\alpha} \log \sum_{\beta} k(\boldsymbol{z}^{\alpha}, \boldsymbol{z}^{\beta}) k(\boldsymbol{y}^{\alpha}, \boldsymbol{y}^{\beta}) && (3.14) \end{aligned}$$

We absorb all terms that do not depend on latent space elements (including possible normalization constants for the kernels) for convenience into the constant Ω_0. The decomposition of MI into entropy terms (Eq. 3.3) is convenient, since it allows us to assemble the objective from simpler terms. This decomposition will be convenient also when deriving related quantities, and it simplifies the computation of gradients as we shall show.

Learning

Until now we have made the tacit assumption, that we already have the latent representatives $\boldsymbol{Z}$, which is not the case, of course. However, the MI estimate is a function of the set of latent data representatives, so in order to perform dimensionality reduction, we can maximize it with respect to $\boldsymbol{Z}$ to learn the latent representation. Any gradient-based optimization method can be used for this purpose. Since the objective function decouples into the sum of two terms, so does the gradient.The derivative of the first term of the objective function wrt. to a single latent space element $\boldsymbol{z}^\alpha$ is given by:

$$\frac{\partial \hat{H}(\boldsymbol{Z})}{\partial \boldsymbol{z}^\alpha} = -\frac{1}{N}\sum_\beta \left(\kappa_Z^\alpha + \kappa_Z^\beta\right)\frac{\partial k(\boldsymbol{z}^\alpha, \boldsymbol{z}^\beta)}{\partial \boldsymbol{z}^\alpha} \tag{3.15}$$

where we abbreviate

$$\kappa_Z^\alpha = \frac{1}{\sum_\beta k(\boldsymbol{z}^\alpha, \boldsymbol{z}^\beta)}.$$

The gradient for the second term is:

$$\frac{\partial \hat{H}(\boldsymbol{Y}, \boldsymbol{Z})}{\partial z^\alpha} = -\frac{1}{N}\sum_\beta \left(\kappa_{YZ}^\alpha + \kappa_{YZ}^\beta\right) k(\boldsymbol{y}^\alpha, \boldsymbol{y}^\beta)\frac{\partial k(\boldsymbol{z}^\alpha, \boldsymbol{z}^\beta)}{\partial \boldsymbol{z}^\alpha} \tag{3.16}$$

with

$$\kappa_{YZ}^\alpha = \frac{1}{\sum_\beta k(\boldsymbol{y}^\alpha, \boldsymbol{y}^\beta)k(\boldsymbol{z}^\alpha, \boldsymbol{z}^\beta)}.$$

In the following we will use the Gaussian kernel, which takes the form

$$k(\boldsymbol{z}^\alpha, \boldsymbol{z}^\beta) = \exp(-\frac{1}{h_Z}\|\boldsymbol{z}^\alpha - \boldsymbol{z}^\beta\|^2). \tag{3.17}$$

For this kernel we have:

$$\frac{\partial k(\boldsymbol{z}^\alpha, \boldsymbol{z}^\beta)}{\partial \boldsymbol{z}^\alpha} = -\frac{2}{h_Z}k(\boldsymbol{z}^\alpha, \boldsymbol{z}^\beta)(\boldsymbol{z}^\alpha - \boldsymbol{z}^\beta). \tag{3.18}$$

In the observable space, we also use Gaussian kernels, with a corresponding observable bandwidth parameter h_Y:

$$k(\boldsymbol{y}^\alpha, \boldsymbol{y}^\beta) = \exp(-\frac{1}{h_Y}\|\boldsymbol{y}^\alpha - \boldsymbol{y}^\beta\|^2). \tag{3.19}$$

Training the model amounts to arranging the representatives in the latent space to maximize the MI estimate. Since the elements are free to move, any change in the latent space kernel width can always be compensated by rescaling the elements. The choice of the latent space bandwidth is therefore arbitrary, and we fix $h_Z = 1.0$ in the following. The data-space bandwidth is *not* arbitrary and has to be set by hand, cross-validation, or some heuristics. Fortunately, KIE is relatively robust with respect to the choice of bandwidth, as we discuss below. However, if needed, we can easily get a good estimate of the optimal data-space bandwidth by using a leave-one-out estimate of the data-log-likelihood [Härdle, 1989]. The leave-one-out estimate can be defined as

$$L^{\mathrm{loo}}(\boldsymbol{Y}) = \frac{1}{N}\sum_{\alpha} \log \sum_{\beta \neq \alpha} k(\boldsymbol{y}^\alpha, \boldsymbol{y}^\beta) - \log(N\mathcal{C}_Y) \tag{3.20}$$

Optimizing it with respect to the kernel bandwidth is possible with a one-dimensional grid-search. We have used this approach in some of the experiments that we describe below.

Regularization

A closer inspection of Eq. 3.14 shows that there is a trivial way to maximize the objective, which is to drive all latent representatives infinitely far apart from one another: To see this, note that we can re-write the rhs. of Eq. 3.14 as

$$\Omega_0 + \frac{1}{N}\sum_{\alpha} \log \sum_{\beta} \frac{k(\boldsymbol{z}^\alpha, \boldsymbol{z}^\beta)}{\sum_{\beta'} k(\boldsymbol{z}^\alpha, \boldsymbol{z}^{\beta'})} k(\boldsymbol{y}^\alpha, \boldsymbol{y}^\beta),$$

which shows that we can express the objective function as a linear combination of data-space kernel function evaluations. The weights in the linear combination are given by a "softmax", which depends on the latent space elements. The best we can do to increase the objective is by turning the softmax into an actual $\max$-function that picks out the best data-space kernel instead of averaging them. In order to turn the soft-max into a max function, however, we just need to drive the latent space elements away from each other.

To obtain a maximum that is meaningful, we therefore need to constrain the range of the latent elements to a finite region of the latent space, or to impose some other kind of

constraint on Z. A natural constraint would be

$$\mathrm{tr}(C_Z) \leq r \tag{3.21}$$

where C_Z denotes the latent covariance matrix, or an estimate $\frac{1}{N}Z^TZ$. This constraint restricts the variances of the latent variables to a maximally allowable power r.

In practice, this constraint amounts to imposing an L2-penalty on the matrix of latent representatives. An alternative to solving an optimization problem with Eq. 3.21 as a constraint, is to implement the constraint implicitly and solve an unconstrained problem instead, by adding $\frac{\lambda}{N}\mathrm{tr}(C_Z)$ to the objective function (similarly for other types of constraint), where λ controls the amount of regularization.

The effect of this penalty is that it controls the overall scale of the latent elements and thereby regularizes the model. Interestingly, such a restriction on the allowable latent space power parallels the necessary power restriction on input variables when optimizing a channel capacity [Cover and Thomas, 1990]: Without a constraint we get a solution that is perfect but meaningless. Intuitively, the need to constrain the latent representation reflects the fact *we cannot learn without making assumptions.* A perfect model of the data, that we could approach arbitrarily closely by using a weaker and weaker regularizer (for example, by driving λ to 0), would give us an extremely good model in terms of reconstruction error. But it would be of no more use than the original data-set itself, since it would simply fit the noise (and thereby lose the essential distinction between signal and noise.)

Using an L2-penalty is a common way to restrict the latent variables in many embedding methods. The main reason is computational: An L2-constraint can lead to a convenient optimization problem. In particular, spectral methods typically rely on an L2-constraint on the latent variables in order to obtain the eigen-decomposition as the solution (see, for example, [Roweis and Saul, 2000], [Tenenbaum et al., 2000], [Schölkopf et al., 1998]). However, depending on how we assume the data is distributed along the manifold, it could be desirable to restrict higher order moments, or to constrain the latent variables in other ways. Regularization amounts to *making assumptions*. Imposing constraints on the latent variable

realizations allows us to express prior knowledge and to define what we actually mean by "signal" and what we mean by "noise". Often, an L2-penalty does not strongly interfere with our assumptions and is justified. However in cases in which we have some knowledge about some aspects of the latent variables, an embedding based on an eigen-decomposition would force us to neglect that knowledge.

Annealing

Controlling the complexity of a model can be useful during optimization as a way to find a good local optimum of the objective function. By setting λ to a large value in the beginning and decreasing slowly during optimization, we can perform a kind of "deterministic annealing" to find and track a good optimum of the objective function. This type of approach is generally known as *continuation* or *path following* (see, for example, [Allgower and Georg, 1993]). Its application in unsupervised learning problems has been studied and described in detail, for example, by [Meinicke, 2000].

We have used this procedure in some of our experiments, and have found that it is a very effective way to overcome problems of bad local maxima, avoiding the necessity, for example, to perform multiple restarts or other tricks to obtain good solutions. Annealing makes the experiments relatively invariant to initial conditions (up to a rotational invariance inherent in the objective function). For more details on the optimization, see also the respective sections themselves.

3.1.2 Example

Figure 3.1 shows as an example the result of applying KIE to a two-dimensional "S"-curve manifold embedded in three dimensions. The data-set consists of 2000 points sampled uniformly from the "S"-shaped sheet. The right plot shows a two-dimensional embedding with a color-coding scheme that reveals the underlying mapping.

We used an L4-penalty on the latent space as the regularizer. The contours of the L4-

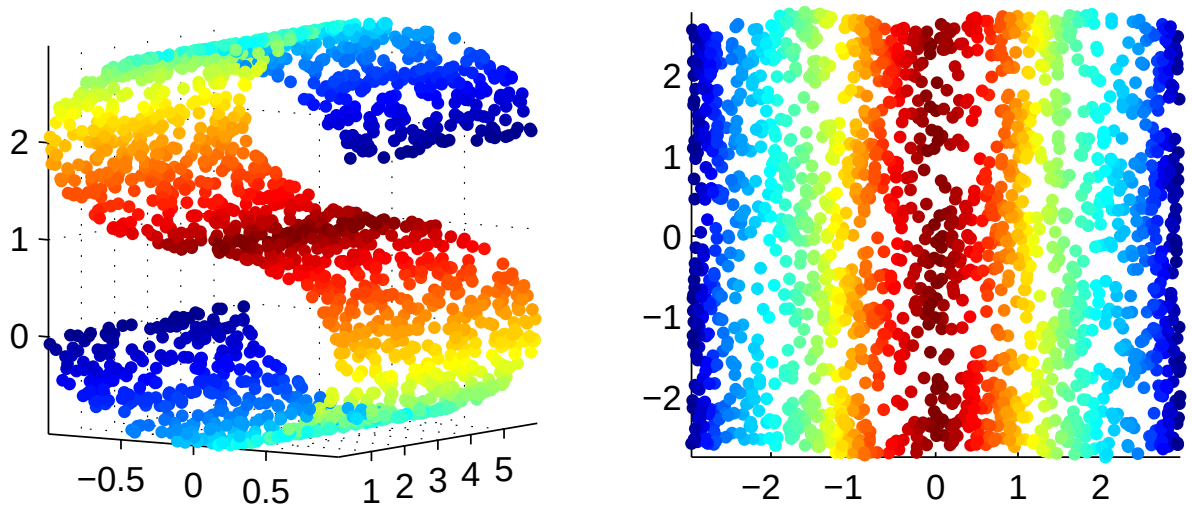

Figure 3.1: Three-dimensional "S-curve" data-set (left) and its two-dimensional embedding (right). The alignment of the embedding with the coordinate axes is encouraged with a penalty on the latent variables.

norm in 2 dimensions look like a "rounded box" that is aligned with the coordinate axes. Using this regularizer therefore encourages the embedding to align with the coordinate axes as well, as can be seen in the figure. (Note that this kind of alignment would be impossible to achieve using, for example, a spectral method.)

We used a Gaussian kernel with $h = 10.0$. To train the model, we initialized the $\boldsymbol{z}_i$ to small random values, and used the annealing procedure described above, with λ set to 0.1 initially and multiplied by 0.8 for 20 steps. In each step we optimized Eq. 3.14 using simple unconstrained optimization. We have repeated the experiment ten times with different random initializations and with no noticeable difference in the result (except that in some cases the embedding is rotated by 90 degrees in the latent space).

We also experimented with a wide range of kernel bandwidths, again without significant differences in the results. Note that, regardless of the bandwidth, the kernel density estimate necessarily underestimates the true density in this data-set. The data-set is drawn from an infinitely thin, two-dimensional sheet in the three-dimensional space, so the density is

infinite within this sheet. Even though the estimate is necessarily sub-optimal, KIE finds the underlying two-dimensional structure. The reason for this robustness is that more important than an accurate density estimate is the fact that the similarity structure in data-space is captured well.

3.1.3 The forward- and backward-mappings

Since we learn a full probabilistic model of the joint density $p(\boldsymbol{z}, \boldsymbol{y})$, deriving both the forward- and backward-mapping is straightforward.

A joint density $p(\boldsymbol{z}, \boldsymbol{y})$ implicitly contains all possible, stochastic dependencies between $\boldsymbol{z}$ and $\boldsymbol{y}$. A common way of making such dependencies explicit, given a joint density model, is by using the *regression functions* associated with the joint. The regression functions are defined as the conditional expectations $E[\boldsymbol{y}|\boldsymbol{z}]$ and $E[\boldsymbol{z}|\boldsymbol{y}]$, which give us the forward- and backward-mapping, respectively.

Since we learn a joint kernel density estimate over $\boldsymbol{z}$ and $\boldsymbol{y}$, these functions take a particularly simple form here. A straightforward calculation shows that a conditional expectation under a KDE with zero-mean kernel functions takes the form of a simple kernel expansion (see, for example [Bishop, 1995]). In our case, we get:

$$f(\boldsymbol{z}) = E[\boldsymbol{y}|\boldsymbol{z}] = \sum_{\alpha} \frac{k(\boldsymbol{z}, \boldsymbol{z}^{\alpha})}{\sum_{\beta} k(\boldsymbol{z}, \boldsymbol{z}^{\beta})} \boldsymbol{y}^{\alpha} \tag{3.22}$$

and

$$g(\boldsymbol{y}) = E[\boldsymbol{z}|\boldsymbol{y}] = \sum_{\alpha} \frac{k(\boldsymbol{y}, \boldsymbol{y}^{\alpha})}{\sum_{\beta} k(\boldsymbol{y}, \boldsymbol{y}^{\beta})} \boldsymbol{z}^{\alpha} \tag{3.23}$$

The forward mapping is given by a linear combination of data-points and the backward-mapping by a linear combination of latent space elements. The weights are normalized kernel functions. Because of the normalization, the mappings are actually convex combinations of the respective elements. We never leave the convex hull of the data-points or latent-space elements, when applying any of the mappings.

In contrast to previous non-parametric methods, we get both mappings not just one. Both have a simple form and can be computed very efficiently. The backward-mapping takes a form similar to the out-of-sample extension that has been suggested for kernel PCA [Bengio et al., 2004]. [Memisevic, 2003] used the mapping together with a latent space error criterion to obtain a spectral embedding method that automatically comes with a backward-mapping. In Section 2.3.2 we generalized the approach to arbitrary basis functions, by showing that they are generalization of PCA. However, we obtain only the backward-mapping that way.

[Memisevic, 2003] and [Meinicke et al., 2005] describe a model that uses Eq. 3.22 as the forward mapping and minimizes observable space reconstruction error for training. [Lawrence, 2004] use a similar forward mapping using a Gaussian process. In Section 2.3.2 we discuss a generalization of these approaches, too. However, all these methods do not come with a backward-mapping. To solve the problem, [Memisevic, 2003] suggested defining the backward-mapping as the solution to a non-linear optimization problem, or combining the backward- and forward-mapping based models. [Lawrence and Quinonero-Candela, 2006] describe a heuristic that endows the GPLVM with a backward-mapping by explicitly parameterizing the latent space elements as functions of the observables.

It is interesting to note that rather than modelling only the conditional expectations in Eqs. 3.22 and 3.23, we could similarly obtain estimates of other statistics as well. Conditional *variances*, for example, could be used to obtain estimates of uncertainty, which is important in many applications.

3.1.4 Related methods

Non-parametric estimates of information theoretic quantities have been used in many different contexts (see *eg.* [Principe et al., 1999] for a general overview, or [Torkkola, 2003] and [Viola et al., 1996] for applications in embedding contexts). In contrast to all these methods, KIE does not model parametric transformations between the latent and the observable space.

Instead, in the spirit of modern embedding methods such as LLE or SNE, we parameterize the embedding objective directly in terms of the latent data representatives themselves. Advantages of this approach are that (a) it is naturally nonlinear, (b) it only depends on a single group of parameters (the latent representatives), and as a result does not require alternating optimization schemes akin to the EM-algorithm, (c) it allows us to include side-information and thereby to supplement the embeddings with supervision signals as we will show below.

It is interesting to note that we can view Eqs. 3.23 and 3.22 as maximizing *conditional log-likelihoods*, which shows that we can interpret KIE as a kind of non-parametric regression method itself. The objective (Eq. 3.13) can then be viewed as an estimate of $\log p(Y|Z)$, and defines a kind of regression on latent variables. However, the presence of kernels in the latent- and data-space assures that after training, both the forward- and backward-mapping can be computed efficiently.

Using non-parametric regression for embedding has been suggested also by [Meinicke et al., 2005] and [Lawrence, 2004] and is discussed in detail in Section 2.3.2. Besides the presence of the forward- and backward-mapping when using KIE, there are a few further notable differences to those approaches. A difference to the former method is that KIE maximizes likelihood instead of reconstruction error, and therefore does not need to postulate an explicit noise model in the data space. A further advantage is that the optimization used to compute the embedding does not scale with the dimensionality of the data space.

In contrast to the GPLVM [Lawrence, 2004], the KIE objective function (Eq. 3.14) scales only quadratically (instead of cubically) with the number of data-points and is therefore scalable to much larger data-sets without the need for approximations and heuristics to speed up the optimization. The cubic complexity makes the GPLVM-model difficult to fit in real-world tasks and requires active-set heuristics and other optimization tricks on (see, for example, [Grochow et al., 2004]).

The fact that training the KIE model, in contrast to both other methods does not scale with the dimensionality of the data-space can be an additional advantage especially in very

high-dimensional data-spaces. Once all kernel matrices have been computed, the training complexity is only $O(N^2q)$.

Both, [Meinicke et al., 2005] and [Lawrence, 2004] are forward-regression models, as discussed in Section 2.3.2, and therefore define the forward mapping implicitly. This is in contrast to most spectral methods, which define the backward-mapping instead. Note, that KIE learns both, the forward and backward-mapping (Eqs. 3.22 and 3.23).

A further advantage over both methods is that KIE's information theoretic view suggests new ways of introducing supervision signals as we shall show.

3.1.5 Experiments

'Noisy S'

A standard problem that requires the presence of both a forward and a backward mapping is noise-reduction. After computing an embedding for noisy data, we can obtain a de-noised version for a noisy data-point $\boldsymbol{y}$ as $f(g(\boldsymbol{y}))$, that is by projecting into the latent space and back.

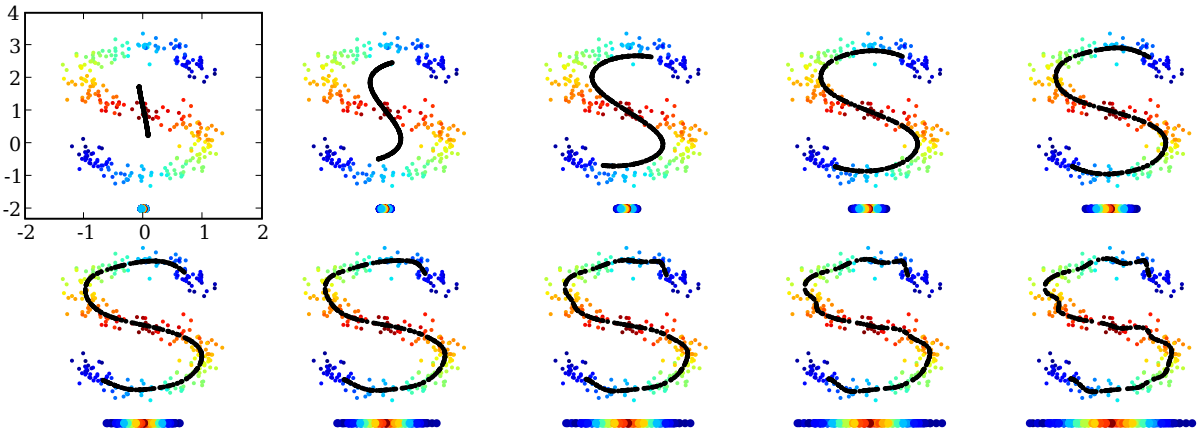

Figure 3.2: De-noising example. Test-data is projected onto a one-dimensional, noisy "S"-shaped manifold by computing $f(g(\boldsymbol{y}^\alpha))$ for each test-case $\boldsymbol{y}^\alpha$. Ten iterations of the annealing schedule are shown. The one-dimensional codes for the test-cases are show below each plot.

To demonstrate the use of forward and backward mappings in KIE, we used the 'noisy S' toy data-set shown in figure 3.2 We used a sample of 300 points from the one-dimensional manifold embedded in 2d. We computed an embedding with Gaussian kernels with bandwidth $\frac{1}{2}$ in the data-space. We set λ to 1.0 initially and decreased by multiplying with 0.8 for 50 times during the optimization. In each plot we show the projection of a *test-set* containing also 300 points onto the manifold defined by the model. The figure shows how the embedding unfolds in the data-space, and eventually overfits. This occurs after about 35 training iterations as can be seen from the plots. The optimal value for the regularization parameter λ therefore is about 0.001 in this problem.

Oil flow

The oil flow data-set (see [Bishop et al., 1998]) is a common benchmark data-set for testing dimensionality reduction methods. The training set (which is the commonly used sub-set for visualization) consists of 1000 12-dimensional measurements, that fall into three different classes. Figure 3.3 shows a two-dimensional visualization learned with KIE using the power-penalty. Different classes are depicted by using different colors and different symbols. For training we initialized $\boldsymbol{Z}$ to small random values. We set λ to 0.1 initially, which we decreased by multiplying with 0.8 for 50 iterations for annealing. We set h_Y to the optimal leave-one-out-bandwidth, which is approximately 8.23 on this data-set. (We used the raw data without any preprocessing). The left plot shows the two-dimensional PCA-representation for comparison. PCA is obviously not able to separate the classes very well using two dimensions. The plot in the middle shows the KIE-solution after 15 iterations of annealing. At this point λ is about 0.004. The right plot shows the KIE-solution after 50 iterations, where λ is less than 10^{-5}. At this point KIE's tendency to pull points apart from each other starts to become apparent, but there is still a strong separation of classes. The KIE-visualizations show not only the low-dimensional representatives but also the latent space densities (overlayed) using gray-values, where darker means higher density. The ability to explicitly represent density is a feature

that KIE shares with the methods in [Meinicke et al., 2005] and [Lawrence, 2004]. Spectral methods do not have this possibility. It has been shown to be useful especially in interactive tasks such as the analysis of motion capture data. We discuss this kind of application in the following.

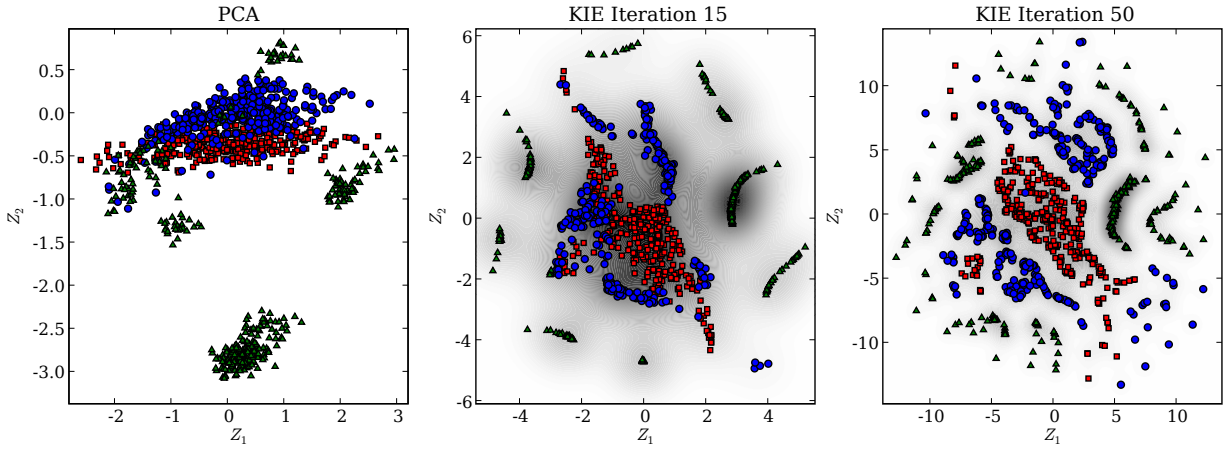

Figure 3.3: Visualization of oil flow data. Left: PCA. Middle and right: KIE. In the KIE-plots the latent space density is overlayed using gray-values.

KIE versus spectral embedding

Spectral methods, such as kernel PCA, LLE, and many others have gained a lot of popularity in the last few years. The main advantage of spectral methods over KIE and other iterative methods is that they have closed-form solutions. One draw-back of spectral methods, as we point out above, is the difficulty of including prior knowledge about the latent representation.

In this experiment we show evidence for a different problem with spectral methods. As we describe in Section 2, spectral methods usually do not come with a forward-mapping. [Memisevic, 2003] use a kernel density estimate in the joint input-/latent space to obtain a spectral method that comes with both the forward- and the backward-mapping. The motivation for that approach is that, as we describe in Section 3.1.3, a joint kernel-density estimate allows us to define both the mappings as kernel-smoothers. And minimizing reconstruction

error in the latent space gives rise to a spectral solution if the backward-mapping takes the form of a kernel smoother (see Section 2.3.2).

A natural approach to obtain also the forward-mapping is therefore to compute the spectral solution (which by itself does not come with a forward mapping) and then to optimize a latent-space kernel bandwidth in order to obtain the *joint* kernel density estimate in the input-/latent space. (Instead of estimating the bandwidth, [Memisevic, 2003] equivalently re-scale the learned latent representation while holding the bandwidth fixed.) Given the joint density estimate, both mappings can then defined as the regression functions (Eqs. 3.22 and 3.23).

This solution seems very natural, since retrospectively it amounts to using a kernel density estimate in the joint input-/latent space. Note, however, that this approach does not use a kernel in the latent space when computing the embedding. The forward-mapping is determined by estimating the latent space bandwidth *after* the embedding has been computed. The approach can therefore be interpreted also as performing *supervised learning* to estimate the forward mapping, after computing the embedding using a spectral solution. This is in contrast to KIE, which defines the joint kernel density estimate *before* computing the embedding. With KIE both the embedding and the mappings are learned simultaneously by maximizing the mutual information criterion.

We now describe an experiment showing how decoupling learning $\boldsymbol{Z}$ from learning the mappings can give rise to far inferior solutions in practice as one would expect. We used the noisy S-curve data-set that we describe in the beginning of this section. We trained the spectral method described in [Memisevic, 2003] and subsequently rescaled the latent representation such that the resulting reconstruction error (computed by projecting onto the learned manifold) on a validation set was minimized. For comparison, we then used the resulting set of latent representatives as an initialization for KIE. We set the KIE observable space bandwidth using a grid search on the leave-one-out log-likelihood and the regularization parameter λ using the validation set. We also compared with training KIE using annealing.

We initialized $\boldsymbol{Z}$ randomly in that case and set the final regularization parameter λ using the validation set.

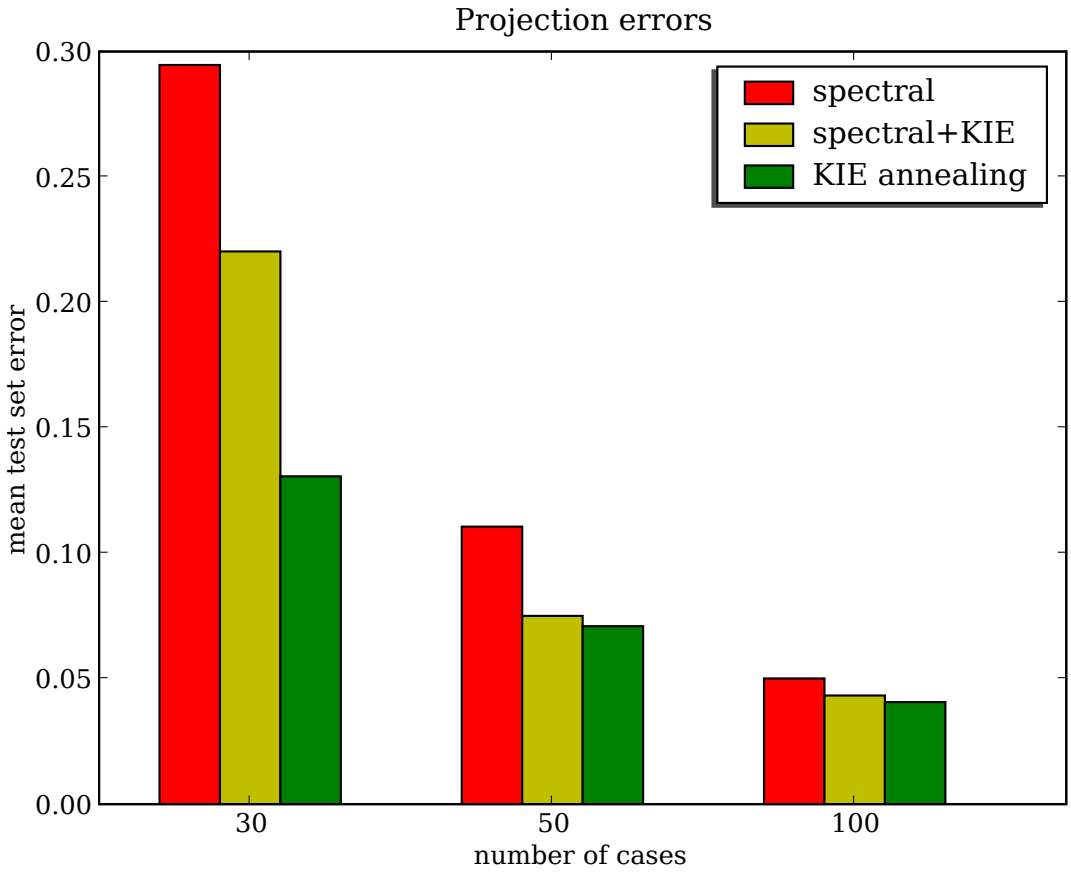

Figure 3.4: Test-set reconstruction errors from (i) spectral embedding (left-most bar in each experiment), (ii) KIE initialized with a spectral method (middle bar in each experiment), and (iii) KIE trained with annealing (right-most bar in each experiment).

To assess how well the three approaches can model the data we computed the reconstruction error on a test-set containing 1000 cases. Figure 3.4 shows the errors, averaged over 50 runs, for different training set sizes. The plot shows that optimizing the spectral solution using KIE can give a strong performance improvement on the test-set as compared to using just the spectral solution (followed by supervised optimization of the latent space bandwidth),

in particular when the training set is small. However, the plot also shows that annealing gives rise to yet far better results.

It is not very surprising that "refining" a spectral solution for $\boldsymbol{Z}$ by optimizing the KIE objective is superior to simply estimating a latent space bandwidth. Optimizing the latent space bandwidth can be viewed as rescaling $\boldsymbol{Z}$, which in turn can be viewed as a highly constrained form of optimizing $\boldsymbol{Z}$. Allowing the latent representatives to move independently provides much more freedom in the optimization. In both cases the validation data allows us to avoid overfitting. However, it is important to note that annealing yields by far the best results. [Memisevic, 2003] describe similar findings when comparing spectral initializations for a forward-mapping based model with annealing.

Modeling motion capture data

Analyzing and modifying human motion capture data has become a common application for dimensionality reduction methods (see, for example, [Grochow et al., 2004] [Wang et al., 2006]). To demonstrate the use of KIE on this type of problem we used the data from the CMU Graphics Lab Motion Database[1]. We used a walk cycle consisting of 298 frames, and trained a model on the 59-dimensional raw data without performing any preprocessing other than eliminating the first three channels to remove drift (the original data has 62 dimensions). Figure 3.5 (bottom left) shows a two-dimensional latent representation of the walk cycle learned by KIE with power penalty (we set λ to 0.05). The plot depicts the latent representatives along with the latent space density (here, higher density is shown using brighter color). A few latent representatives are marked with the corresponding frame number, and the corresponding poses are shown in the top-row of the figure. The visualization of the poses was created with publicly available software [Lawrence, 2006]. For training we used the power penalty and annealing as before. The annealing schedule and how the representation unfolds in the latent space is shown separately in figure 3.6. That figure also

[1] http://mocap.cs.cmu.edu/

shows the latent space density and how it is becoming more complex as training progresses. It is interesting to note that KIE provides a smooth latent representation of the motion capture data. Smoothness of the representations can be difficult to achieve with other models, such as the GPLVM, which is therefore often used in combination with explicit back-constraints to enforce smoothness [Lawrence and Quinonero-Candela, 2006].

A common problem with motion capture data is missing values [Grochow et al., 2004]. Here we show how we can use a learned manifold to fill in missing values. We randomly removed 50% of the entries from the 298×59 data-matrix to simulate drop-out. A simple but far from optimal heuristic for filling in a missing value is by using the nearest non-missing entry within the same channel. The 59 dimensions contain relatively smooth time-series, so this procedure does alleviate the missing value problem to some degree. But it also leads to several glitches and can fail if there are multiple adjacent drop-outs. In the following we describe how we can use the heuristic as an initial solution that can subsequently been improved with KIE.

Applying the nearest-neighbor heuristic gives rise to data that can be thought of as being corrupted by a strange type of (very non-Gaussian) noise. After pre-processing the data with the nearest-neighbor-heuristic, we can de-noise it by learning a latent representation and then projecting the data onto the manifold as described in the beginning of this section. That is, we compute $f(g(\boldsymbol{y}))$ as the de-noised version for each frame $\boldsymbol{y}$.

We trained a KIE model with 10 latent dimensions and used annealing with $\lambda = 0.01$ as the final regularizer. Training takes a few minutes on a 3.4 GHz Pentium. Projecting the 298 frames onto the manifold takes a fraction of a second. The model smoothes out the data and removes artifacts from the fill-in procedure. The projection does over-smooth the data a little, but it leads to a much better visual impression of the walk[2].

Note that we do not make use of any time-information for filling in missing entries, except

[2]The videos of the walking sequence are available at: http://learning.cs.toronto.edu/~rfm/thesis/
The filenames for the videos are: "walkOriginal.avi" for the original walking sequence, "walkNN.avi" for the nearest neighbor heuristic, "walkKie.avi" for the de-noised sequence

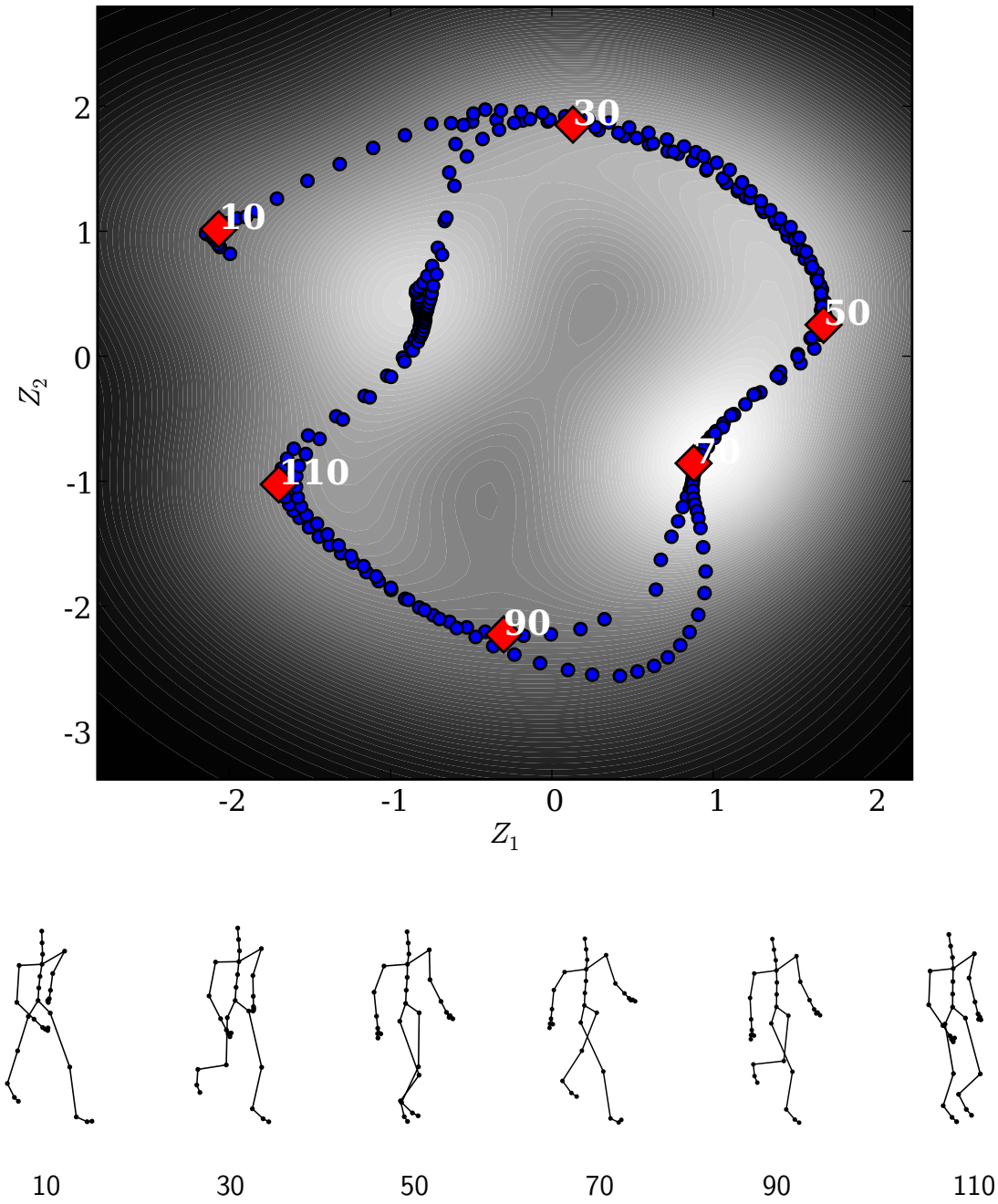

Figure 3.5: KIE applied to motion capture data. Top: Six frames of a walk cycle. Bottom: Two-dimensional latent representation. The six frames are marked in the plot.

when applying the nearest-neighbor heuristic. It would be possible (and straightforward) to incorporate constraints that encode time-structure, for example, by encouraging latent

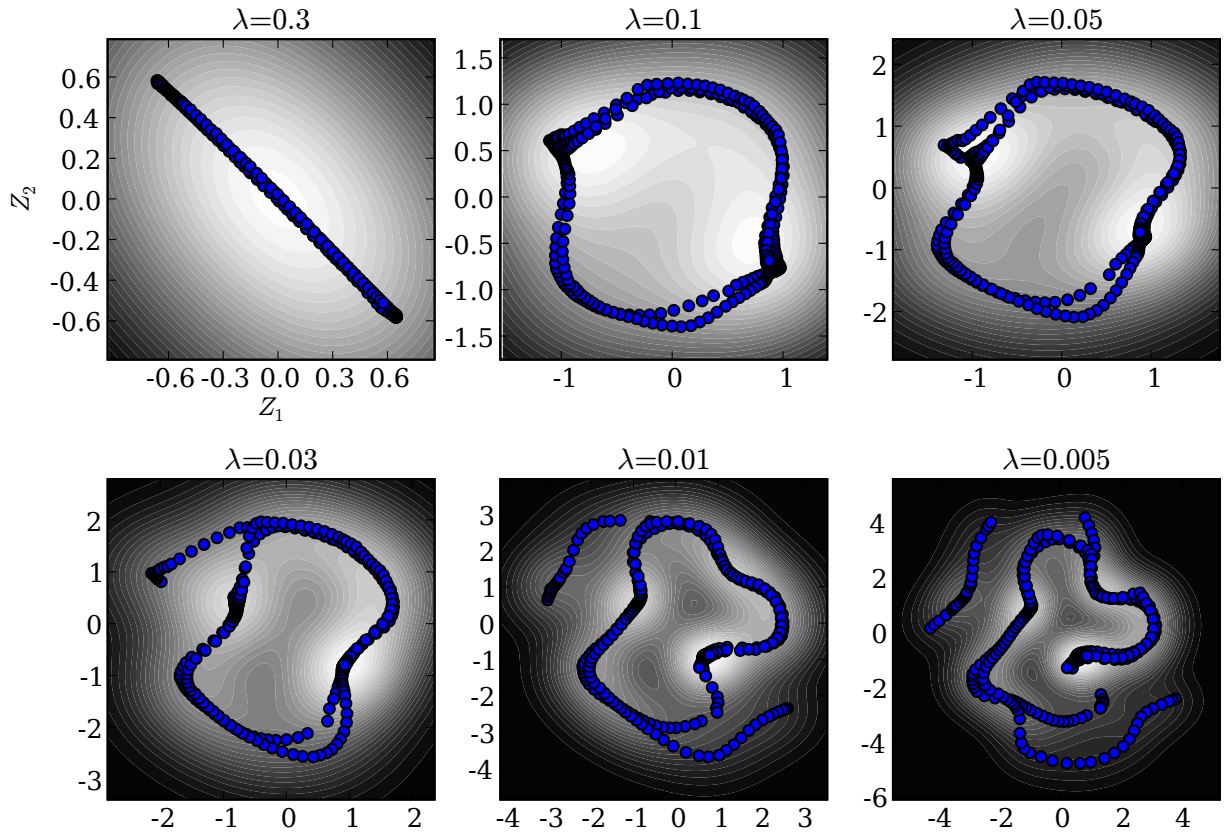

Figure 3.6: Unfolding two-dimensional representation for a walk cycle.

representatives for adjacent frames to be close to each-other. However, since the motion-capture data evolves relatively smoothly in time, it is not clear to what degree this could further improve the results.

3.2 Conditional Embeddings

A well-known problem with standard embedding methods is that they typically compute representations that capture generic similarity structure in the data, but do not necessarily reflect the kind of variability that is important for a specific task. In this case, supervision signals can help adjust the methods so that they actually capture the kind of variability we are interested in (see *eg.* [Tenenbaum and Freeman, 2000]). A particularly useful way to make

use of extra information is by letting it *factor out* the kind of variability we are *not* interested in, and thereby to focus the embedding on the kind of variability that is important to the actual task at hand.

Encoding invariances into learning machines is a difficult problem. The most common approach consists in building knowledge about invariances into the learning architecture itself, as done, for example, in convolutional neural networks [LeCun et al., 1998]. Building invariances into a model itself, however, is a difficult engineering task that is somewhat contrary to the machine learning philosophy of using *data* itself to define the solution.

In the following we show how to specify invariances within the kernel information embedding framework, simply by using a special type of training data for this purpose. To cast the idea in information theoretic terms, let us introduce a random variable X that captures available extra information about the data. Now, the task of computing an embedding that reflects all of the variability in the data, *except* for the variability already represented by X, has a natural solution: We need to maximize the *conditional* mutual information between Y and a latent variable Z, *given* X. Conditional MI can be defined as [Cover and Thomas, 1990]

$$\begin{aligned} I(Y;Z|X) &= H(Y|X) - H(Y|Z,X) \\ &= H(X,Z) - H(X,Y,Z) + \Omega_1, \end{aligned} \tag{3.24}$$

where Ω_1 contains terms that do not depend on Z. Conditional MI, similarly as MI itself, comes with an appealing intuition and answers the question: "What, on average, can random variable Z tell us about Y, and vice versa, given that we *know* X." By minimizing 3.24, we obtain factors that reflect the kind of variability in the data that is *not* represented by, or "correlated" with, X.

In practice, we now obtain a shared latent space in which dimensions that encode knowledge correspond to the random variable X and the remaining, learned, dimensions that are used to encode the remaining degrees of freedom correspond to Z. In other words, we reserve one or more latent space dimensions as the conditioning (the "X")-space and deliberately

position the latent elements $\boldsymbol{x}_i$ in this space according to the kind of variability that we want to encode. To encode, for example, grouping structure, we can cluster the elements $\boldsymbol{x}_i$ accordingly, or use an orthogonal ("one-hot"-) encoding; to express knowledge about a natural ordering in the data we can arrange the elements according to this ordering; etc. After placing the conditioning latent elements and defining a kernel for this space, maximizing the estimate $\hat{I}(\boldsymbol{Y}; \boldsymbol{Z}|\boldsymbol{X})$ yields the elements $\boldsymbol{z}^\alpha$ that capture the remaining degrees of freedom.

Similar as MI (Eq. 3.2), conditional MI decomposes into entropy terms (Eq. 3.24). We can obtain an estimate $\hat{I}(\boldsymbol{Y}; \boldsymbol{Z}|\boldsymbol{X})$ in complete analogy as before for MI, with a gradient that decouples in the same way, too. (Note, that the same restrictions regarding latent space power etc. apply). Similarly as before, the model is trained by gradient based optimization.

3.2.1 Discrete data

We assumed that data, latent variables and side-information are encoded as real-valued vectors, until now. Certainly, for the latent variables this is the natural representation if our goal is dimensionality reduction (as opposed to, say, clustering). Similarly, the observable data can be assumed to be real-valued; if it were drawn from a discrete distribution, simple maximum-likelihood estimation for the discrete distribution could give us all the information contained in the data.

However, it is easily possible for the side-information to be discrete: If the data can be naturally partitioned into a set of discrete "bins", for example, it could be desirable to encode group membership as side-information. It could be possible to encode this type of knowledge in a continuous space by using, for example, an orthogonal encoding and then applying a Gaussian kernel in this space. However, a continuous encoding does not seem quite appropriate for discrete data. A better choice is a kernel that is defined for discrete data in the first place. Note, however that it is necessary that the kernel gives rise to a valid density estimate.

A kernel that satisfies these requirements is given by the inner product of an orthogonal

encoding:

$$k(\boldsymbol{y}, \boldsymbol{y}^\alpha) = \delta_{\boldsymbol{y},\boldsymbol{y}^\alpha}. \tag{3.25}$$

This kernel obviously satisfies $\sum_{\boldsymbol{y}} k(\boldsymbol{y}, \boldsymbol{y}^\alpha) = 1$ and $k(\boldsymbol{y}, \boldsymbol{y}^\alpha) \geq 0$. We can obtain this kernel also as the limit of a Gaussian kernel (Eq. 3.17) defined on the orthogonal encoding kernel, where $h \to \infty$.

It is worth noting that learning with *vector-valued* discrete data can in theory always be interpreted as a simple discrete setting by simply enumerating the (exponentially many) discrete vector instantiations. While for practical purposes this observations is useless at first sight, computing a discrete kernel for vectors is possible using dynamic programming (see, for example, [Memisevic, 2006a]). The discrete kernel could therefore be useful also in a standard (non-conditional) embedding setting, where the task is, for example, to compute embeddings for discrete sequences.

3.2.2 Experiments

Untangling multiple latent factors

Figure 3.8 illustrates the use of an informed latent space as opposed to an uninformed one, using images from the COIL-database [Nene et al., 1996] as an example. The database consists of images of different objects with varying orientations. Thus, two main degrees of freedom give rise to the variability in the data: *Object identity* and *appearance* (viewing angle).

In our first experiment we used images of three kinds of car as shown in figure 3.7. The top row of figure 3.8 shows a three-dimensional embedding obtained by applying kernel PCA [Schölkopf et al., 1998], using a Gaussian kernel (Eq. 3.17) with $h = 5 \cdot 10^7$. (We obtained similar results using uninformed KIE.) The three different types of car are shown using different symbols and colors. The plot shows that the embedding captures the presence of three individual one-dimensional degrees of freedom. However, variability across classes is represented simultaneously in the same latent space as variability that is due to rotation.

As a result, both degrees of freedom are intermixed, and the fact that all three objects vary simply by rotation is not obvious.

Figure 3.7: Example car images from the COIL database. The two major modes of variability are due to (a) object identity and (b) angle of rotation.

The plot on the bottom left of the figure shows the embedding of the same objects, where variability due to class-membership has been removed ("factored out") from the latent representation: The "X"-space (the vertical axis in the figure) was obtained by simply placing representatives for the objects in three different clusters (located at $1.0, 2.0$ and 3.0). For training we initialized to small random values, fixed $\lambda = 0.6$ and used the same kernel bandwidth as for kernel PCA. (Annealing was not used in this experiment.) As before, we used simple gradient based optimization to maximize the conditional MI estimate. The resulting conditional embedding, given by the remaining two axes, captures the rotational degree of freedom that remains after class-membership has been factored out.

The plot on the bottom right shows a conditional embedding where the *appearance* (viewing angle) has been factored out. In order to encode angles as side-information we used a two-dimensional "knowledge"-space in which we arranged representatives in a circle. We also used a Gaussian kernel in this space.This representation of a periodic degree of freedom is certainly not optimal or exact, but it does approximately capture our belief about how the variability in appearance is structured in this data-set. The plot shows the conditional em-

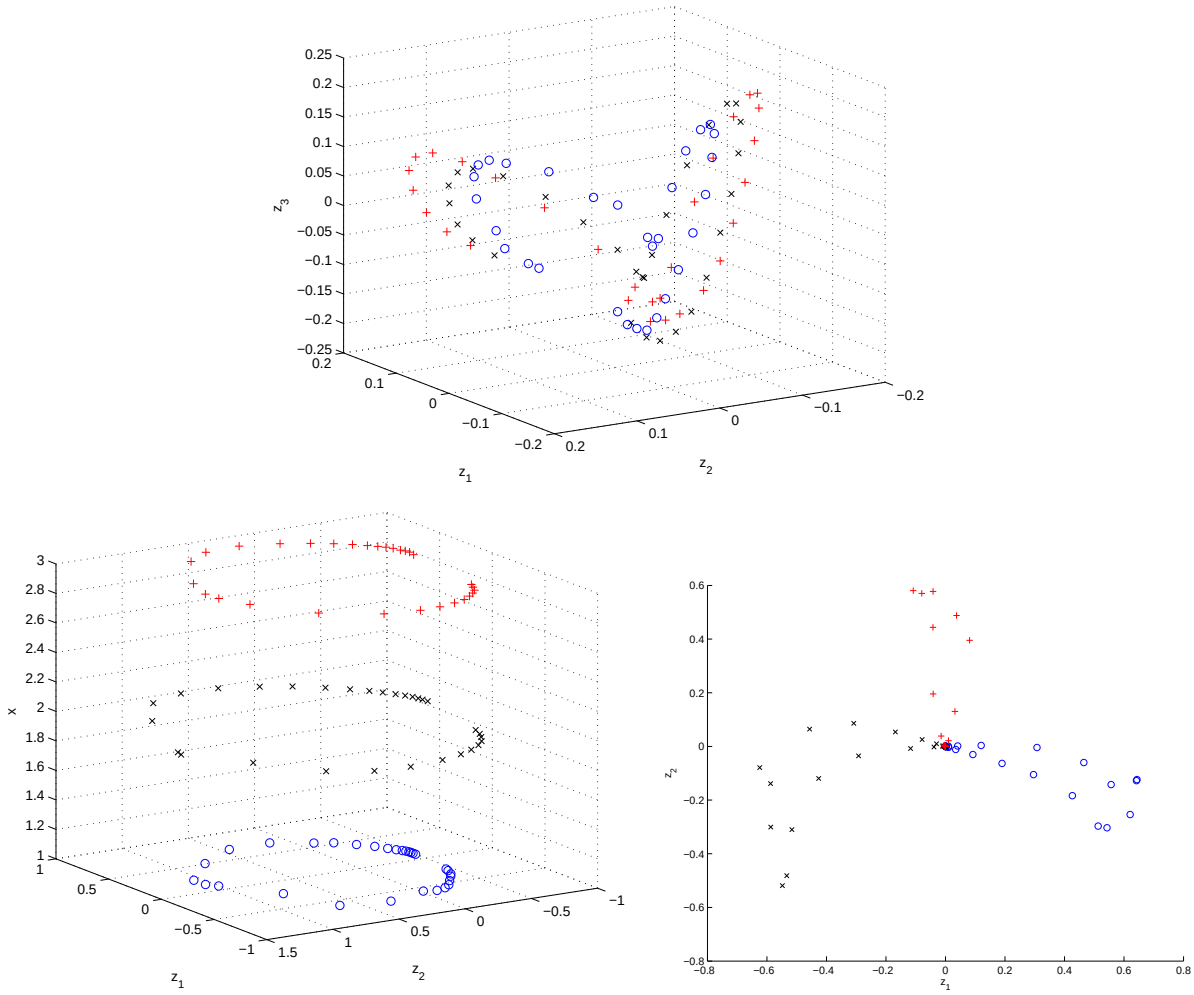

Figure 3.8: Three embeddings of the same set of car images from the Columbia Imaging Database (COIL), using different types of side-information. Object identity is encoded using different colors and different symbols. An uninformed (kernel PCA) embedding is shown on the top; an embedding informed about class membership on the bottom left; an embedding informed about appearance on the bottom right.

bedding given this structure: The model discovers a latent *class-space* in the remaining two dimensions (the two dimensions are shown in the plot). Even though *no* class-labels were provided, the conditional embedding clearly captures the separation into three classes, after appearance has been factored out.

The meaning of "nearest neighbors"

In another experiment, we trained the model on a subset of 30 images for each of the first 5 objects from the same database (ref. figure 3.9 for the objects). In one setting we used plain dimensionality reduction, in the other we informed the model about grouping structure by using an orthogonal encoding for class-membership and a Gaussian kernel with bandwidth 1.0 to obtain a latent "class"-space. We then picked for each object from the first class (the "duck"-images) the nearest neighbors in each of the remaining four classes in the "Z"-space. The four chosen neighbors are shown below each duck-image in figure 3.9. Since class membership has been factored out, the embedding-space captures the remaining degree of freedom very cleanly. We repeated the same neighborhood-picking procedure in the original data-space and in an uninformed latent space, that we obtained by running plain KIE with the same settings as before. For one of the classes ("wood block"-images) data-space similarity also captures the rotational degree of freedom. However, when the degree of freedom is too subtle (as in the other three object classes), it is not reflected in the data-space anymore. While it is not surprising that data-space similarity fails in this context, note that any uninformed latent space does, too, (see figure 3.9, bottom). The reason is that without class-membership factored out, any generic embedding algorithm tries to capture all factors underlying the overall similarity structure in the data. In this setting, for example, it tends to cluster the data according to class membership, while at the same time trying to capture the rotational degree of freedom in the *same* latent space. Ironically, neighborhood in the plain latent space consequently fails entirely to capture *any* of the structure present in the data. Neighborhood in the informed latent space on the other hand is meaningful, after some of the "nuisance" structure has been factored out.

Intelligent image manipulation

As an application of a conditional embedding applied to image manipulation, we consider the task of rendering face images. More specifically, we use a conditional embedding of images

Figure 3.9: Nearest neighbors in {informed latent space (top), data-space (middle), plain latent space (bottom)}.

from a face data-base in order to produce images of persons wearing glasses, if the persons are shown *without* glasses in the data-base. In other words, we use a conditional embedding to "give people glasses".

We use a version of the Olivetti face data-base[3], which consists of 400 images of faces (40 persons with 10 images for each person). The size of the images is 64×64 pixels. We train a conditional embedding on the raw pixel-data where the conditioning information $\boldsymbol{X}$ corresponds to the binary variable "HAS GLASSES". For this purpose, we use a one-dimensional variable $\boldsymbol{X}$ that is 1 for images that show a person with glasses and 0 for the others. We use a Gaussian kernel with variance 1.0 in this space and train an embedding with a 20-dimensional latent space conditioned on $\boldsymbol{X}$. After training we can obtain face-images that are rendered with glasses, by applying the forward-mapping $f(\boldsymbol{z})$, while fixing the $\boldsymbol{X}$-value to a positive value. Doing so on instances which are *not* shown with glasses in the training-set, results in a re-rendering that shows the same person *with* glasses.

Figure 3.10 depicts the original and the "glasses"-rendering for 8 subjects from the data-base. The figure shows that the model is able to infer good renderings of persons with glasses. While the procedure gives surprisingly good results on this fairly small data-set, we expect the approach to be much more effective when using a larger data-base. Note that, since the model is non-parametric, output-images are produced as convex combinations of training images, as we mentioned above. Changing the "glasses"-variable for an instance, has the effect that in the convex combination larger weights are assigned to those training-cases that are shown with glasses. The result is that the model tries to render a person with glasses by focussing on those images in the training-set that show a person with glasses and that are at the same time similar to the query case.

Separating the variability in the data-set into the two factors "identity" and "HAS GLASSES" makes it almost trivially easy to "flip the GLASS-bit" and produce new renderings for given persons. It is important to note that unlike a typical graphical model approach to solving a similar type of task, a conditional embedding requires absolutely no knowledge or understanding of the data-generating process. All our knowledge about the data and the task at hand enters the model in the form of supervision signals. Subtle reflectance and shading

[3]Available at http://www.cs.toronto.edu/~roweis/data.html

Figure 3.10: Intelligent image manipulation. Rows 1 and 3 show face images from the Olivetti data-base. Rows 2 and 4 show the corresponding persons rendered wearing glasses. The images with glasses are produced by first computing an embedding of face images that is conditioned on the binary variable "HAS GLASSES" and then "flipping the glass bit" on the shown persons.

effects, for example, are taken care of automatically by the model. Information about these and similar effects is represented implicitly in the training data instead of being hand-coded.

3.3 Discussion

We used Gaussian kernel functions in all of our experiments. While this is not necessary, and other kernel functions could be used as well, Gaussian kernels are convenient, especially because they are differentiable. Gaussian kernels do not have finite support, *ie.* they are non-zero everywhere. For efficiency it could be advantageous to use a (differentiable) kernel with finite support instead, such as the Quartic-kernel (see, for example [Meinicke et al., 2005]). Kernels with finite support can give rise to a sparse kernel matrices and can therefore reduce the number of calculations required in gradient computations.

As an alternative to imposing latent space constraints in order to regularize the model (Section 3.1.1) we could change all involved kernel densities (for example, Eq. 3.4) to use leave-one-out estimates instead. In that case, the model can no longer "cheat" by turning the softmax in Eq. 3.1.1 into a max-function. A similar leave-one-out trick has been used previously by [Memisevic, 2003], for example, to regularize a kernel based forward-model.

Using a leave-one-out estimate could make it possible to include the observable space bandwidth parameter in the gradient based optimization, and to learn it along with the latent space representatives. That way we could obtain a model whose only free parameter is the latent space dimensionality.

Chapter 4

Multiple relational embedding

In the previous chapter we discussed embedding in a non-parametric framework. We showed how side-information can be encoded by placing elements in a Euclidean space, and subsequently defining a kernel in this space. The dimensions of another, orthogonal, space were then used to encode the similarity structure that remains after factoring out the side-information. In this chapter we describe a variation of this idea where we let a latent variable model *itself* allocate latent space dimensions for embedding. Multiple relations can then be embedded jointly in a single latent space: Both the problem of defining a latent space to accommodate multiple relations and the task of finding low-dimensional codes are solved automatically and simultaneously by the model.

The distinction between data and extra information becomes fuzzy as a result: The input to the model is a set of similarity relations, one of which can (but does not need to) be based on Euclidean data similarity. A single latent space is used to embed all input relations.

4.1 Learning in a common latent space

The previous chapter discussed how we can impose structure onto a latent space by partitioning its dimensions into the learned latent variables on the one hand, and invariance information, that we want to factor out, on the other.

In other words, we explained structure encoded in the observable space kernel matrix using a *shared* latent representation. Fixing part of the representation allowed us to factor out the corresponding variability. A natural question is, whether it is possible to let the model *itself* partition the latent space into meaningful subgroups, instead of dictating the partitioning from the outset. In particular, using multiple subgroups of latent space dimensions could allow us to also to model multiple observed types of structure, instead of the single observable space kernel matrix.

In this chapter we use a set of *latent space transformations* to allow a single, shared latent space to model several, possibly overlapping similarity relations. The approach is applicable to any type of non-parametric model, and we demonstrate it using stochastic neighbor embedding [Hinton and Roweis, 2003]. The idea is illustrated in figure 4.1: A single latent space is used to model multiple similarity relations P^c, where c ranges from 1 to 5 in the illustration. Modelling multiple relations can be achieved by using multiple corresponding latent similarity matrices Q^c. In order to obtain these from a single, shared configuration of latent representatives, we also introduce the transformations R^c, one for each similarity type. Here, we consider diagonal transformations, which lead to a simple *weighting* of latent space dimensions. Unrelated similarity types can be accommodated using separate sets of latent space dimensions.

4.2 Multiple relational embedding

In the following we derive MRE as an extension to stochastic neighbor embedding (SNE). As above we let $\boldsymbol{Z}$ denote the matrix of latent space elements arranged column-wise. We let σ^2 be some real-valued neighborhood variance or "kernel bandwidth". Recall, that SNE finds a low-dimensional representation for a set of input data points $y^\alpha (\alpha = 1, \ldots, N)$ by first constructing a similarity matrix P with entries

$$P_{\alpha\beta} := \frac{\exp(-\frac{1}{\sigma^2}\|y^\alpha - y^\beta\|^2)}{\sum_\gamma \exp(-\frac{1}{\sigma^2}\|y^\alpha - y^\gamma\|^2)} \tag{4.1}$$

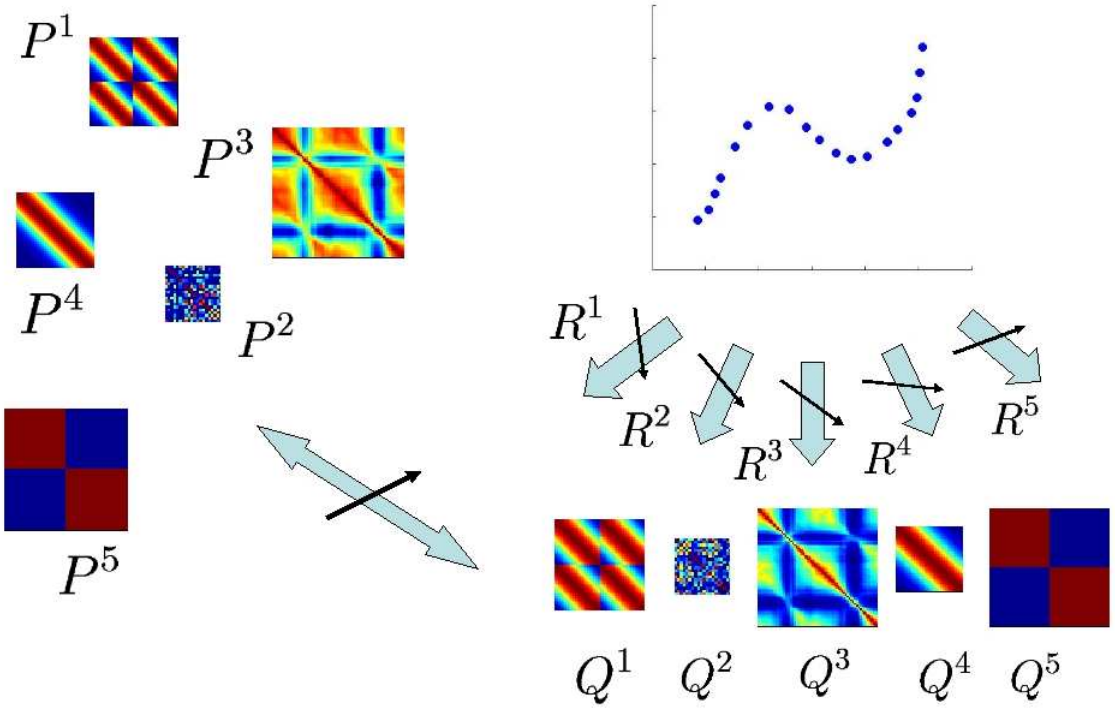

Figure 4.1: Illustration of multiple relational embedding.

and then minimizing (w.r.t the set of latent space elements $\boldsymbol{z}^\alpha (\alpha = 1, \ldots, N)$) the mismatch between P and the corresponding latent similarity matrix $Q(\boldsymbol{Z})$ defined by

$$Q_{\alpha\beta}(\boldsymbol{Z}) := \frac{\exp(-\|\boldsymbol{z}^\alpha - \boldsymbol{z}^\beta\|^2)}{\sum_\gamma \exp(-\|\boldsymbol{z}^\alpha - \boldsymbol{z}^\gamma\|^2)}. \tag{4.2}$$

For learning the latent representation $\boldsymbol{Z}$ we minimize the mismatch between P and Q, which is defined as the sum of Kullback-Leibler-divergences between the respective rows [Hinton and Roweis, 2003].

Our goal is to extend SNE so that it learns latent data representations that not only approximate the input space distances well, but also reflect additional characteristics of the input data that one may be interested in. In order to accommodate these additional characteristics, instead of defining a single similarity-matrix that is based on Euclidean distances in data space, we define several matrices P^c, $(c = 1, \ldots, C)$, each of which encodes some known type of similarity of the data. Proximity in the Euclidean data-space is typically *one* of

the types of similarity that we use, though it can easily be omitted. The additional types of similarity may reflect any information that the user has access about any subsets of the data provided the information can be expressed as a similarity matrix that is normalized over the relevant subset of the data.

At first sight, a single latent data representation seems to be unsuitable to accommodate the different, and possibly incompatible, properties encoded in a set of P^c-matrices. Since our goal, however, is to capture possibly overlapping relations, we do use a single latent space and in addition we define a linear transformation R^c of the latent space for each of the C different similarity-types that we provide as input. Note that this is equivalent to measuring distances in latent space using a different Mahalanobis metric for each c corresponding to the matrix $R^{cT}R^c$.

In order to learn the transformations R^c from the data along with the set of latent representations $\boldsymbol{Z}$ we consider the loss function

$$E(\boldsymbol{Z}) = \sum_c E^c(\boldsymbol{Z}), \tag{4.3}$$

where we define

$$E^c(\boldsymbol{Z}) := \frac{1}{N}\sum_{\alpha,\beta} P^c_{\alpha\beta} \log\left(\frac{P^c_{\alpha\beta}}{Q^c_{\alpha\beta}}\right) \quad \text{and} \quad Q^c_{\alpha\beta} := Q_{\alpha\beta}(R^c\boldsymbol{Z}). \tag{4.4}$$

Note that in the case of $C = 1$, $R^1 = \mathrm{I}$ (and fixed) and P^1 defined as in Eq. (4.1) this function simplifies to the standard SNE objective function. One might consider weighting the contribution of each similarity-type using some weighting factor λ^c. We found that the solutions are rather robust with regard to different sets of λ^c and weighted all error contributions equally in our experiments.

As indicated above, here we consider diagonal R-matrices, which simply amounts to using a rescaling factor for each latent space dimension. By allowing each type of similarity to put a different scaling factor on each dimension the model allows similarity relations that "overlap" to share dimensions. Completely unrelated or "orthogonal" relations can be encoded by using disjoint sets of non-zero scaling factors.

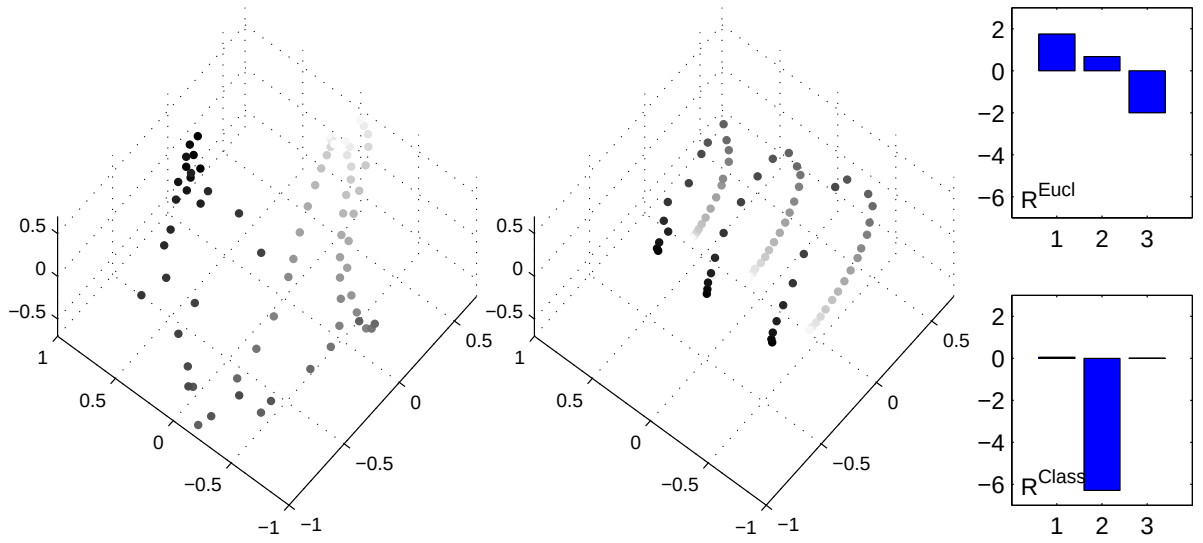

Figure 4.2: Embedding of images of rotated objects. Left: SNE, right: MRE. Latent representatives are colored on a gray-scale corresponding to angle of rotation in the original images. The rightmost plots show entries on the diagonals of latent space transformations R^{Eucl} and R^{Class}.

The gradient of $E(\boldsymbol{Z})$ w.r.t. a single latent space element $\boldsymbol{z}^l$ takes a similar form to the gradient of the standard SNE objective function [Hinton and Roweis, 2003] and is given by

$$\frac{\partial E(\boldsymbol{Z})}{\partial \boldsymbol{z}^\alpha} = \frac{2}{N} \sum_c \sum_\beta \left(P^c_{\beta l} + P^c_{\alpha\beta} - Q^c_{\alpha\beta} - Q^c_{\beta\alpha} \right) R^{cT} R^c (\boldsymbol{z}^\alpha - \boldsymbol{z}^\beta). \tag{4.5}$$

The gradient w.r.t. to R^c is given by

$$\frac{\partial E(\boldsymbol{Z})}{\partial R^c} = 2R^c \left[\sum_\alpha \sum_\beta \left(P^c_{\alpha\beta} - Q^c_{\alpha\beta} \right) (\boldsymbol{z}^\alpha - \boldsymbol{z}^\beta)(\boldsymbol{z}^\alpha - \boldsymbol{z}^\beta)^T \right] \tag{4.6}$$

For diagonal R^c the gradient simplifies. W.r.t. to a single entry of diagonal of R^c we get:

$$\frac{\partial E(\boldsymbol{Z})}{\partial R^c_{ll}} = \frac{2}{N} R^c_{ll} \left[\sum_\alpha \sum_\beta \left(P^c_{\alpha\beta} - Q^c_{\alpha\beta} \right) (\boldsymbol{z}^\alpha_l - \boldsymbol{z}^\beta_l)^2 \right], \tag{4.7}$$

where $\boldsymbol{z}^\alpha_l$ denotes the l^{th} component of latent representative α .

As an illustrative example we ran MRE on the set of images from the Columbia object images library (COIL) [Nene et al., 1996], that we used in Chapter 3. We took same three

subsets of images depicting toy-cars (recall that each subset corresponds to one of three different kinds of toy-cars) and embedded the first 30 images of each of these subsets in a three-dimensional space. We used two similarity relations: The first, P^{Eucl}, corresponds to the standard SNE objective; the second, P^{Class}, is defined as a block diagonal matrix that contains homogeneous blocks of size 30×30 with entries $(\frac{1}{30})$ and models class membership, i.e. we informed the model using the information that images depicting the same object class belong together.

We also ran standard SNE on the same dataset[1]. The results are depicted in figure 4.2. Similarly as kernel PCA in Chapter 3, SNE's unsupervised objective, which tries to preserve Euclidean distances, leads to a representation where class-membership is intermingled with variability caused by object rotation (left-most plot), in the MRE approximation the contribution of class-membership is factored out and represented in a separate dimension (next plot). In contrast to the conditional embedding (Chapter 3) here the entries of the R-matrices can tell us something about the relation between the different similarity-types. The two right-most plots show the diagonal on the diagonal of the two matrices. R^{Class} is responsible for representing class membership and can do so using just a single dimension. R^{Eucl} on the other hand makes use of all dimensions to some degree, reflecting the fact that the overall variability in "pixel-space" depends on class-membership, as well as on other factors (here mainly rotation). Note that with the variability according to class-membership factored out, the remaining two dimensions capture the rotational degree of freedom very cleanly.

[1]For training we set σ^2 manually to $5 \cdot 10^7$ for both SNE and MRE and initialized all entries in X and the diagonals of all R^c with small normally distributed values. In all experiments we minimized the loss function defined in Eq. (4.3) using Carl Rasmussens' matlab function "minimize" for 200 iterations (simple gradient descent worked equally well, but was much slower).

4.2.1 Partial information

In many real world situations there might be side-information available only for a subset of the data-points, because labelling a complete dataset could be too expensive or for other reasons impossible. A partially labelled dataset can in that case still be used to provide a hint about the kind of variability that one is interested in. In general, since the corresponding transformation R^c provides a way to access the latent space that represents the desired similarity-type, a partially labelled dataset can be used to perform a form of supervised feature extraction in which the labelled data is used to specify a kind of feature "by example". It is straightforward to modify the model to deal with partially labelled data. For each type of similarity c that is known to hold for a subset containing N^c examples, the corresponding P^c-matrix references only this subset of the complete dataset and is thus an $N^c \times N^c$-matrix. To keep the latent space elements not corresponding to this subset unaffected by this error contribution, we can define for each c an index set I^c containing just the examples referenced by P^c and rewrite the loss for that type of similarity as

$$E^c(\boldsymbol{Z}) := \frac{1}{N^c} \sum_{i,j \in I^c} P^c_{ij} \log \left(\frac{P^c_{ij}}{Q^c_{ij}} \right). \tag{4.8}$$

4.3 Experiments

4.3.1 Learning correspondences between image sets

In extending the experiment described in the previous section we trained MRE to discover correspondences between sets of images, in this case with different dimensionalities. We picked 20 successive images from one object of the COIL dataset described above and 28 images (112×92 pixels) depicting a person under different viewing angles taken from the UMIST dataset[Graham and Allinson, 1998]. We chose this data in order to obtain two sets of images that vary in a "similar" or related way. Note that, because the datasets have different dimensionalities, here it is not possible to define a single relation describing Euclidean distance

between all data-points. Instead we constructed two relations P^{Coil} and P^{Umist} (for both we used Eq. (4.1) with σ^2 set as in the previous experiment), with corresponding index-sets I^{Coil} and I^{Umist} containing the indices of the points in each of the two datasets. In addition we constructed one class-membership relation in the same way as before and two identical relations P^1 and P^2 that take the form of a 2×2-matrix filled with entries $\frac{1}{2}$. Each of the corresponding index sets I^1 and I^2 points to two images (one from each dataset) that represent the end points of the rotational degree of freedom, i.e. to the first and the last points if we sort the data according to rotation (see figure 4.3, left plot). These similarity types are used to make sure that the model properly aligns the representations of the two different datasets. Note that the end points constitute the *only* supervision signal; we did not use any additional information about the alignment of the two datasets.

After training a two-dimensional embedding[2], we randomly picked latent representatives of the COIL images and computed reconstructions of corresponding face images using a kernel smoother (i.e. as a linear combination of the face images with coefficients based on latent space distances). In order to factor out variability corresponding to class membership we first multiplied *all* latent representatives by the inverse of R^{class}. (Note that such a strategy will in general blow up the latent space dimensions that do not represent class membership, as the corresponding entries in R^{class} may contain very small values. The kernel smoother consequently requires a very large kernel bandwidth, with the net effect that the latent representation effectively collapses in the dimensions that correspond to class membership – which is exactly what we want.) The reconstructions, depicted in the right plot of figure 4.3, show that the model has captured the common mode of variability.

4.3.2 Supervised feature extraction

To investigate the ability of MRE to perform a form of "supervised feature extraction" we used a dataset of synthetic face images that originally appeared in [Tenenbaum et al., 2000]. The

[2]Training was done using 500 iterations with a setup as in the previous experiment.

Figure 4.3: Face reconstructions by alignment. Left: Side-information in form of two image pairs in correspondence. Right: Reconstructions of face images from randomly chosen cat images.

face images vary according to pose (two degrees of freedom) and according to the position of a lighting source (one degree of freedom). The corresponding low-dimensional parameters are available for each data-point. We computed an embedding with the goal of obtaining features that explicitly correspond to these different kinds of variability in the data.

We labelled a subset of 100 out of the total of 698 data-points with the three mentioned degrees of freedom in the following way: After standardizing the pose and lighting parameters so that they were centered and had unit variance, we constructed three corresponding similarity matrices ($P^{\mathrm{Pose1}}, P^{\mathrm{Pose2}}, P^{\mathrm{Lighting}}$) for a randomly chosen subset of 100 points using Eq. (4.1) and the three low-dimensional parameter sets as input data. In addition we used a fourth similarity relation P^{Ink}, corresponding to overall brightness or "amount of ink", by constructing for each image a corresponding feature equal to the sum of its pixel intensities and then defining the similarity matrix as above. We set the bandwidth parameter σ^2 to 1.0 for all of these similarity-types[3]. In addition we constructed the standard SNE relation P^{Eucl} (defined for all data-points) using Eq. (4.1) with σ^2 set[4] to 100.

We initialized the model as before and trained for 1000 iterations of "minimize" to find an embedding in a four-dimensional space. Figure 4.4 (right plot) shows the learned latent space metrics corresponding to the five similarity-types. Obviously, MRE devotes one dimension

[3]This is certainly not an optimal choice, but we found the solutions to be rather robust against changes in the bandwidth, and this value worked fine.

[4]See previous footnote.

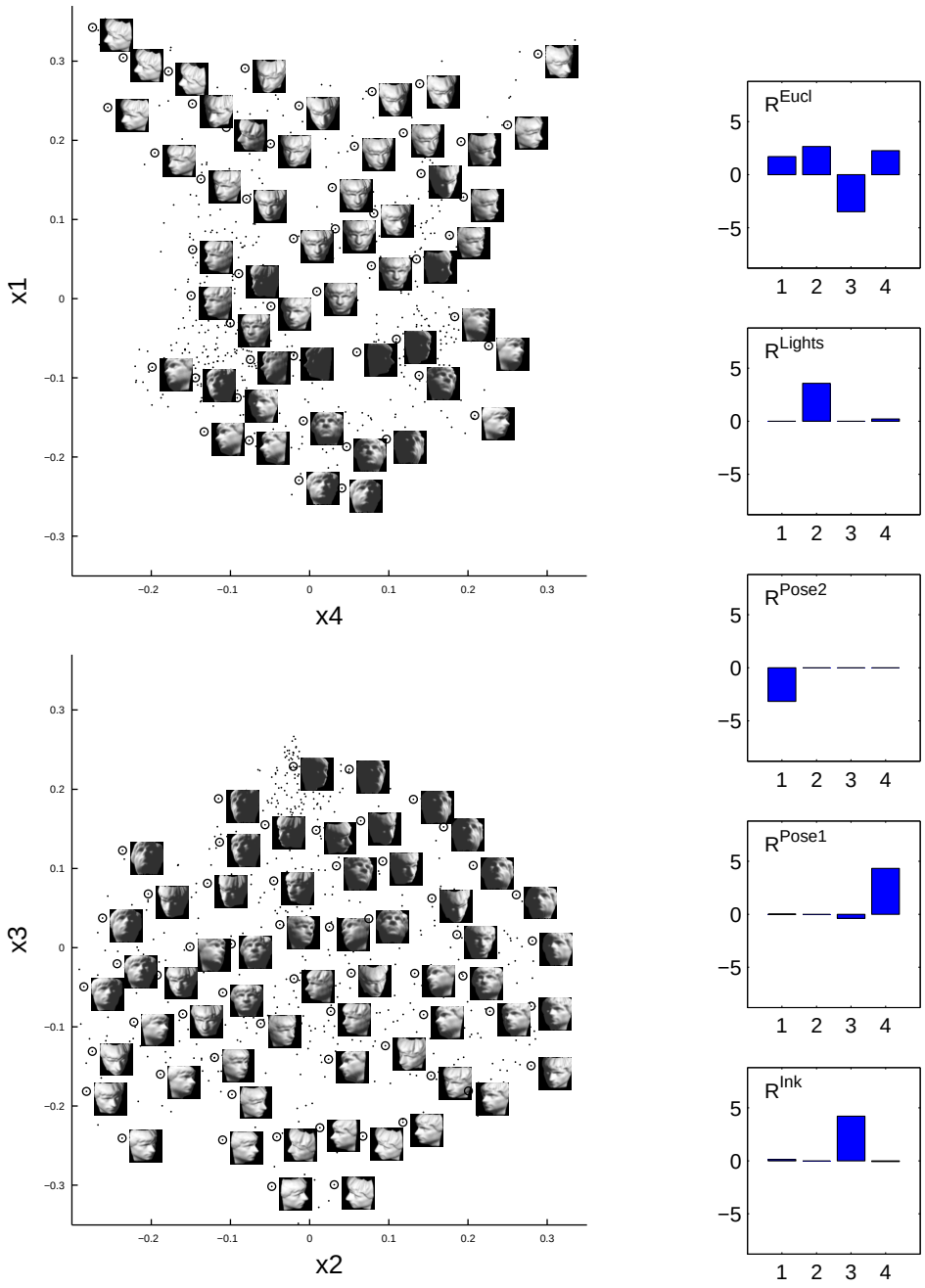

Figure 4.4: Left: Embedding of face images that were not informed about their low-dimensional parameters. For a randomly chosen subset of these (marked with a circle), the original images are shown next to their latent representatives. Right: Entries on the diagonals of five latent space transformations.

to each of the four similarity-types, reflecting the fact that each of them describes a single one-dimensional degree of freedom that is barely correlated with the others. Data-space similarities in contrast are represented using all dimensions. The plots on the left of figure 4.4 show the embedding of the 598 unlabeled data-points. The top plot shows the embedding in the two dimensions in which the two "pose"-metrics take on their maximal values, the bottom plot shows the dimensions in which the "lighting"- and "ink"-metric take on their maximal values. The plots show that MRE generalizes over unlabeled data: In each dimension the unlabeled data is clearly arranged according to the corresponding similarity type, and is arranged rather randomly with respect to other similarity types. There are a few correlations, in particular between the first pose- and the "ink"-parameter, that are inherent in the dataset, i.e. the data does not vary entirely independently with respect to these parameters. These correlations are also reflected in the slightly overlapping latent space weight sets. MRE gets the pose-embedding wrong for a few very dark images that are apparently too far away in the data space to be associated with the correct labeled data-points.

4.4 Discussion

While we extended SNE as an example method, we could inform other types of method, too. If we use a spectral method as a starting point, we could retain some of the computational convenience of spectral methods, for example performing coordinate descent iteratively in the transformations and the latent representatives. However, as we have argued above, non-linear optimization is usually not a problem in embedding problems.

Note that there is a degeneracy in the transformation matrices: The sign of the diagonal transformations does not actually matter in our case. However, it is possible to ignore this degeneracy in the optimization.

Chapter 5

Embedding data transformations

In the second part of the thesis we focus on parametric models. We consider learning about relations in data that are represented more explicitly than in the first part: Input data consists of *pairs of objects*, and our goal is to compute embeddings that represent the dependencies between the objects. We show how conditioning a parametric model can be used for this purpose. As a motivating application we introduce the task of modeling transformations of images.

5.1 Motivation: Learning about image transformations

To motivate the problem we consider the domain of natural images. Similarly as most real world data sets, natural images are not "random": They exhibit a great deal of statistical regularity, both at the level of single pixels and at the level of larger regions. Standard unsupervised learning methods, such as PCA, ICA, and others have become an essential part of the standard toolbox for solving recognition, detection, denoising, and other tasks involving natural images.

In this chapter we consider the related, but also more difficult problem, of discovering structure in the ways images *change*. Typical transformations of images in, say videos, can similarly be highly regular and structured, and it is likely that systems can profit from discov-

ering, and then exploiting this structure as well.

How images can be transformed is intricately related to how images themselves are structured: Natural images, that are composed of edges and junctions, etc., will typically show transformations that re-orient or slightly shift these constituents. This suggests that the statistics of the set of unordered images should be kept in mind, when trying to model image transformations, and that the task of learning about transformations should be tied in with that of learning image statistics.

5.1.1 Encoding transformations with conditional latent variables

To model structure in the way images (or other types of data) change, we develop a probabilistic model of *data transformations* in this chapter. When applied to streams of images, such as videos, the model tries to predict the current (output) image from the previous (input) one. The model takes a form similar to the bi-partite undirected models described in Section 2.2.3. However, in contrast to those models, here the latent variables act as "mapping" units, that can develop efficient codes for the observed *transformations*, rather than encoding static structure. The model learns to encode image data efficiently by developing filters, similarly as a standard bi-partite model that is trained on images. But in this model, the filters are *conditioned* on the previous image in the stream. They are thus functions of the previous image and therefore encode dependency structure. At test time, the model can infer the transformation from a new pair of input-output images as the conditional distribution over the latent mappings units.

To be able to capture all the potential dependencies between the transformation, input, and output units, the three types of unit form three-way cliques in the model. As a result, the task of performing feature *extraction* is tied to the task of feature *mapping*, and both are learned simultaneously.

Once a model for transformations has been trained, there are many potential uses for it. A difficult ongoing problem in pattern recognition is dealing with invariances. We show how

learning about transformations can greatly improve the performance in a recognition task in which the class labels are invariant under these transformations.

5.1.2 Related work

While there has not been much work on *learning* to encode transformations, the idea of using mapping units to encode transformations is not new and dates back at least to [Hinton, 1981], who describe an architecture for modeling simple transformations of letters. However, no learning is performed in the model. A line of research that was inspired by this approach then tried to use mapping units to modulate feature extraction pathways (see [Olshausen, 1994]), however without learning.

Another early, biologically inspired, architecture for modeling transformations of images is described in [Rao and Ballard, 1997]. The model was able to learn some simple synthetic transformations, but no applications were reported.

[Fleet et al., 2000] and [Belongie et al., 2002] constructed systems to model domain-specific transformations (the first for modeling motion, the latter for modeling geometric invariances). Both models were hand-crafted to work in their specific domains, and not trained to perform feature extraction, but they showed some interesting results on real-world tasks. [Jojic and Frey, 2000] describe a generative model of images that contains a latent variable for modelling transformations. The model uses an implicit encoding of transformation in order to obtain a good model of images, rather than learning to extract transformations from image-pairs.

Our model is a type of higher-order Boltzmann machine [Sejnowski, 1986] and can be viewed as a conditional version of a restricted Boltzmann machine (RBM). It therefore bears some resemblances to [He et al., 2004], which used a kind of conditional RBM in a pixel labelling task. In that model, however, the dependence on the inputs is simply in the form of biases for the output units, whereas in our model, input, hidden and output units form three-way cliques, so the effect of an input unit is to modulate the *interaction* between

transformation units and output units. As result, *filters* on the outputs, and not just the outputs themselves, depend on the inputs, which is crucial for the task of learning image transformations.

5.2 Gated Boltzmann machines

In contrast to previous chapters, here we discuss how to modulate *parametric* models using input data. Similarly as the previous chapters, we use probabilistic conditioning as the basis for informing latent variables using extra information.

The basic idea of our model of transformations is to predict the next observation in a stream of observations, and to use *hidden variables* to capture the many possible ways in which the next observation can depend on the previous one.

5.2.1 The model

Similarly as in Section 2.2.3 we introduce the basic model using binary units, which greatly simplifies the exposition. We show later how to deal with more general distributions. To model the probability of an "output" data-point $\boldsymbol{y}$, given an "input" data-point $\boldsymbol{x}$, we consider a score-function that combines all components of input and output points. To explicitly capture the many possible ways in which the outputs can depend on the input, we introduce an additional vector of binary hidden variables $\boldsymbol{z}$.

A simple score function that captures all possible three-way correlations between single components of $\boldsymbol{x}, \boldsymbol{y}$ and $\boldsymbol{z}$ is

$$S(\boldsymbol{y}, \boldsymbol{z}; \boldsymbol{x}) = \sum_{ijk} W_{ijk} x_i y_j z_k, \tag{5.1}$$

where W_{ijk} are the components of a three-way parameter-tensor $\boldsymbol{W}$, that learns from training data to weight the importances of the possible correlations. When learning on images, the components x_i, y_j of $\boldsymbol{x}$ and $\boldsymbol{y}$ can be either pixel intensities or higher-level descriptors such

as the outputs of non-linear filters. The score captures the compatibility between the input, output and hidden units.

Note that, in the way that the energy is defined, each hidden unit z_k can "blend in" a slice $W_{..k}$, of $\boldsymbol{W}$, which defines a linear mapping from inputs to outputs. Therefore, when we *fix* all hidden units, we obtain simply a linear mapping as transformation if the output units are linear. However, in practice we will derive a probability distribution over the set of hidden and output units, and then *marginalize* over all possible mappings as we describe below, which gives rise to highly nonlinear and possibly (if we use more than one hidden unit) compositional mappings.

Using this energy function, we can now define the joint distribution $p(\boldsymbol{z}, \boldsymbol{y}|\boldsymbol{x})$ over outputs and hidden variables by exponentiating and normalizing as we do for the standard models (ref. Section 2.2.3):

$$p(\boldsymbol{y}, \boldsymbol{z}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \exp(S(\boldsymbol{y}, \boldsymbol{z}; \boldsymbol{x})) \tag{5.2}$$

where

$$Z(\boldsymbol{x}) = \sum_{\boldsymbol{y},\boldsymbol{z}} \exp(S(\boldsymbol{y}, \boldsymbol{z}; \boldsymbol{x})) \tag{5.3}$$

is a normalizing constant. Note that, in contrast to previously (Section 2.2.3), the partition function now depends on the input $\boldsymbol{x}$. The reason is that the model is a conditional model, that captures the *dependency* of $\boldsymbol{y}$ on $\boldsymbol{x}$. We do not model $\boldsymbol{x}$ and therefore do not normalize over $\boldsymbol{x}$. In other words, every possible input $\boldsymbol{x}$ gives rise to its own distribution over $\boldsymbol{y}$ and $\boldsymbol{z}$.

To obtain the distribution over outputs, given the input, we marginalize and get:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \sum_{\boldsymbol{z}} p(\boldsymbol{y}, \boldsymbol{z}|\boldsymbol{x}) \tag{5.4}$$

Note that, similarly as in standard RBMs, we cannot actually compute $p(\boldsymbol{y}|\boldsymbol{x})$ or $Z(\boldsymbol{x})$ exactly, since both contain sums over the exponentially large number of all possible hidden unit instantiations (and output unit instantiations, for $Z(\boldsymbol{x})$). In practice, we do not need to compute these quantities to perform inference or learning, as we shall show.

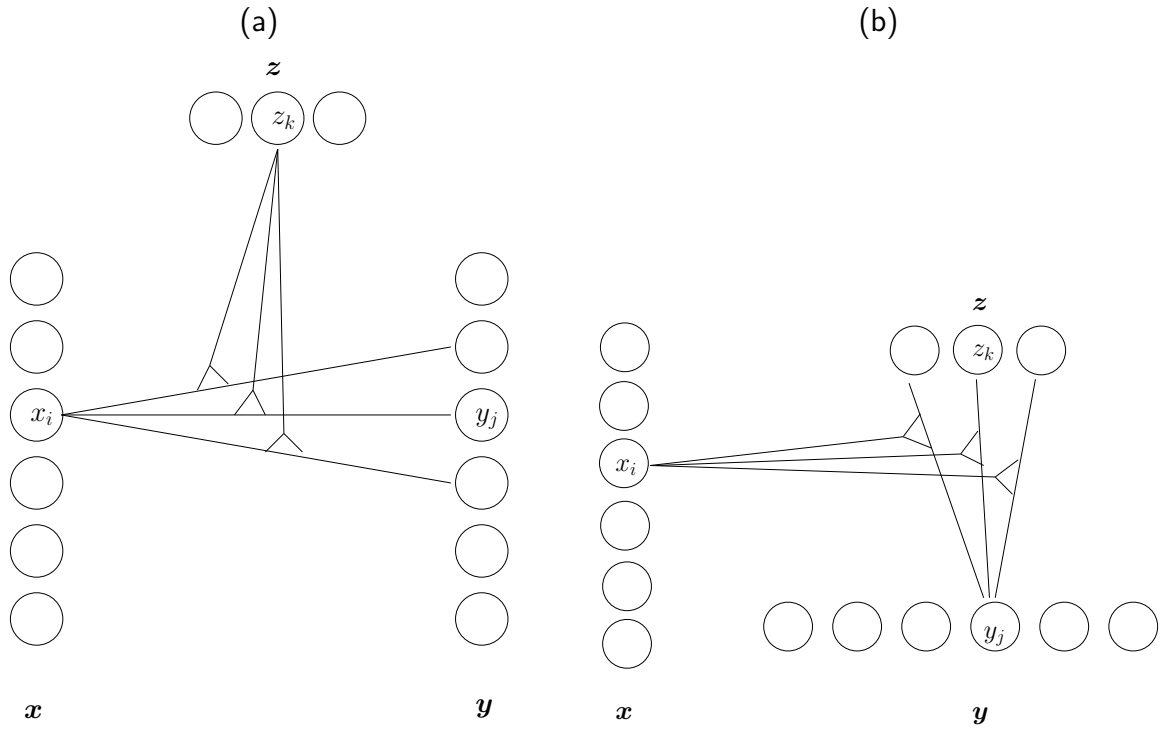

Figure 5.1: Two views of the basic model. (a) Gated regression: Hidden units can add slices $W_{..k}$ into a blend of linear transformations. (b) Modulated filters: Input units gate a set of basis functions that learn to reconstruct the output.

Inference at test time consists of inferring the transformation, or equivalently its encoding $\boldsymbol{z}$, from a *given* pair of observed images $\boldsymbol{x}$ and $\boldsymbol{y}$. But from Eqs. 5.1 and 5.2 it follows easily that

$$p(z_k = 1|\boldsymbol{x}, \boldsymbol{y}) \quad = \quad \frac{1}{1 + \exp(-\sum_{ij} W_{ijk} x_i y_j)} \tag{5.5}$$

for every mapping unit z_k. Note the similarity of Eq. 5.5 with Eq. 2.12. In particular, we retain the convenient independence property. The mapping units are independent binary variables given the input-output pair, and can be computed efficiently. For the distribution over outputs, when input and mapping units are given, we get:

$$p(y_j = 1|\boldsymbol{x}, \boldsymbol{z}) \quad = \quad \frac{1}{1 + \exp(-\sum_{ik} W_{ijk} x_i z_k)} \tag{5.6}$$

Generalized biases

In practice, to be able to model affine and not just linear dependencies, we need to add biases to the output and hidden units. In contrast to unconditional models, where biases are simply additive components that can "shift" the activity level of a unit (ref. Section 2.2.1), in this more general model we can use also *gated biases*, that shift an activity conditionally. A score function that contains both gated biases, as well as standard biases, can take the form:

$$S(\boldsymbol{y}, \boldsymbol{z}; \boldsymbol{x}) = \sum_{ijk} W_{ijk} x_i y_j z_k + \sum_{jk} W_{jk}^{\boldsymbol{yh}} y_j z_k + \sum_{j} W_j^{\boldsymbol{y}} y_j + \sum_{k} W_k^{\boldsymbol{z}} z_k \tag{5.7}$$

Even more general forms of bias are possible in this model. In fact, any set of weights that affects either one or two groups of units (inputs-to-hiddens, inputs-to-outputs, hiddens-to-outputs, just hiddens, or just outputs) can be considered a "generalized bias" in this model. In most of our experiments we have used all possible connections. But the energy function defined in Eq. 5.7 (containing only the single-node biases and the hidden-to-output biases) is often sufficient to get good results.

5.2.2 Learning

To train the probabilistic model, we maximize the average conditional log likelihood of a set of N training pairs $(\boldsymbol{x}^{\boldsymbol{\alpha}}, \boldsymbol{y}^{\boldsymbol{\alpha}})$. We can equivalently minimize the negative log-likelihood to get the objective function:

$$L^{\mathrm{gbm}} = -\frac{1}{N} \sum_{\alpha} \log p(\boldsymbol{y}^{\boldsymbol{\alpha}} | \boldsymbol{x}^{\boldsymbol{\alpha}}) \tag{5.8}$$

We can use gradient based optimization to minimize L^{gbm}. As in the standard RBM, the gradient of the log-likelihood for each training case is the difference of two expectations:

$$\frac{\partial L^{\mathrm{gbm}}}{\partial \boldsymbol{W}} = \sum_{\alpha} \sum_{\boldsymbol{z}} \frac{\partial S(\boldsymbol{y}^{\boldsymbol{\alpha}}, \boldsymbol{z}; \boldsymbol{x}^{\boldsymbol{\alpha}})}{\partial \boldsymbol{W}} - \sum_{\boldsymbol{z}, \boldsymbol{y}} \frac{\partial S(\boldsymbol{y}, \boldsymbol{z}; \boldsymbol{x}^{\boldsymbol{\alpha}})}{\partial \boldsymbol{W}} \tag{5.9}$$

where we let $\boldsymbol{W}$ denote the set of all weights, including any biases, if present. The first expectation is over the posterior distribution over mapping units and can be computed efficiently

using Eq. 5.5. The second is an expectation over all possible output/mapping instantiations and is intractable. Note however that, because of the conditional independences of $\boldsymbol{z}$ given $\boldsymbol{y}$, and $\boldsymbol{y}$ given $\boldsymbol{z}$ (see previous section), we can easily sample from the conditional distributions $p(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y})$ and $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{z})$.

As in standard RBMs, Gibbs sampling therefore suggests itself as a way to approximate the intractable term. And as in Section 2.2.3 we can train using contrastive divergence, that is we can initialize the sampling with the training data and perform only one iteration of sampling, in order to shape the energy-surface only locally. We have used contrastive divergence training in all of the experiments, that we describe below.

5.2.3 Two views

Gated regression

Since the model defines a conditional distribution over outputs, it can be thought of as an autoregressive model. In particular, Eq. 5.4 shows that it is a kind of mixture of experts, with a very large number of mixture components (exponential in the number of mapping units). Unlike a normal mixture model, the exponentially many mixture components share parameters which is what prevents it from overfitting. The number of parameters scales only linearly, not exponentially, with the number of mapping units.

Each binary mapping unit z_k that is active effectively "blends" in a slice $W_{..k}$ of the weight tensor $\boldsymbol{W}$ into the mixture (see Eq. 5.1). The model can therefore *compose* a given transformation from a set of simpler transformations, which is crucial for modeling many real-world transformations. In image transformations, for example, it is possible that different parts of an image transform in different ways. Likewise at test time, the hidden units *decompose* an observed transformation into its basic components by measuring correlations between input- and output-components. The importance that hidden unit z_k attributes to the correlatedness (or anti-correlatedness) of a particular pair x_i, y_j is determined by W_{ijk}, as

can be seen also from Eq. 5.5: If W_{ijk} is positive then a positive correlation between x_i and y_j will tend to excite unit z_k, and a negative correlation tend to inhibit it. Importantly, the task of *defining* the set of basis transformations that is needed for some specific task at hand, is considered to be a domain-specific problem, and is therefore left to be solved by learning.

Modulated filters

An alternative view of the model is shown in figure 5.1 (b): Each given, fixed input $\boldsymbol{x}$ defines a standard RBM. When applied to images, the model captures spatial correlations using *input-dependent filters* $\hat{W}_{jk} = \sum_i W_{ijk} x_i$. The filters are input-weighted sums of slices $W_{i\cdot\cdot}$ of the weight tensor $\boldsymbol{W}$. Folding the inputs into the weights this way lets us rewrite the score in the form of a (case-dependent) RBM as:

$$S(\boldsymbol{y}, \boldsymbol{z}; \boldsymbol{x}) = \sum_{jk} \hat{W}_{jk} y_j z_k + \sum_j W_j^{\boldsymbol{y}} y_j + \sum_k W_k^{\boldsymbol{z}} z_k \tag{5.10}$$

(Note that, as before, we can also add generalized biases to the input-modulated weights, by defining the weights alternatively as: $\hat{W}_{jk} = \sum_i W_{ijk} x_i + W_{jk}^{\boldsymbol{zy}}$)

In other words, the model defines a bipartite conditional random field over output images and mappings. Instead of specifying spatial correlations ahead of time, as would be done for example with a Markov random field, here the possible correlations are learned from training data and can be domain-specific, input-dependent, and possibly long-range.

The fact that the model defines an RBM once the input has been fixed, is convenient for inference and learning, since it allows us to make use of all the available machinery that has been developed for similar tasks in RBMs. Furthermore, approaches for extending RBMs also carry over easily to GBMs (see Section 5.2.5, for example).

5.2.4 Example: Transformations of binary images

As a proof of concept, we trained a model with 20 mapping units on synthetic videos showing random binary images of size 10×10 pixels that are shifted by one pixel in a random direction

in each frame (see figure 5.2, leftmost two columns). To generate the data, we first generated a set of initial images by turning on each pixel in an image randomly with probability 0.1. Then, repeatedly we chose for each image a direction randomly from the set $\{$up, down, left, right, up-left, up-right, down-left, down-right, no shift$\}$, and shifted the images, filling in newly appearing edges randomly, and independently as before with probability 0.1 We trained the model on batches containing 100 image pairs each. Since the image pairs were generated on the fly, training was online, with a gradient update after each presentation of a batch. We trained the model on several thousand batches, which takes a few minutes on a 3.4 GHz Pentium 4. Since training data can be generated on the fly, the amount of available training data is essentially unlimited. (Note, that training on this data set would be much more difficult with non-parametric methods, as discussed in Section 2.4.)

After training, the model can infer "optical flow" from a pair of test-images by computing the hidden unit activations using Eq. 5.5. The model represents the flow that it observes implicitly in these mapping unit activations. But it is possible to visualize an approximation of the models "percept", by drawing for each input-pixel an arrow to the output-pixel to which the input-pixel connects the most according to the (marginalized) weight-tensor W.

Figure 5.2 shows seven example image pairs and the inferred "max-flow"-fields. The model is able to consistently infer the correct motion over the whole patch (except at edges that move into the image, where pixels are random and cannot be predicted).

Given a set of hidden unit activations $\boldsymbol{z}$, we can apply the encoded transformation to a new, previously unseen image $\boldsymbol{x}^{\text{new}}$ by computing $p(\boldsymbol{y}|\boldsymbol{z}, \boldsymbol{x}^{\text{new}})$. The right-most column in the figure depicts these transfered transformations using gray values to represent the probability of turning a pixel "on".

In this example, a mixture of experts, *ie.* a model that contains a single, multinomial mapping unit, would in principle be sufficient, because the set of possible transformations has a small size (9 in this case). Figure 5.3 shows a variation of the experiment, where the transformations are *factorial* instead. The image pairs are generated by transforming the top

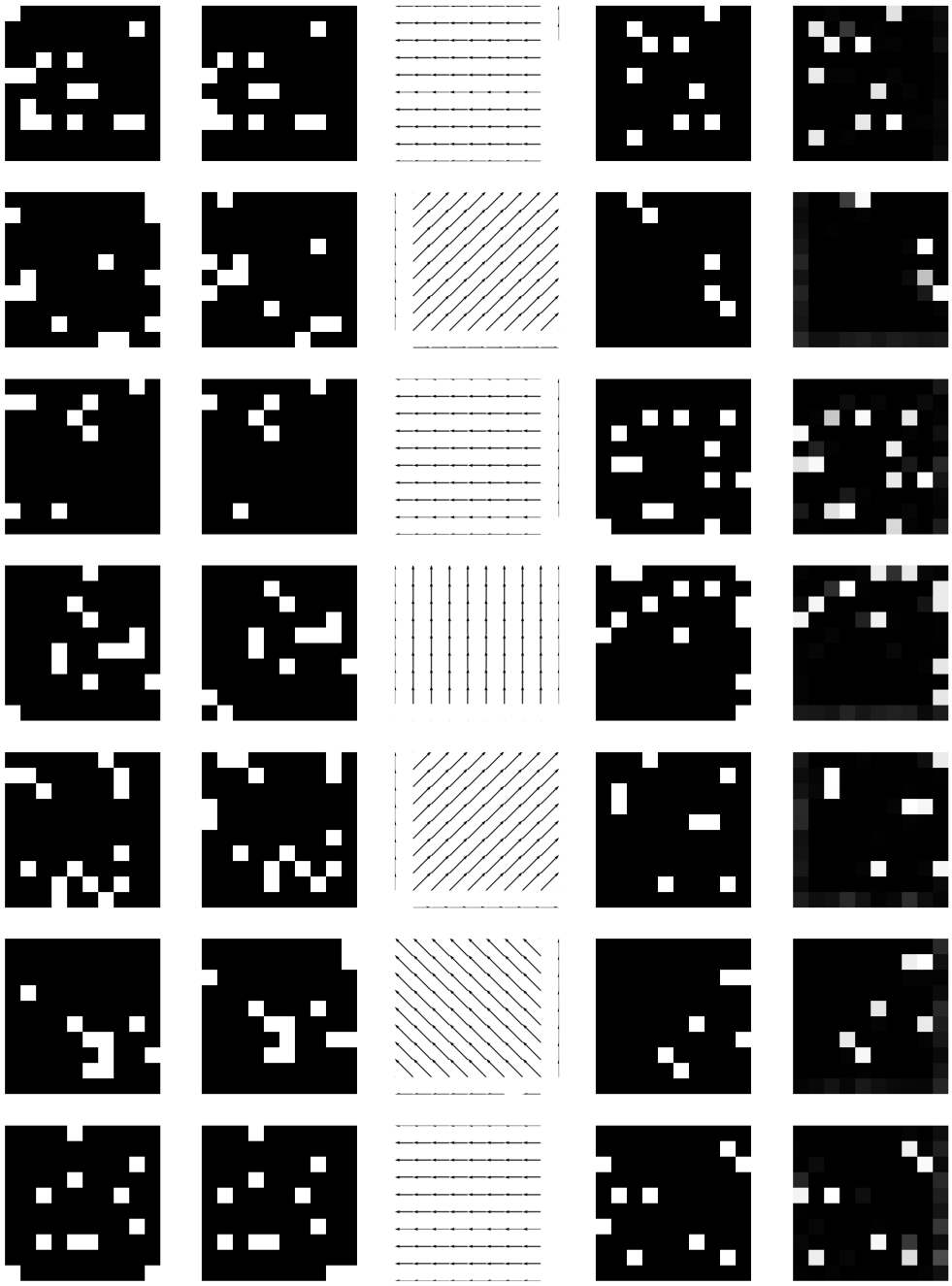

Figure 5.2: Columns (left to right): Input images; output images; inferred flowfields; random target images; inferred transformation applied to target images. For the transformations (last column) gray values represent the probability that a pixel is "on" according to the model, ranging from black for 0 to white for 1.

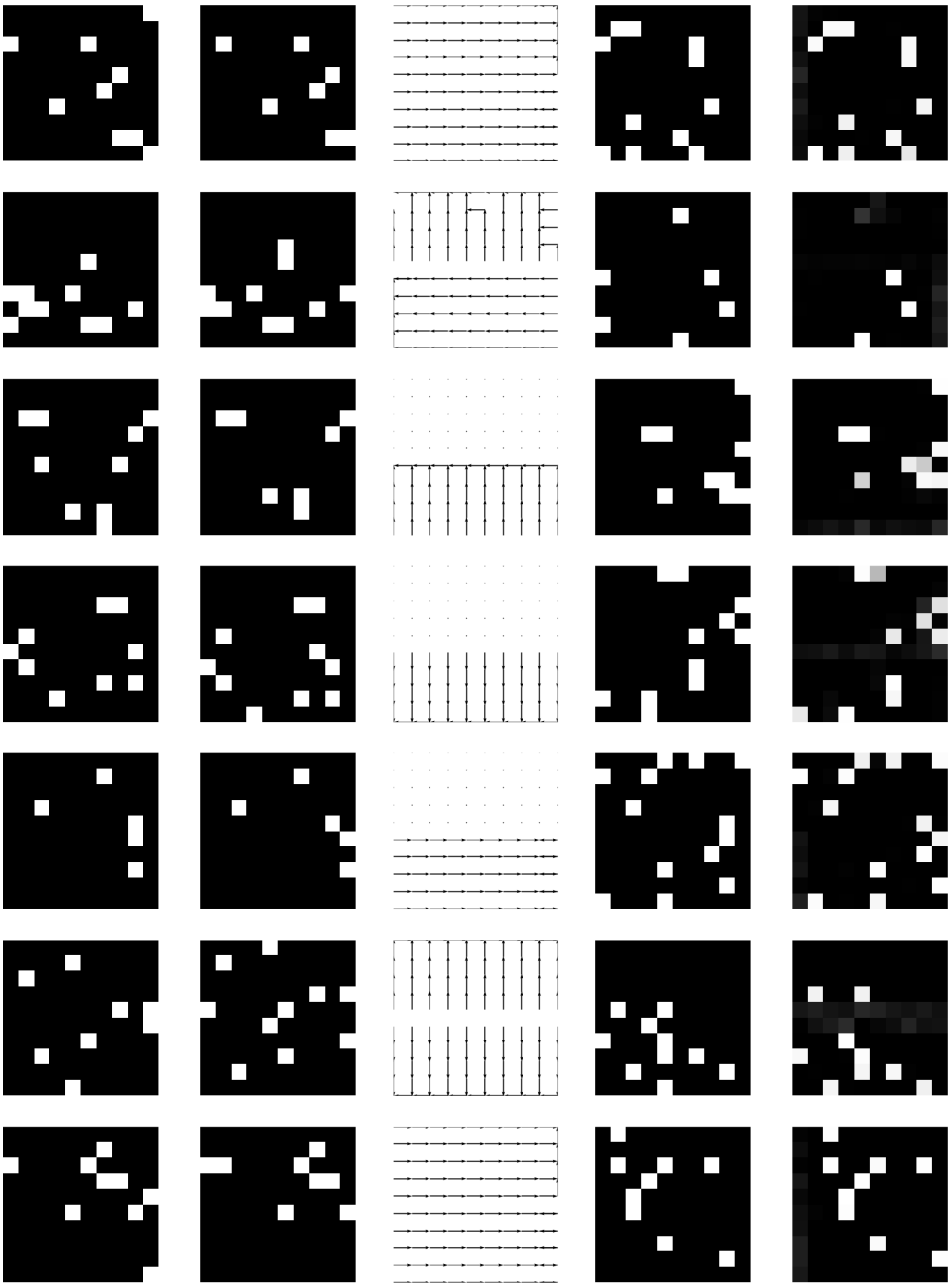

Figure 5.3: Factorial transformations. Display as in figure 5.2.

and bottom halves of the input images independently of each other. We used 5 possible transformations for this experiment (shift left, right, up, down and no shift), and 10 mapping units as before. While 10 mixture components in an ordinary mixture model would not be sufficient to model the resulting $5 \times 5 = 25$ transformations, the GBM finds a factorial code

that can model the transformations, as can be seen from the figure. While in these toy-examples there are no correlations between the pixels in a single image, in natural images we would expect such correlations to provide further cues about the transformations. We revisit this kind of experiment using natural images in Section 5.4.1.

5.2.5 Generalization to non-binary units

The model defined so far defines a binary distribution over output and mapping units. However, similarly is in standard RBMs, binary values are not always the best choice, and it is often desirable to be able to use more general distributions, for either output or mapping units. In particular, continuous values for the outputs can be useful in image modelling tasks, but also other choices (for either outputs or hidden units) are imaginable, such as multinomial, Poisson, or others.

Continuous outputs and high-dimensional regression

Modifying the model in order to deal with continuous outputs, while keeping the hidden units binary, for example, can be achieved straightforwardly, following the same approach that is used for continuous standard RBMs [Hinton and Salakhutdinov, 2006]: We re-define the score as (similarly when using the other kinds of generalized bias):

$$S(\boldsymbol{y}, \boldsymbol{z}; \boldsymbol{x}) = -\frac{1}{2\nu^2} \sum_j (y_j - W_j^{\boldsymbol{y}})^2 + \frac{1}{\nu} \sum_{ijk} W_{ijk} x_i y_j z_k + \sum_k W_k^{\boldsymbol{z}} z_k, \tag{5.11}$$

and then define the joint distribution as before (Eq. 5.2). Now, while the conditional distribution over hidden units $p(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y})$ remains the same (because the additional term cancels out after exponentiating and conditioning), it is straightforward to show that the distribution over outputs turns into a Gaussian (see [Hinton and Salakhutdinov, 2006] for details):

$$p(y_j|\boldsymbol{x}, \boldsymbol{z}) = \mathcal{N}\left(y_j; \nu \sum_{ik} x_i z_k W_{ijk} + W_j^{\boldsymbol{y}}, \nu^2\right) \tag{5.12}$$

We use a spherical Gaussian for the outputs here, but we could also use a different variance ν_j for each output-component.

Since the conditional distribution is a Gaussian, that is independent across components y_j, Gibbs sampling is still straightforward in the model. Note, that the marginal distribution is *not* Gaussian, but a mixture of exponentially many Gaussians. As before, it is intractable to evaluate the marginal, so it is fortunate that it does not need to be evaluated for either learning or inference.

Note also, that the inputs $\boldsymbol{x}$ are always conditioned on in the model. They therefore do not need to be treated differently than for the binary case. Any scaling properties of the inputs can be absorbed into the weights, and are therefore taken care of automatically during training though learning is faster and more stable if the scales are sensible.

Using Gaussian outputs allows us to perform *regression* tasks, in which the outputs are high-dimensional and structured. It therefore constitutes an interesting extension to recent work on classification with structured outputs (see [Lafferty et al., 2001]). In contrast to the recent kernel based attempts in this direction (*eg.* [Teh et al., 2005]) our model can be trained on much larger datasets, and is capable of online learning.

Conditional exponential family harmoniums

While extending the model to deal with Gaussian outputs is useful for many problems, we do not need to stop there. In fact, it is easy to extend the model such that it uses any member of the exponential family as the conditionals $p(\boldsymbol{y}|\boldsymbol{z},\boldsymbol{x})$ or $p(\boldsymbol{z}|\boldsymbol{y},\boldsymbol{x})$. [Welling et al., 2005] describe a framework for constructing RBMs with arbitrary exponential family distributions as the conditionals, and their approach can be applied similarly to GBMs, because once we condition on an input the model simply takes the form of an RBM ("View 2" in Section 5.2.3).

Until now we have represented each state of a unit using a single binary value. The reason is that the sufficient statistics of binary variables are their binary states themselves. The sufficient statistics of more general random variables can be more complicated and can contain more than just one value. (Consider a Gaussian, for example, whose sufficient statistics

are given by its mean and variance.)

We therefore introduce indices a and b and represent the variables y_j and z_k using their sufficient statistics $f_{ja}(y_j)$ and $g_{kb}(z_k)$, respectively. The joint distribution under this representation can then be re-defined as:

$$p(\boldsymbol{y}, \boldsymbol{z}; \boldsymbol{x}) \propto \exp\left(-\sum_{jakb} \hat{W}_{ja}^{kb} f_{ja}(y_j) g_{kb}(z_k) - \sum_{ja} W_{ja}^{\boldsymbol{y}} f_{ja}(y_j) - \sum_{kb} W_{kb}^{\boldsymbol{z}} g_{kb}(z_k) \right), \quad (5.13)$$

where $\hat{W}_{ja}^{kb}$ are the modulated weights that we obtain similarly as before, by "folding" the inputs into case-independent weights: $\hat{W}_{ja}^{kh} = \sum_i W_{ijakb} x_i$.

Eq. 5.13 is again simply a standard (but case-dependent) RBM. As before, and as in a standard RBM, the conditional distributions under this joint distribution decouple into products of independent distributions, that now are exponential family distributions themselves, and whose parameters are just "shifted" versions of the biases:

$$p(y_j|\boldsymbol{z}; \boldsymbol{x}) \propto \exp\left(\sum_a \left[\sum_{kb} \hat{W}_{ja}^{kb} g_{kb}(z_k) + W_{ja}^{\boldsymbol{y}} \right] f_{ja}(y_j) \right) \quad (5.14)$$

$$p(z_k|\boldsymbol{y}; \boldsymbol{x}) \propto \exp\left(\sum_b \left[\sum_{ja} \hat{W}_{ja}^{kb} f_{ja}(y_j) + W_{kb}^{\boldsymbol{z}} \right] g_{kb}(z_k) \right) \quad (5.15)$$

Note that we let the inputs modulate the couplings as before, instead of modulating the sufficient statistics. Fixing the sufficient statistics has the advantage that it keeps learning and inference simple. Since the conditionals decouple as previously for the binary case, Gibbs sampling is still straightforward. In the practical implementation, basically all that changes for training the model is the routines for sampling from the conditionals.

5.3 Extensions

5.3.1 Fields of Gated Experts

Until now we have considered a single global model, that connects each input component with each output component.

When modelling image data, this can be problematic for two reasons. First, for images that are large, connecting all components (pixels) is not going to be tractable. The most simple solution for natural video data is to restrict the connections to be local, since the transformations on these data-sets are typically not very long-range.

Secondly, however, in many real-world tasks (albeit not in all) it is often the case that the kind of transformations that occur in one part of the image could occur in principle also in any other. Besides restricting connections to be local, it can therefore makes sense to also use some kind of weight-sharing, so that we apply essentially the same *model* all over the image, though not necessarily the same particular transformation all over the image.

In image modelling tasks, we can define a single patch-model, as before, but define the distribution over the whole output-image as the *product* of distributions over patches centered at each output-pixel. Each patch (centered at some output-pixel y_j) contains its own set of hidden units z_k^j. Formally, we simply re-define the score to be (we drop the biases here for simplicity):

$$S(\boldsymbol{y}, \boldsymbol{z}; \boldsymbol{x}) = \sum_s \sum_{ijk} W_{ijk} x_i^s y_j^s h_k^s, \tag{5.16}$$

where s ranges over all sites, and x_i^s denotes the (i^{th}) component of $\boldsymbol{x}$ in site s (analogously for $\boldsymbol{y}$ and $\boldsymbol{z}$). Inferring the hidden unit probabilities at some site s, given the data, can be performed exactly the same way as before independently of the other sites, using Eq. 5.5. When inferring the data distribution at some site s, given the hiddens, some care needs to be taken because of overlapping output patches. Learning can be performed the same way as before using contrastive divergence.

This approach is the direct conditional analogue to modeling a whole image using patch-wise RBMs as used in [Roth and Black, 2005a], [Roth and Black, 2005b] for the non-conditional case. A similar approach for conditional models has also been used by [He et al., 2004] in a discriminative setting. The analog of the discriminative product model for sequence-structured data is described in [Stewart, 2005].

5.3.2 Neural network premappings

The fact that the model defines a conditional (and not joint) distribution over outputs makes it easy to develop extensions, where the model learns to pre-process the inputs prior to transforming them.

Note that we can re-interpret the energy-function (Eq. 5.1) as follows: We start by using a standard RBM to model the output data $\boldsymbol{y}$. We want to introduce side-information in order to compute transformations rather than static structure in the data. So the side-information consists of the original data-points $\boldsymbol{x}$ in this case.

Conditioning a *parametric* model on $\boldsymbol{x}$ can be achieved by making the *model parameters* functions of $\boldsymbol{x}$. Since RBMs are bi-partite models, doing so is particularly simple for these. The parameters W_{jk} of the RBM (ref. Section 2.2.3) turn into functions $W_{jk}(\boldsymbol{x})$. The most simple form of dependency that we can use is linear, in which case we obtain:

$$W_{jk}(\boldsymbol{x}) = \sum_i W_{ijk} x_i \tag{5.17}$$

Now, plugging these weights into the standard RBM score function (Eq. 2.9) gives us the three-way score function in Eq. 5.1.

This interpretation of the GBM energy function immediately suggests a variation, where instead of using a linear dependency, we use a non-linear one. That is, can set $W_{jk} = f_{jk}(\boldsymbol{x})$, where each function $f_{jk}()$ can be some flexibel non-linear model, such as a neural network. Using a separate function for each RBM weight would amount to using dq functions, where d is the number of output units and q the number of mapping units, and could mean a prohibitively large number of parameters, in particular, if the functions $f_{jk}()$ contain many parameters themselves. A possible compromise is to model each weight W_{jk} using a flexible model, such as a neural network, but to let the models *share parameters*.

We can, for example, use a neural network with a linear output-layer, and a shared hidden layer. This is equivalent to applying a *premapping* to the inputs, which extracts non-linear features, and modulating an RBM using this feature-representation. That is, we can define a

set of feature functions $\phi_i(\boldsymbol{x})$, and re-define the score (Eq. 5.1) as:

$$S(\boldsymbol{y}, \boldsymbol{z}; \boldsymbol{x}) = \sum_{ijk} W_{ijk} \phi_i(\boldsymbol{x}) y_j z_k \tag{5.18}$$

(similarly for the biases). Each RBM weight now takes the form $W_{jk}(\boldsymbol{x}) = \sum_i W_{ijk} \phi_i(\boldsymbol{x})$.

Training the whole model consists in adapting both the mapping parameters $\boldsymbol{W}$ and the parameters of the feature functions simultaneously. If we use any differentiable functions for the features, we can use back-propagation to compute the gradients [LeCun et al., 1998] [Rumelhart et al., 1986].

Pre-processing the input in this way can have several advantages. One practical advantage is that we can use dimensionality reduction to reduce the computational complexity of the model. Another is that good feature extraction can improve generalization accuracy. In contrast to using the responses of fixed basis functions, such as PCA features, training the whole architecture at once allows us to extract "mappable features" that are optimized for the subsequent transformation with the GBM.

5.4 Experiments

5.4.1 The transformation-fields of natural videos

Recall (Eq. 5.5 for binary units, Eq. 5.15 for more general units) that mapping unit probabilities are inferred by measuring correlations between input- and output-components. The mapping units act like Reichardt detectors [Reichardt, 1969], which are simple neural circuits for detecting motion. In contrast to simple standard Reichardt detectors the mapping units make use of *spatial pooling*: Each detector (mapping unit) connects a set of input units with a set of output units, and can therefore generalize over larger image regions and deal with the presence of noise. The particular correlation patterns between input pixels and output pixels that are prevalent in a certain domain are encoded in the weight tensor $\boldsymbol{W}$. Most importantly, these patterns, and thereby also the distribution of responsibilities across mapping units, are

not assumed to be known, but are *learned* from training data.

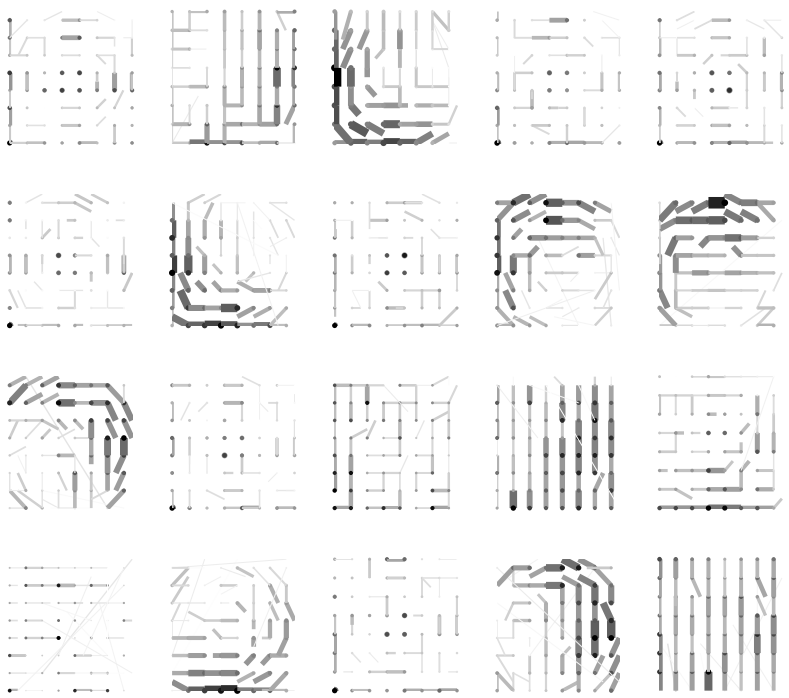

Figure 5.4: 20 basis flowfields learned from synthetic image transformations. The flowfields show, for each input pixel, the strengths of its positive outgoing connections, using lines whose thicknesses are proportional to the strengths.

To see which forms these "Reichardt pools" take on when trained on real world data, we used a database of digitized television broadcasts [van Hateren and Ruderman, 1998]. The original database contains monochrome videos with a frame-size of 128×128 pixels, and a frame rate of 25 frames per second. We reduced the frame-rate by a factor of 2 in our experiments, *ie.* we used only every other frame. Further details about the database can be found in [van Hateren and Ruderman, 1998].

Learning synthetic transformations: First, to see whether the model is able to discover very simple transformations, we trained it on synthetically generated transformations of the images. We took random-patches from the video-database described above (without considering temporal information) and generated sequences by transforming the images with

shifts and rotations. We used 20 mapping units and trained on images of size 8×8 pixels. We used the pixel intensities themselves (no feature extraction) for training, but smoothed the images with a Gaussian filter prior to learning.

Figure 5.4.1 displays resulting "excitatory basis-flow-fields" for several mapping units, by showing for each mapping unit z_k the strength of the *positive* connections W_{ijk}, between pixel i and pixel j, using a line whose thickness is proportional to the connection strength. We obtained a similar plot (but poled in the opposite direction) for negative connections, which means that the model learns to locally shift edges. (See below for further results on this.)

Furthermore, the figure shows that the model infers locality, *ie.* input pixels are connected mostly to nearby pixels. The model decomposes the observed transformation across several mapping units. (Note that this display loses information, and is used mainly to illustrate the resulting flow-fields.)

Broadcast videos: To train the model on the actual videos we cut out pairs of patches of size 20×20 pixels at the same random positions from adjacent frames. We then trained the model to predict the second patch in each pair from the first, as described previously. To speed up training, we used 100 PCA-coefficients instead of the original pixel-intensities as the patch-representations $\boldsymbol{x}$ and $\boldsymbol{y}$, reducing the input-dimensionality from 400 to 100. We used no other forms of preprocessing in these experiments. (It is possible to obtain similar results with the original, pixel-wise, image representations instead of PCA-coefficients, in which case Gaussian smoothing and/or whitening can help the learning.)

A way of visualizing the learned basis flowfields is shown in figure 5. The figure shows that the model has developed sets of local, conditional edge-filters. That is, input pixels only have significant weights to output pixels that are nearby and the weights form an oriented edge filter. The fact that the model decides to latch onto edges to infer information about the observed transformation is not surprising, given that image gradient information is essential for inferring information about optical flow. Note however, that this information is learned

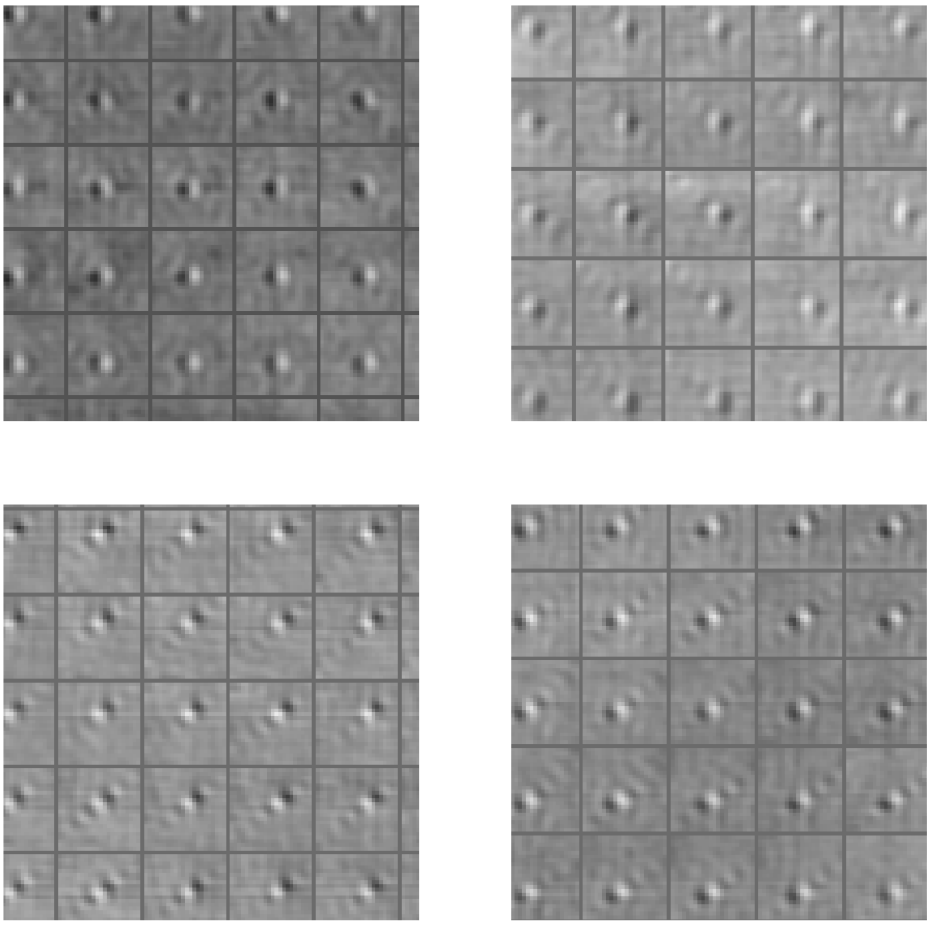

Figure 5.5: A visualization of parts of the learned basis flowfields W_{ijk} for four different hidden units, z_k. For each hidden unit, the input pixels in a small patch are laid out in a coarse grid. At each grid location, there is an intensity image that depicts the weights from that input pixel to all of the output pixels. The intensities range from black for strong negative weights to white for strong positive ones. We invert the PCA encoding before showing the results.

from the database and not handcoded. No sparsity constraints were required to obtain these results. The database – being based on broadcast television – shows many small motions and

camera-shifts, and contains motion as a predominant mode of variability.

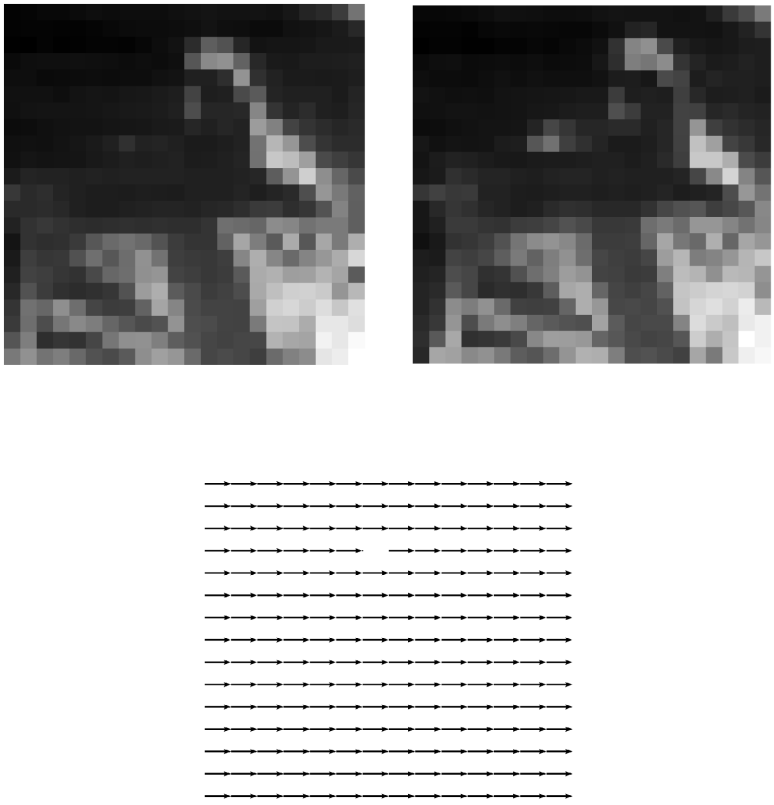

Figure 5.6: Typical image patch pair from the video database, and the inferred flow field. Spatial pooling facilitates generalization and noise suppression.

More interesting than the learned motion edge-features is the fact that adjacent input-pixels tend to be mapped similarly by given mapping units, which shows that the model has learned to represent mostly global motion within the 20×20-patch, and to use spatial pooling to infer information about the observed transformation.

Given the learned model, it is straightforward to generate dense flow-fields as in Section 5.2.4. The plots in the top row of figure 5.6 show a pair of two adjacent example time-frames cropped randomly from the video-database, and showing a more or less even right-shift over the whole patch. To generate the flow-field, as before, at each input-pixel position in the center region of the image (we left out a frame of 4 pixels width, where the field cannot be

inferred precisely) an arrow is drawn that shows to which output-pixel it connects the most[1] (bottom row of the figure). The resulting flow-field shows that the model infers that there was a more or less global motion within the patch, even though corresponding pixel intensities vary considerably between frames, and there are large homogeneous regions. The reason that the model infers a global motion is that it considers it to be the most probable, given the observed evidence, and that such motions are typical in the dataset, whose log-probability is what is being optimized during training.

Note that we do not necessarily advocate using this method to infer dense flow-fields such as the one shown. Rather, the flow-information is represented implicitly here in the form of hidden unit probabilities, and condensing it to the max-flow field would usually mean that potentially useful information gets lost. Also, the kinds of transformations that the model learns are not restricted to motion (as we show below), and are therefore not necessarily local as in the example. To make use of the implicitly encoded information one can use *learning* on the hidden units instead (As done in [Fleet et al., 2000], for example).

5.4.2 Learning an invariant metric

Once a model has been trained to recognize and encode transformations, it is possible to perform recognition tasks that are *invariant* under those transformations, by defining a corresponding invariant metric *w.r.t.* the model. Since the GBM model is trained entirely from data, there is no need to provide any knowledge about the possible transformations to define a metric. Similarly as the model in Section 3.2, GBMs allows us to use *training data* to specify invariances. This is in contrast to many previous approaches that hard-code invariances into an architecture [Chopra et al., 2005].

An obvious way of computing the distance between two images, given a trained GBM

[1]Note that the restriction to integer flow here is not a restriction of the model per se – in particular, since the described model was trained on basis-transformed patches. The flow-field is used mainly for illustration purposes as described in the main text. The full flow representation is given by the mapping unit activations themselves.

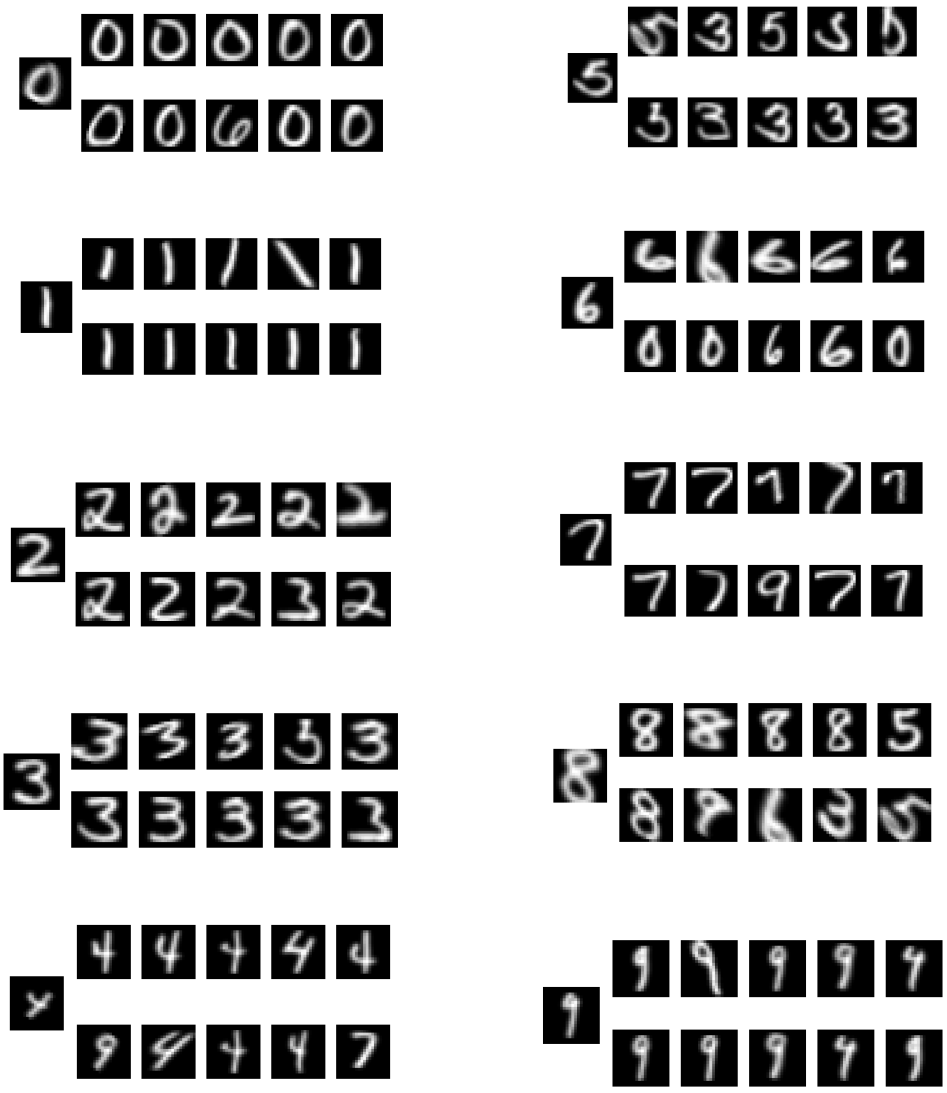

Figure 5.7: Nearest neighbors according to two different metrics. In each block, the left-most digit is a query-case, the digits in the top row are the five nearest neighbors according to the learned model, and the digits in the bottom row are five the nearest neighbors according to Euclidean distance.

model, is by measuring how well the model can transform one image into the other. If it does a good job at modelling the transformations that occur in the distribution of image pairs that the data was drawn from, then we expect the resulting metric to be a good one.

A very simple, but as it turns out, very effective, way of measuring how well the model can

transform an input image $\boldsymbol{x}$ into another image $\boldsymbol{y}$, is by first inferring the transformation, then applying it, and finally using Euclidean distance to determine how well the model did. Formally this amounts to using the following three-step algorithm:

1: Set $\hat{\boldsymbol{z}} = \arg\max_{\boldsymbol{z}} p(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y})$

2: Set $\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}, \hat{\boldsymbol{z}})$

3: Define $d(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|$,

where $d(\cdot, \cdot)$ is the resulting distance measure, which is strictly speaking not a distance since it is not symmetric. (It could easily be turned into a proper distance by adding the two opposite non-symmetric versions, but in many applications symmetry is not actually needed.) Note that both operations can be performed efficiently because of the independence of the conditional distributions.

It is interesting to note that we can interpret the procedure also as measuring how well the model can reconstruct an output $\boldsymbol{y}$ under the conditional distribution defined by the clamped input $\boldsymbol{x}$, using a one-step reconstruction similar to the one used during contrastive divergence learning. Points that lie in low-density (and correspondingly high-energy) regions tend to get "pulled" towards higher density regions more strongly than points that already reside in high density regions, and therefore experience a larger shift in terms of Euclidean distance.

Figure 5.7 shows the nearest neighbors for a few test-cases from a digit dataset (see next section for a detailed description of the dataset) among three hundred randomly chosen training cases according to a GBM trained to predict affine transformations of the digits. We used a model with 30 mapping units and with a linear one-layer neural network premapping (see Section 5.3.2) to reduce the input dimensionality to 30 in order to speed up training. Both the model parameters and parameters of the pre-mapping were trained simultaneously. The premapping was trained by backpropagating the derivatives produced by contrastive divergence. The derivatives that are used to learn the bias of a unit can always be used to learn how this bias should depend on some other, given input. After training, we picked query-cases $\boldsymbol{y}$ from a test-set (that was not seen during training) and determined the five

nearest neighbors among 300 randomaly chosen training cases using (i) the metric induced by the model and using (ii) Euclidean distance. The figure shows that, in contrast to Euclidean distance, the distance induced by the model cares little about pixel overlap and considers digits to be similar, if they can be transformed into each other.

Application to digit classification

In the task of digit classification, viewpoint invariance is usually not a central concern because typical datasets are composed of normalized images. This allows nearest neighbor classification in pixel space to obtain reasonably good results.

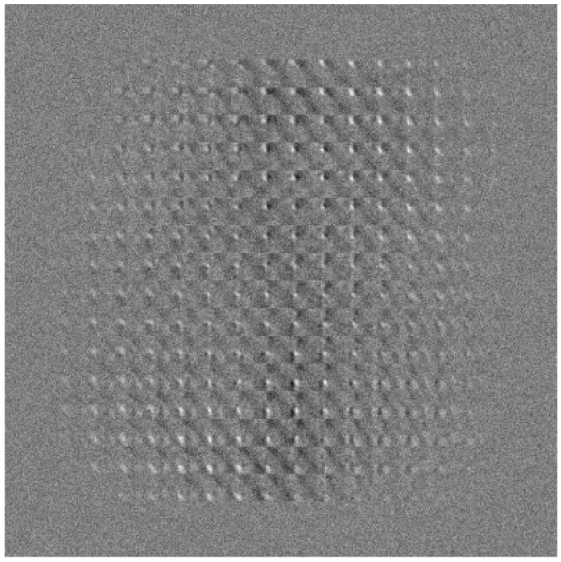

Figure 5.8: A transformation field learned from affine digits. Connection strengths are represented as in figure 5.5

Here we modify the digit classification task by taking 5000 randomly chosen examples from the USPS-dataset (500 from each class) and generating 15000 extra examples using random affine transformations (3 for each case).

Predicting transformed digits from their originals requires quite different transformation fields than those required for predicting video images. Figure 5.8 shows a typical transfor-

mation field after training a GBM with 50 mapping units on the raw pixel images. One obvious feature is that the model is indifferent with regard to the edges of the image (where it encounters much less variability than in the middle).

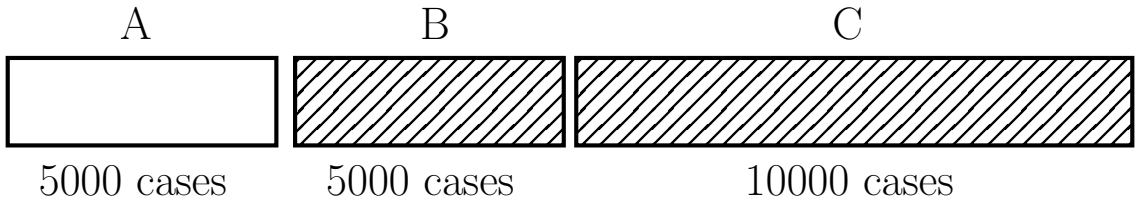

Figure 5.9: Illustration of the dataset used in the k-nearest neighbors classification experiment.

We now consider the problem of predicting 10000 of the transformed digits from the 5000 original training points (for which we have labels available). We partition the data-set as illustrated in figure 5.9 in order to obtain two different experimental settings. The data-set labelled 'A' in the figure represents the 5000 original digits. Data-sets 'B' and 'C' contain the (overall 15000) transformed digits. We use data-set 'C' as the test-set for classification.

To learn a metric we first train the GBM to transform digits from data-set 'A' into the corresponding (transformed) digits in data-set 'B'. For these experiments we trained a GBM with 50 mapping units on 100 PCA-features of the digits.. In one setting of the classification experiment, we then use only the original digits (data-set 'A') as the template set for classification. In the other setting we include both the original digits (data-set 'A') and the transformations which we also used to train the GBM (data-set 'B') in the template set. The first problem is one of transfer learning: We are given labels for the unmodified training cases, but the actual task is to classify cases from a different, but *related*, test set. The relation between the sets is provided only implicitly, in the form of correspondences between digits and their transformed versions.

Figure 5.10 compares the nearest neighbor error rates on the 10000 test cases, obtained using Euclidean distance vs. the distance computed using the model of transformations. To compute nearest neighbors using the non-symmetric distance measure, we let the training

cases be the inputs $\boldsymbol{x}$ in the three-step algorithm (previous section). In other words, we measure how well the model can transform prototypes (training cases) into the query cases.

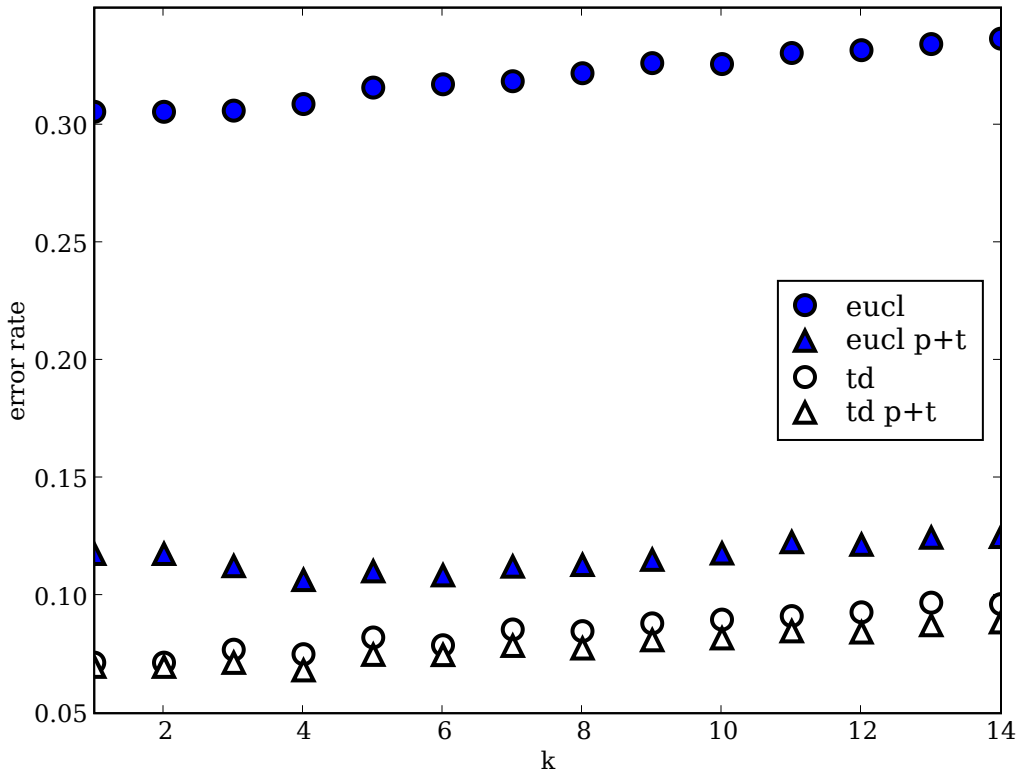

Figure 5.10: Error rates using k-nearest neighbors. The model induced distance is compared with Euclidean distance for two different training sets. Adding affine transformed digits to the training set helps the Euclidean distance a lot, but it is of very little help to the learned distance because the model can learn about transformations from the basic training set.

In task (i) ("eucl" vs. "td" in the plot) Euclidean distance[2] fails miserably, when transformed digits have to be predicted from the original dataset, while the transformation metric does quite well. In task (ii) ("eucl p+t" vs. "td p+t" in the plot) where transformed versions

[2]For the Euclidean distance results, we also compared with Euclidean distance in the 100-dimensional PCA-coefficient space and to normalized data, but Euclidean distance in the original data-space gives consistently the best results.

are included in the training set, Euclidean distance improves significantly, but still does considerably worse than the transformation metric. The transformation metric itself gains only little from including the transformations. The reason is, that most of the available information resides in the way that the digits can be transformed, and has therefore already been captured by the model. Including transformed digits in the training set does not, therefore, provide a significant advantage and leaving them out does not entail a significant disadvantage.

5.4.3 Image transformations

Once we have a way of encoding transformations, we can also apply the these transformation to previously unseen images. If we obtain the transformation from some "source" image pair and apply it to a target image, we are basically performing an analogy. [Hertzmann et al., 2001] discuss a model that performs these general kinds of "image analogies" using a regression-type architecture. An interesting question is, whether it is possible to perform similar kinds of transformations using a general purpose generative model of transformations.

We used a source image-pair as shown in the top row in figure 5.11, generated by taking a publicly available image from the database described in [Grigorescu et al., 2003], and applying an "artistic" filter that produces a canvas-like effect on the image. The image sizes are 512×512 pixels. We trained a field of gated experts model with 10 hidden units on the raw images. The target image is another image from the same database and is shown in the row below, along with the transformation, which we obtained by performing a few hundred Gibbs iterations on the output (the one-step reconstruction works, too, but more iterations improve the results a little). The blow-up shows that the model has learned to apply a somewhat regular, canvas-like structure to the output-image, as observed in the source image.

5.5 Conclusions

There has been surprisingly little work on the problem of learning explicit encodings of data transformations, and one aim of this chapter is to draw attention to this approach which, we believe, has many potential applications.

There are several further interesting directions for future work. One is to learn mappings of multiple different types of feature simultaneously. Doing this at multiple scales could be interesting especially for motion-analysis. Potential applications (other than dealing with invariances for discrimination) are video compression and denoising using the temporal structure.

It would also be interesting to investigate further the relationship between learning of transformations and the ability to make analogies. Analogy making can be argued to be among the key abilities underlying human problem solving skills and creativity [Hofstadter, 1984].

Another interesting problem is the construction of layered architectures in which mapping units are shared between two adjacent layers. This allows transformations that are inferred from pixel intensities to be used to guide feature-extraction, because features of an image that are easy to predict from features of the previous image will be preferred. While we have considered image transformations in this chapter, the model can easily be trained on more general data types than images

An interesting question is whether systematic three-way interactions could be used by biological systems when solving tasks such as motion analysis or stereo vision. Interestingly, ([Anzai et al., 1999a], [Anzai et al., 1999b]) present evidence that multiplicative connections are involved in binocular visual processing in the cat's striate cortex. Multiplicative connections in general have been argued to play an essential role in biological systems for solving a variety of computational tasks (see, for example, [Schmitt, 2002] for a review). Some of the interesting open questions are whether there is evidence for three-way interactions in motion processing, and how learning in biological system could be related to learning in the model.

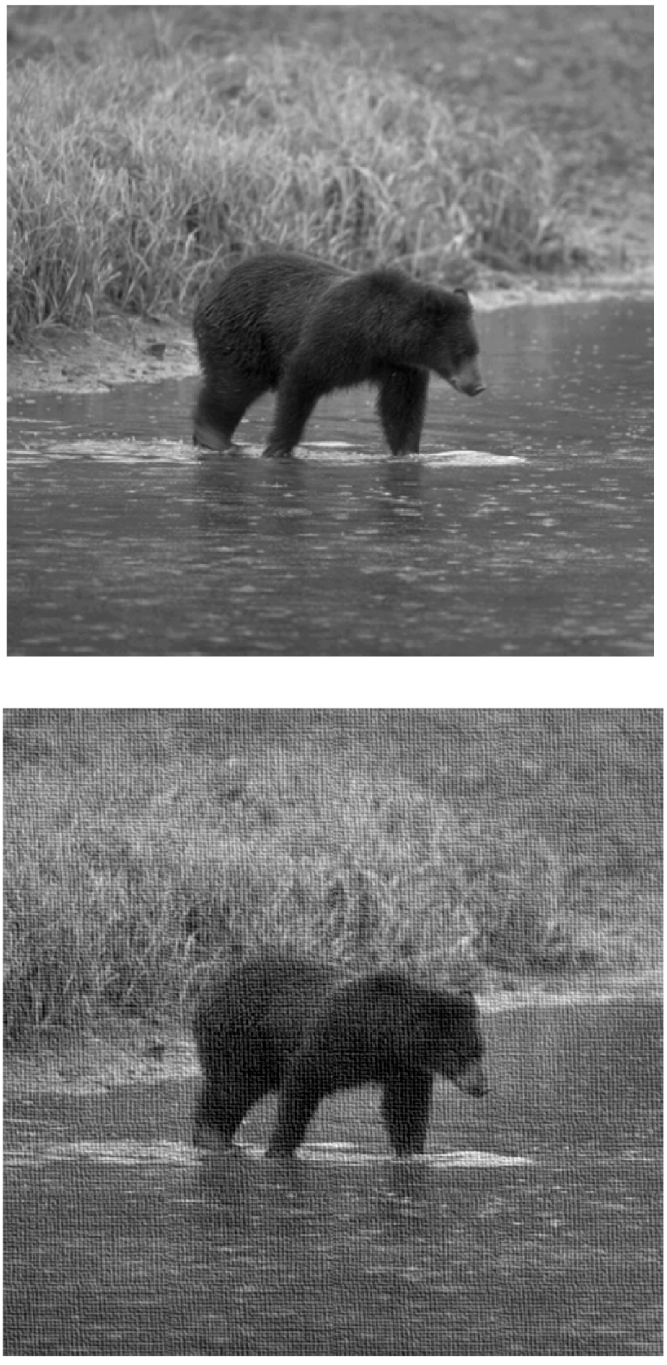

Figure 5.11: Image analogy (source image pair). Source image (top) and its transformation showing a canvas-like effect (bottom).

Figure 5.12: Image analogy (target image pair). Target image (top) and the learned transformation applied to the target image.

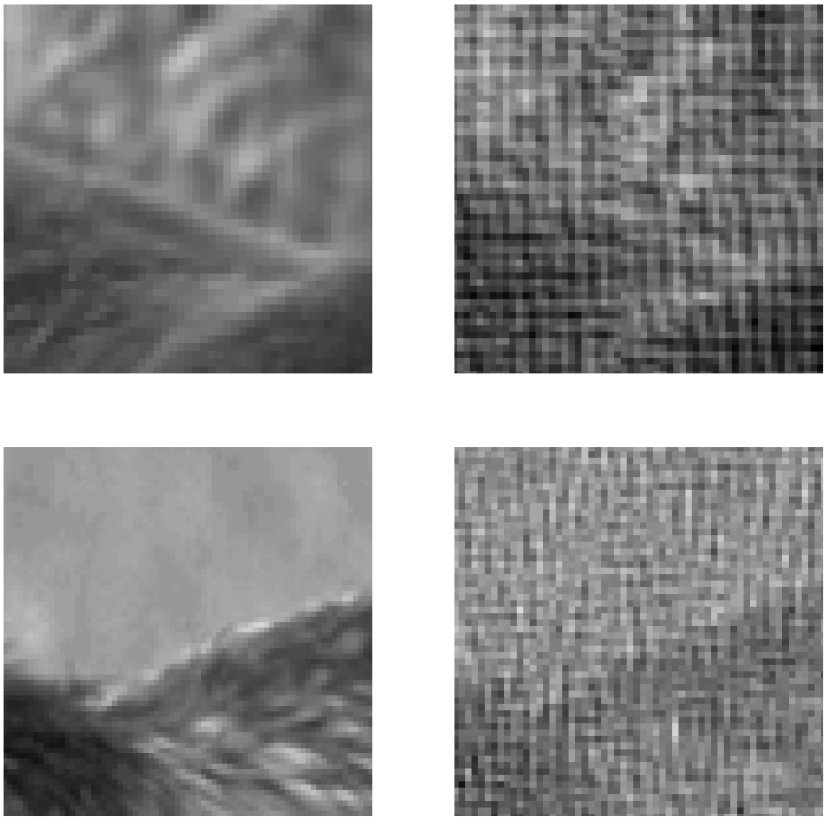

Figure 5.13: Image analogy blow-ups. Top row: Small patch from the source image and its transformation. Bottom row: Small patch from the target image and its transformation.

Chapter 6

Structure prediction by learning a conditional manifold

In the previous chapter we introduced gated Boltzmann machines for learning probabilistic embeddings conditionally. In this chapter we discuss an approach to learning a conditional manifold in a non-probabilistic way. In the same way that an auto-encoder can be viewed as a deterministic analogue of a restricted Boltzmann machine, these models can be thought of as deterministic analogues of the gated Boltzmann machine models introduced in the previous chapter. The previous section discussed learning data transformations; in this chapter we focus on making structured, high-dimensional predictions.

6.1 Structure prediction

In Section 2.6 we discussed the task of supervised learning with structured outputs: We are given a set of pairs $(\boldsymbol{x}^\alpha, \boldsymbol{y}^\alpha)$, and we want to learn a mapping $\boldsymbol{x} \rightarrow \boldsymbol{y}$ that we can apply to previously unseen cases. If the outputs $\boldsymbol{y}$ are highly structured high-dimensional vectors, standard supervised learning methods do not work well, as discussed in Section 2.6.

Recall, that gated Boltzmann machines learn to encode *transformations* inherent in point-

pairs $(\boldsymbol{x}, \boldsymbol{y})$. We described in Section 5 how we can *infer* a transformation from a test-pair and subsequently *apply* it to a new input point in order to transfer it to a test-case, effectively performing an analogy. We demonstrated this idea on images. If we call source data-pairs in an analogy making task the "training data" and the target-inputs "test data", we can interpret the analogy-making task also as a supervised prediction problem, and vice versa. In supervised learning, we usually have multiple input-output pairs to infer the functional relation.

Note, that the GBM defines an intractable probability distribution over outputs given a test-input $\boldsymbol{x}$ (by marginalizing over $\boldsymbol{z}$). So when applied in a supervised learning task, it is not always clear, how to provide the correct "answer" $\boldsymbol{y}$ for a given test-input. [He et al., 2004] and [Stewart, 2005] deal with this output-ambiguity by *sampling* from the intractable output-distribution and then defining the "correct" output as the vector of maximum posterior marginals. Furthermore, as we discussed in Section 5 training is intractable and makes use of approximations.

In this chapter we propose an alternative view of structured supervised learning as that of learning a conditional manifold. We make use of the close relation between auto-encoders and RBMs discussed in Section 2.2.1. We show that, when using a simple model where only the biases but not the connections are conditioned on input data, the output ambiguity disappears in the non-probabilistic formulation, and applying the model amounts to a simple application of a forward-mapping. In the case of gated connections, each input has a non-linear manifold as the output, so we do have an output-ambiguity here, too. However, similarly as in Section 5.4.2, we can use a simple reconstruction scheme in that case to obtain a good answer as we shall show. An additional advantage of the deterministic approach is that it allows us to perform training without making use of approximations.

The following section discusses structured supervised learning in more detail. The section that follows describes conditional auto-encoders for learning a manifold conditionally and discusses how to apply them to structured prediction problems.

6.1.1 Structured predictions with conditional graphical models

The supervised problem of simultaneously predicting multiple, correlated variables has attracted a lot of research recently. As mentioned in Section 2.6, many common tasks in computer vision, language processing and other areas, can be conveniently cast within this framework. Typical applications of structured discriminative learning include predicting a sequence of part-of-speech tags [Lafferty et al., 2001] or detecting structure in images [Kumar and Hebert, 2004]. Making multiple simultaneous predictions in these and many other problems has been shown to work much better than making many independent scalar predictions.

Currently, the standard way to solve structured prediction problems is by using a probabilistic graphical model on the outputs. The graphical model is conditioned on – ie. a function of – the input variables, which turns it into a discriminative model. Similarly as in unsupervised learning, using a graphical model makes it possible to take correlations between variables into account, while ensuring that inference and learning stay (at least approximately) tractable. Most models are formally defined as undirected graphs whose potential functions are functions of some input $\boldsymbol{x}$. In other words, the general form:

$$p(\boldsymbol{y}|\boldsymbol{x};W) = \frac{1}{Z(\boldsymbol{x})} \prod_s \Psi_s(\boldsymbol{y}_s;\boldsymbol{x},W). \tag{6.1}$$

Note that, in contrast to unsupervised learning, here the partition function is dependent on the inputs $\boldsymbol{x}$, similarly as in Chapter 5. The most common dependency structure on the outputs is a sequence structure, in which case the cliques $\boldsymbol{y}_s$ are pairs of nodes, and the model can thus be written:

$$p(\boldsymbol{y}|\boldsymbol{x};W) = \frac{1}{Z(\boldsymbol{x})} \prod_s \Psi_s(y_s, y_{s-1};\boldsymbol{x},W). \tag{6.2}$$

This model is the conditional analog of HMMs and is described in [LeCun et al., 1998]. The special case of *linear* potential functions was made popular recently by [Lafferty et al., 2001] as conditional random field (CRF). Most recent applications of this type of model make use of the linear special case.

In many structured supervised learning tasks, not only the outputs, but also the inputs are decomposed according to a graph. A common application of CRFs, for example, is the classification of elements in a sequence, such as tagging words in a text [Lafferty et al., 2001]. This and similar problems can be thought of as a sequence of scalar classification problems, in which there are correlations in the labels.

The existing structured prediction models are usually decomposable into an *observation component* and a *structure component*. The structure component is the graphical model on the output variables and the observation component is a simple element-wise predictor that is applied to each output variable independently. This decomposition is the direct analog of the decomposition into an observation model and a prior in unsupervised graphical models (Eq. 2.7). The structure component is a graphical model that assumes a fixed independence structure, such as a sequence or a grid structure.

Like in unsupervised problems, where we can use nonlinear manifolds as an alternative to graphical models for learning correlation structure, here we suggest learning a *manifold on the outputs* in a structured prediction problem, rather than fixing the output-structure to be a graphical model. The correlation structure on the outputs can then be learned from data similarly as in an unsupervised learning task. As in unsupervised learning, for efficiency and regularization we can use bi-partite structure and low-dimensional codes as an alternative to making conditional independence assumptions. An advantage of learning the output correlations instead of fixing them is that it makes it possible to let the correlation structure *itself* depend on the inputs, instead of modeling the correlation structure separately from the predictor.

6.2 Learning a manifold conditionally

In order to define a manifold conditionally, we now describe how we can turn an auto-encoder into a function of inputs.

6.2.1 Conditional auto-encoders

Recall that we can define the activity of hidden unit z_k in a linear auto-encoder as $\sum_j A_{jk} y_j$, where A_{jk} is the strength of the connection from y_j to z_k (Section 2.2.1). In Section 5.3.2 we described how we can condition an RBM by turning its parameters into functions of the conditioning information.

Using the same idea we can make the auto-encoder dependent on inputs $\boldsymbol{x}$ by turning the parameters A_{jk} into functions of $\boldsymbol{x}$. If we choose a linear dependency, we have, similarly as is Section 5.3.2:

$$A_{jk}(\boldsymbol{x}) = \sum_i A_{ijk} x_i, \tag{6.3}$$

where A_{ijk} is the i^{th} component of the linear mapping for weight A_{jk}.The output of hidden unit z_k now becomes $z_k = \sum_{ijk} A_{ijk} x_i y_j$.

As before we can think of the weight A_{ijk} also as a three-way connection that learns to model the *compatibility* between x_i, z_k and y_j. Alternatively, we can think of the input variables as being able to "vote" for connections to be present in the network. In other words, the inputs decide what kinds of correlations should be present in the outputs. (This view is the basis for the interpretation of GBMs as modulated filters (Section 5.2.3).)

A difference to Section 5 is that here the connectivity is not symmetric. To pursue the analogy to GBMs further, we also need to gate the weights W_{kj} from the hidden layer to the output layer to obtain a gated forward-mapping. We can define these similarly as

$$W_{jk}(\boldsymbol{x}) = \sum_i W_{ijk} x_i \tag{6.4}$$

Reconstructions can now be computed from hidden unit activities as $\hat{y}_j = \sum_{ik} W_{ijk} x_i z_k$.

Since gating the connections can be viewed as letting the inputs $\boldsymbol{x}$ modulate correlational structure, hidden units now capture how the inputs are *transformed* into outputs, rather than capturing static structure in the outputs themselves. The "gated auto-encoder" is illustrated in figure 6.1.

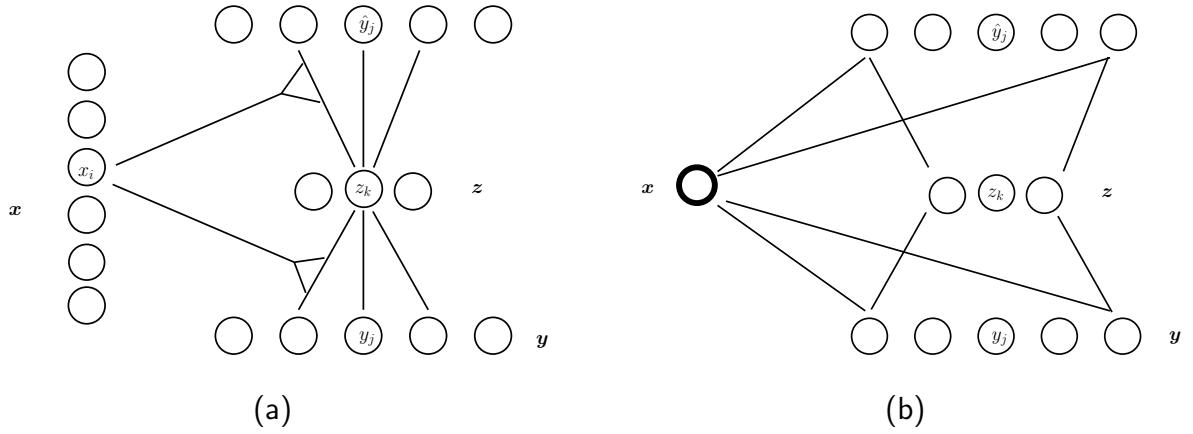

Figure 6.1: Learning a manifold conditionally: (a) Gated auto-encoder. Inputs modulate the correlation structure in the outputs. (b) Biased auto-encoder. Inputs affect only the biases (the vector of inputs $\boldsymbol{x}$ is depicted as a ring).

The model we described does not contain any biases, but we need to include these to be able to model affine dependencies. As in Section 5, the term "bias" here has a more general meaning than for unconditional models. In particular, we can include any connections that affect one or more of the possible subgroups of units: inputs-to-hiddens, inputs-to-outputs, inputs-to-reconstructions, outputs-to-hiddens, hiddens-to-reconstructions, just hiddens, just outputs, or just reconstructions.

An important special case of the model is a network that contains only the bias-connections $W^{\boldsymbol{xy}}$, $W^{\boldsymbol{yz}}$ and $W^{\boldsymbol{zy}}$, which connect inputs to outputs, outputs to hiddens, and hiddens to reconstructions, respectively. This model is the deterministic equivalent of the model described in [He et al., 2004]. It is illustrated in figure 6.1 (b): The embedding model cleans up predictions made by a separate prediction model. We call this model a biased auto-encoder, in contrast the gated auto-encoder described above, since here only the biases are functions of inputs. In the deterministic model, the application to test data does not come with the output-ambiguities and amounts to the simple application of a forward function. Both this model and [He et al., 2004] can be viewed as the "feature learning"-equivalent of common

decomposable models, such as CRFs. While those models consist of an observation- and a structure-component, here output-correlations are learned instead of being fixed. (Structure in the *dependencies* of the outputs on the inputs are not learned in the biased model, but they are in the gated model.)

In contrast to the GBM and [He et al., 2004], we can train these models without resorting to approximations. The presence of a tractable objective function also makes it easier to perform model comparison and to validate the correctness of model implementations by comparing gradients with finite-difference approximations.

Instead of using linear dependencies on the inputs (Eqs. 6.4 and 6.3), we can easily use non-linear ones, by modelling $W_{jk}(\boldsymbol{x})$ and $A_{jk}(\boldsymbol{x})$ as arbitrary non-linear functions, for example neural networks. As in Section 5.3.2, a possible compromise between flexibility and a manageable number of parameters is to share parameters by pre-mapping the inputs. For this end we can define: $W_{jk}(\boldsymbol{x}) = \sum_i W_{ijk}\phi_i(\boldsymbol{x})$, where $\phi(\boldsymbol{x})$ is a non-linear function that is trained along with the mappings using back-propagation. Similarly for the forward mapping $W_{jk}(\boldsymbol{x})$.

Applying a biased auto-encoder is a simple application of a feed-forward function. In the gated auto-encoder, fixing the input $\boldsymbol{x}$ yields a *manifold* in the output-space. This manifold is the deterministic analog of the distribution that we obtain from a GBM. In many real-world tasks (albeit not in all) we are interested in point estimates for the output rather than a whole manifold. To obtain a point estimate, similarly as in Section 5, we can *project* a suboptimal guess for the output onto the manifold in order to obtain the model-output. In the simplest case this guess can be the input itself (in case inputs and outputs have the same dimensionality). Alternatively, we can use a simple element-wise predictor to obtain the sub-optimal guess. In the latter case, there are no output-ambiguities even for the gated model, and the application of the model is just a feed-forward mapping.

6.2.2 Training

To compute gradients we can use back-propagation. This is possible as long as the dependency structure of the units forms a directed acyclic graph [LeCun et al., 1998] [Bottou, 1991], which is the case for both the biased model and the gated model.

It is important to note that the implementation of back-propagation through a tree is straightforward, and no different than back-propagation through a feed-forward model, when using an object oriented language [LeCun et al., 1998]. Bi-partite modules need to provide a forward-function and a backward-function, that implement the function application and gradient propagation, respectively. Complex models can be constructed simply by sticking these modules together.

We consider regression tasks in this section, in which the outputs are real-valued vectors, and we use squared reconstruction error as the cost function. However, output-units can also be defined as multinomial units, and the model trained using cross-entropy loss as is common in classification tasks.

6.2.3 Learning conditional "manifields"

In the model described above, the number of outputs is fixed, and has to be known (and to be the same) at training- and at test-time. However, as we mentioned before, in many structured prediction problems this number can be variable. A typical example is sequence labelling, where the length of the sequence can differ from case to case. To deal with this issue, we can make a stationarity assumption, similarly as in Section 5.3.1, *ie.* apply a single model defined only for a small patch to every possible location of the sequence and average, or add, the contributions of each one of them to compute the output. The approach for an unconditional sequence model is illustrated in figure 6.2. For images, we can use a grid-structure instead of a sequence, as we did in Section 5.3.1. The unconditional version of this approach is a kind of time-delay neural network [Waibel et al., 1989], and can be thought of

also as a simple version of a convolutional network [LeCun et al., 1998]. Note however that, although the weights are shared across different locations, the hidden units are not. Each site s has an individual and independent set of hidden units $\boldsymbol{z}_s$.

The model can also be viewed as a hybrid between a non-linear manifold and a field of experts [Roth and Black, 2005a], so we will call it a "manifield" in the following. Note that,

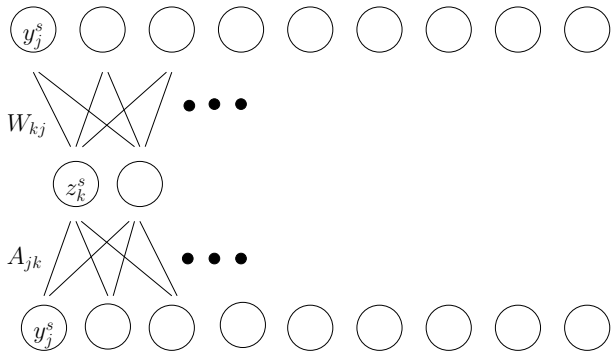

Figure 6.2: Illustration of an (unconditional) "manifield". A simple template-model is copied over a larger sequence to obtain a convolutional version of an auto-encoder, or equivalently the deterministic version of a field of experts.

here we are interested in conditional manifields, whose weights W_{ijk} are *functions* $W_{ijk}(\boldsymbol{x})$. For training we use back-propagation as before.

6.3 Experiments

6.3.1 De-noising a sequence

To demonstrate the use of a biased auto-encoder, we first consider the problem of de-noising a simple one-dimensional signal.

The problem of noise reduction can be defined as a structured supervised learning task, if we let the noisy data be the input and the de-noised data the output. The advantage of supervised learning is that no prior needs to be defined. Instead, the model is trained by

example to make the desired predictions. We consider a simple de-noising problem in which a sine-wave is corrupted by multiplicative noise. This type of noise is more difficult to deal with than additive noise. However, when using discriminative learning, knowledge of the type of corruption does not need to be known and is provided implicitly in the training data.

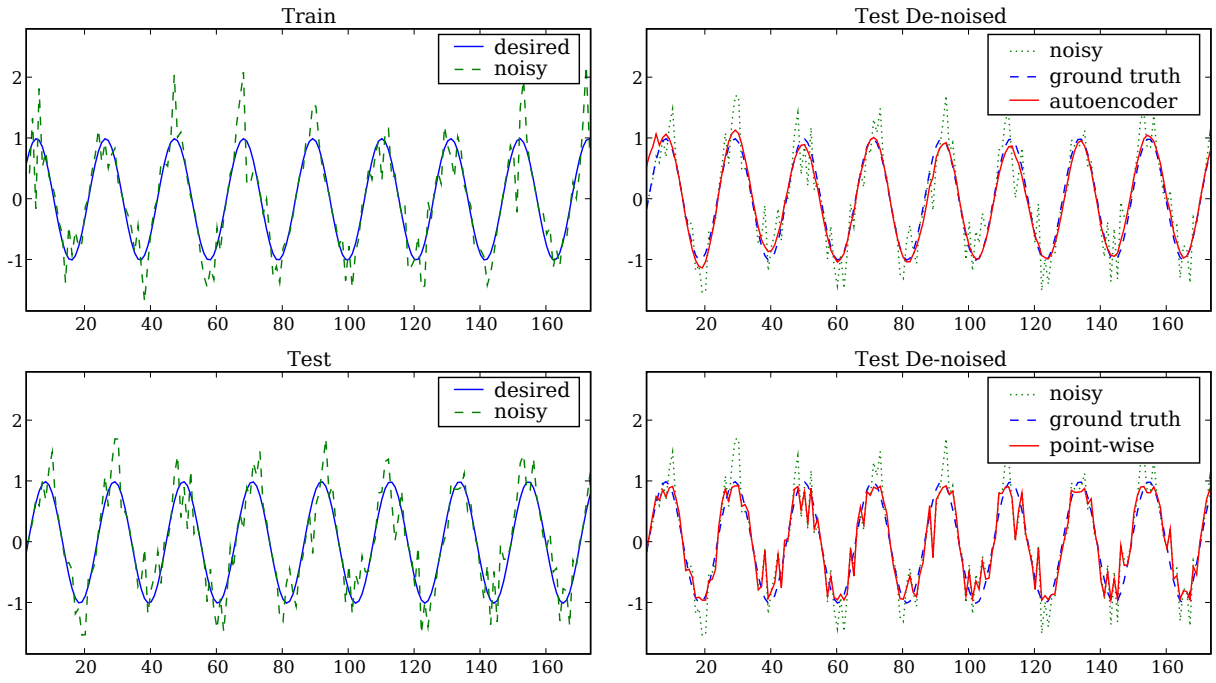

Figure 6.3: Removing multiplicative noise. Top left: Training data (noisy input, desired output). Bottom left: Test data. Top right: De-noising result from biased auto-encoder. Bottom right: De-noising result from independent predictor.

We corrupt the input sequence by multiplying each frame randomly and independently using a multiplier that is drawn from a Gaussian with mean 1 and variance 0.1. Parts of the training and test sequences are shown together with the noise-free ground truth in figure 6.3 (top and bottom left). We modelled the sequence as a conditional "manifield" with patch-size 7. We used a biased auto-encoder as the patch-model. The predictor is a backprop-network with one hidden layer containing 30 units. We experimented with varying the number of hidden units and the patch size without noticing any significant impact on the result. The whole training sequence contained 500 frames, which we split into 250 frames

for training and 250 frames for validation. The input $\boldsymbol{x}$ to the manifield model is the noisy sequence, the desired output the de-noised sequence. For training we added $\lambda\|W\|^2$ to the cost, where W is a vector containing all model parameters. The validation set was used to choose the optimal weight decay. The optimal weight decay on the validation set was 0.0001 (We have tried the all weight-decays in the set $\{0.1, 0.01, 0.001, 0.0001, 0.00001, 0.0\}$). We used a sequence of length 500 for testing. The top right plot of figure 6.3 depicts part of the de-noised test-sequence. The bottom right plot shows the de-noising result obtained by applying the backprop-network (denoted "independent predictor" in the figure) independently in each frame. We trained the network on the same training/validation set, where the optimal weight-decay was 0.001. The auto-encoder, not surprisingly, does a much better job than the independent predictor. The squared reconstruction errors on the test-sequence are 7.98 for the auto-encoder and 28.21 for the independent predictor.

6.3.2 Image super-resolution

The previous section described an application of a sequence-structured biased auto-encoder. In this section we consider the task of super-resolution of images: Predicting a high-resolution image from a lower-resolution input. We describe an experiment with a homogeneous model applied to image patches and an experiment with a grid-structured model applied to whole images.

Homogeneous model

In this experiment we consider a homogeneous model in which connections are gated by the input (Figure 6.1 (a)). We can think of the super-resolution problem as an image analogy problem as in Section 5. In this case, the source image pair consists of one low-resolution image and one high-resolution image, and the target image is a different low-resolution image for which we want to predict the higher-resolution rendering.

Since we use a homogeneous, patch-based model instead of a manifield in this experiment,

the model produces super-resolution versions of fixed-sized patches only. An application of the patch-based model is *zooming* into an image.

We used as the training data an image that we used previously in chapter 5 in a different type of analogy problem (figure 5.11) (left). We randomly sampled patches of size 32×32 pixels and sub-sampled these to obtain smaller patches of size 8×8. We trained a gated auto-encoder to predict the larger patches from the smaller ones. In other words, the input to the model is an 8×8-patch, from which we predict the 32×32-patch. We simply trained on the raw pixel intensities without performing any preprocessing on the data. The gated auto-encoder has 1024 input units, 20 hidden units, and 64 output units. We applied the model to the test image show in figure 5.12 (top) to test the model.

Figure 6.4 shows the result of applying the auto-encoder to zoom in on random patches of the test image. To generate the zooms, we cut out patches from the low-resolution test image and provide these as input to the model. Note that each input image defines a *manifold* in the output space. We can project the input image onto this manifold in order to compute the desired output. Note, however that inputs and outputs have different dimensionality in this case. In order to obtain an initial guess for the outputs, we first scaled up the input image to the output-size of 32×32 pixels, and then projected this image onto the manifold defined by the input. The results are show in the middle column. For comparison we also show the ground-truth in the right-most column. The results show that the model not only smoothes out the patches, but also fills in fine-structured details. Note that the model, being trained on only a single image pair, also tends to "invent" structure in order to make sense of the input. The dark patch in middle row, for example, gets rendered as "fur", since this is the structure among the training data patches that comes closest the observed input (see Figure 5.11). Existing super-resolution methods solve the same problem by performing an explicit look-up in a large database of patches to find the high-resolution match for a given low-resolution input. In our parametric framework, all the information from the training data-set is condensed in the model parameters after training. This is not only more efficient but

also makes the application of the model much simpler.

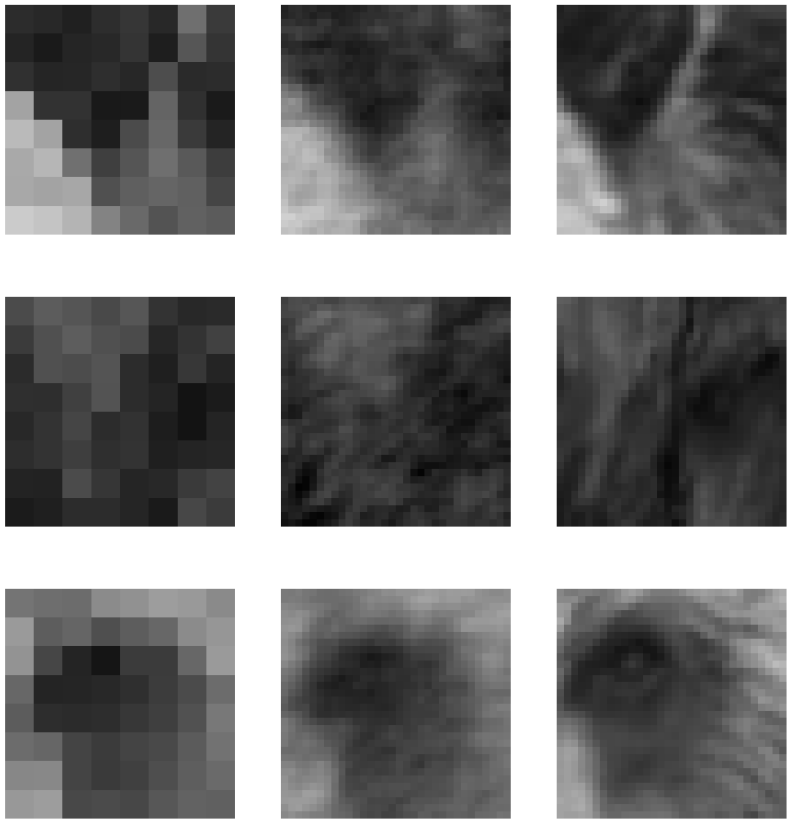

Figure 6.4: Discriminative super-resolution (homogeneous model).

Manifield model

In this experiment we consider super-resolution for larger images using a conditional manifield in the form of a grid-structured model. We trained a biased auto-encoder with three hidden layers, each of which has 50 units. The inner-most layer is a linear layer, the other two hidden layers are sigmoid layers. The independent predictor is a back-prop network with one hidden layer that has 50 units. The independent predictor is applied to 5×5 image patches. The

patch-size for the overlapping output-patches of the manifield-model is also 5×5.

We trained the model on 15×15 patches cropped at random positions from images taken from the Corel image database. We used sub-sampled images as inputs and the original images as outputs. Figures 6.5 and 6.6 show the result of applying the model on test-images not seen during training. In both figures we show the original (high-resolution) image on the top, the sub-sampled (input) image in the middle, and the high-resolution image produced by the trained model on the bottom.

6.4 Discussion

In this chapter we introduced a deterministic model for learning a manifold conditionally and demonstrated how a conditional manifold can be applied in structured prediction problems.

Using non-probabilistic, energy-based learning has been suggested as an alternative to probabilistic models for many machine learning applications (see, for example, [LeCun and Huang, 2005]). Our model can be viewed as an instantiation of that view. [Taskar et al., 2004] describe a different non-probabilistic model for learning with structured outputs. However that model is based on fixed graphical model structures and does not learn features. Furthermore, the basic model is linear. To achieve non-linearity it is, similarly as support vector machines [Cortes and Vapnik, 1995], typically trained in a dual representation with quadratic complexity in the number of training cases. The non-linear model is thus very difficult to apply to large data-sets or to train online. Our model can also be viewed as a particular instantiation of a higher-order neural network [Giles and Maxwell, 1987].

The sequence model bears some resemblances to an Input/Output HMM [Bengio and Frasconi, 1995]. However, our model is an undirected rather than directed model and the parameter sharing is quite different. Furthermore, the hidden units are not constrained to represent multinomial distributions, which allows them to model combinatorial structure.

There are several interesting directions for future research. It is likely that many struc-

tured prediction problems can profit from features that encode the *dependency* of the high-dimensional outputs on inputs, rather features that encode properties on the outputs only, and it could be useful to perform a rigorous comparison between these two approaches.

A general purpose model of transformations can be used to model variability *within* a manifold. Such a model can allow us to re-define a manifold as the set of transformations that produce it from a small set of prototype cases. [Vetter and Poggio, 1997] apply such a modeling approach in image synthesis tasks. It would be interesting to apply these models in this domain. A related problem is learning to cluster: Being able to model transformations from example-pairs makes it possible to build a system that can learn to distinguish when two cases belong together, and when not. Learning to cluster from a set of positive examples is therefore another possible application of the model.

Figure 6.5: Discriminative super-resolution with a conditional manifield.

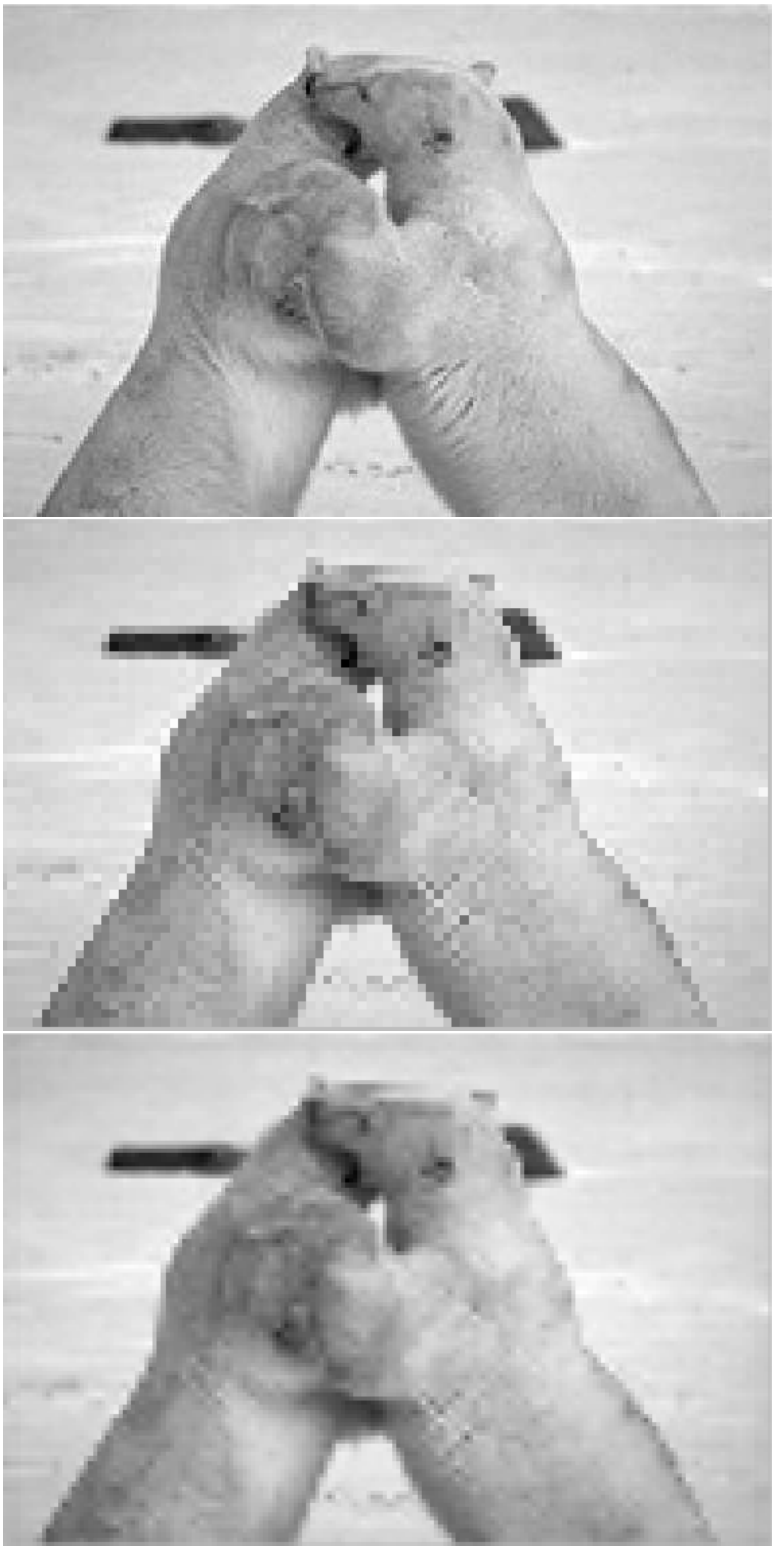

Figure 6.6: Discriminative super-resolution with a conditional manifield.

Chapter 7

Conclusions

In this section we provide a short summary and discussion of the work described in this thesis.

7.1 Summary

In this thesis we introduced a range of novel approaches to modeling structure in high-dimensional data. In the first part of this thesis we discussed existing methods and their relations and limitations, and described how supervision signals can be important in applications as a way to encode prior knowledge. Subsequently we have introduced several new models that overcome the limitations of existing approaches. We have used conditioning as a common means to encode prior knowledge and to model invariances.

The first half of the thesis focussed on non-parametric models. In chapter 3 we described a new non-linear latent variable model based on kernel density estimation. The model combines the benefits of non-parametric approaches and probabilistic generative models. In contrast to previous non-parametric methods it comes with a "forward-" and a "backward"-mapping and can therefore generalize to unseen data. Both training and applying the model is much more efficient than for existing non-parametric methods and the model is applicable to much larger data-sets. We demonstrated the use of the model on several example problems. We also showed how we can use conditioning as a way to encode invariances and to untangle

several interacting latent factors. We extended the idea of untangling variability in chapter 4, where we introduced a new model for learning about multiple similarity relations in a single shared latent space.

The second half of the thesis deals with parametric models. In chapter 5 we showed how modulating the parameters of a bi-partite model allows us to view embedding as learning about data transformations, and we demonstrated how a model of transformations can be used to encode invariances. In chapter 6 we described an extension of the approach to non-probabilistic models, and applied it to the problem of extracting features in structured discriminative learning tasks.

7.2 Discussion

We have introduced both parametric and non-parametric models in this thesis. As we mentioned in Section 2, these two types of approach have complementary benefits and weaknesses, and which type of method to choose is strongly application dependent. Problems like working with motion capture will likely profit more from the non-parametric models, because the data-sets are relatively small and non-parametric models can very quickly reveal highly non-linear structure in the data. Problems that require any form of online-processing of data are likely to be better off with parametric models. Online-processing is much more natural for parametric models and there has been no satisfying solution yet to the problem of enabling non-parametric methods to process data online.

A popular approach to modeling structure in data is given by *graphical models*. General graphical models describe the variability in data in a much more elaborate way than simple embedding methods. Unfortunately, encoding invariances in a graphical model can be difficult, and usually requires a significant amount of domain-knowledge. Graphical models can easily become quite complex in applications and modeling systems with a graphical model is sometimes equivalent to taking a "boxology"-approach. The modeling philosophy underlying

graphical models is somewhat contrary to the "engineering by example" philosophy that is common to many machine learning approaches, especially in supervised learning. Supervised methods allow us to specify the desired system behavior entirely by example, which minimizes the amount of required hand-coding. Because of this convenience supervised methods have been very successful in applications.

The models described in this thesis are located somewhere between supervised and unsupervised learning. The latent variable models on which our models are based are conceptually very simple. They are based on pairwise similarity relations in the non-parametric case (Chapters 3 and 4), and bi-partite structure in the parametric case (Chapters 5 and 6). The basic modeling philosophy is to keep the models as simple as possible and to use supervisions signals whenever possible. Invariances are encoded using side-information.

When introducing extra information a model usually turns into a "three-way"-model: In non-parametric models we partition the data-space into three groups in order to accomodate the three types of information (data, latent variables and extra knowledge); in parametric models we use use three-way parameters to connect the components of these three types of vector. However, it is important to note that since we always *condition* on the side-information and that information is *always given*, inference is exactly the same as in the corresponding standard embedding method. (This is in contrast to an alternative way of making use of multi-way interactions for modelling multiple degrees of freedom, where we fill in and then de-compose a multi-dimensional grid with multiple views of the same data [Tenenbaum and Freeman, 2000] [Vasilescu and Terzopoulos, 2002].) Retaining the simple bi-partite or non-parametric structure makes training relatively straightforward and allows us to apply the models without resorting to variational or similar approximations for performing inference.

Bibliography

[Allgower and Georg, 1993] Allgower, E. L. and Georg, K. (1993). Continuation and path following. In *Acta numerica, 1993*, pages 1–64. Cambridge Univ. Press, Cambridge.

[Anzai et al., 1999a] Anzai, A., Ohzawa, I., and Freeman, R. D. (1999a). Neural mechanisms for processing binocular information I. Simple cells. *Journal of Neurophysiology*, 82(2):891–908.

[Anzai et al., 1999b] Anzai, A., Ohzawa, I., and Freeman, R. D. (1999b). Neural mechanisms for processing binocular information II. Complex cells. *Journal of Neurophysiology*, 82(2):909–924.

[Baldi and Hornik, 1995] Baldi, P. and Hornik, K. (1995). Learning in linear neural networks: A survey. *IEEE Transactions on Neural Networks*, 6(4):837–858.

[Belkin and Niyogi, 2003] Belkin, M. and Niyogi, P. (2003). Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396.

[Bell and Sejnowski, 1997] Bell, A. J. and Sejnowski, T. J. (1997). Edges are the independent components of natural scenes. In Mozer, M. C., Jordan, M. I., and Petsche, T., editors, *Advances in Neural Information Processing Systems*, volume 9, page 831. The MIT Press.

[Belongie et al., 2002] Belongie, S. J., Malik, J., and Puzicha, J. (2002). Shape matching and object recognition using shape contexts. Technical report, University of California at Berkeley, Berkeley, CA, USA.

[Bengio et al., 2004] Bengio, Y., Delalleau, O., Le Roux, N., Paiement, J.-F., Vincent, P., and Ouimet, M. (2004). Learning eigenfunctions links spectral embedding and kernel PCA. *Neural Computation*, 16(10):2197–2219.

[Bengio and Frasconi, 1995] Bengio, Y. and Frasconi, P. (1995). An input output HMM architecture. In Tesauro, G., Touretzky, D., and Leen, T., editors, *Advances in Neural Information Processing Systems*, volume 7, pages 427–434. The MIT Press.

[Bishop, 1995] Bishop, C. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press.

[Bishop et al., 1998] Bishop, C. M., Svensén, M., and Williams, C. K. I. (1998). Gtm: the generative topographic mapping. *Neural Computation*, 10(1):215–234.

[Bottou, 1991] Bottou, L. (1991). *Une Approche théorique de l'Apprentissage Connexionniste: Applications à la Reconnaissance de la Parole*. PhD thesis, Université de Paris XI, Orsay, France.

[Bottou, 2004] Bottou, L. (2004). Stochastic learning. In Bousquet, O. and von Luxburg, U., editors, *Advanced Lectures on Machine Learning*, number LNAI 3176 in Lecture Notes in Artificial Intelligence, pages 146–168. Springer Verlag, Berlin.

[Bowling et al., 2005] Bowling, M., Ghodsi, A., and Wilkinson, D. (2005). Action respecting embedding. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 65–72, New York, NY, USA. ACM Press.

[Chopra et al., 2005] Chopra, S., Hadsell, R., and LeCun, Y. (2005). Learning a similarity metric discriminatively, with application to face verification. In *CVPR '05*, Washington, DC, USA. IEEE Computer Society.

[Cohn, 2003] Cohn, D. (2003). Informed projections. In S. Becker, S. T. and Obermayer, K., editors, *Adv. in Neural Information Processing Systems 15*, pages 849–856. MIT Press.

[Cook et al., 2007] Cook, J. A., Sutskever, I., Mnih, A., and Hinton, G. E. (2007). Visualizing similarity data with a mixture of maps. In *AI and Statistics, 2007*. Society for Artificial Intelligence and Statistics.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

[Cover and Thomas, 1990] Cover, T. and Thomas, J. (1990). *Elements of Information Theory*. John Wiley and Sons.

[Elgammal and Lee, 2004] Elgammal, A. and Lee, C. S. (2004). Separating style and content on a nonlinear manifold. In *Computer Vision and Pattern Recognition 2004*.

[Fleet et al., 2000] Fleet, D. J., Black, M. J., Yacoob, Y., and Jepson, A. D. (2000). Design and use of linear models for image motion analysis. *Int. J. Comput. Vision*, 36(3):171–193.

[Gehler et al., 2006] Gehler, P. V., Holub, A. D., and Welling, M. (2006). The rate adapting poisson model for information retrieval and object recognition. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, New York, NY, USA. ACM Press.

[Giles and Maxwell, 1987] Giles, C. and Maxwell, T. (1987). Learning, invariance, and generalization in high order neural networks,. *Applied Optics*, 26(23):4972.

[Goldberger et al., 2005] Goldberger, J., Roweis, S., Hinton, G., and Salakhutdinov, R. (2005). Neighbourhood components analysis. In *Adv. in Neural Information Processing Systems 17*.

[Graham and Allinson, 1998] Graham, D. B. and Allinson, N. M. (1998). Characterizing virtual eigensignatures for general purpose face recognition. In *Face Recognition: From Theory to Applications*, volume 163.

[Grigorescu et al., 2003] Grigorescu, C., Petkov, N., and Westenberg, M. A. (2003). Contour detection based on nonclassical receptive field inhibition. *IEEE Transactions on Image Processing*, 12(7):729–739.

[Grochow et al., 2004] Grochow, K., Martin, S. L., Hertzmann, A., and Popović, Z. (2004). Style-based inverse kinematics. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 522–531, New York, NY, USA. ACM Press.

[Härdle, 1989] Härdle, W. (1989). *Applied Nonparametric Regression*. Cambridge University Press, Cambridge.

[Hastie et al., 2001] Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning: data mining, inference and prediction*. Springer.

[He et al., 2004] He, X., Zemel, R. S., and Carreira-Perpinan, M. A. (2004). Multiscale conditional random fields for image labeling. In *CVPR*, volume 02, pages 695–702, Los Alamitos, CA, USA. IEEE Computer Society.

[Hertzmann et al., 2001] Hertzmann, A., Jacobs, C. E., Oliver, N., Curless, B., and Salesin, D. H. (2001). Image analogies. In *SIGGRAPH '01*, New York, NY, USA. ACM Press.

[Hinton, 1981] Hinton, G. (1981). A parallel computation that assigns canonical object-based frames of reference. In *Proc. of the 7th IJCAI*, Vancouver, BC, Canada.

[Hinton and Roweis, 2003] Hinton, G. and Roweis, S. (2003). Stochastic neighbor embedding. In *Adv. in Neural Information Processing Systems 15*.

[Hinton, 2002] Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800.

[Hinton and Salakhutdinov, 2006] Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

[Hofstadter, 1984] Hofstadter, D. (1984). The copycat project: An experiment in nondeterminism and creative analogies. Technical Report AIM-755, Massachussets Institute of Technology.

[Horn and Johnson, 1994] Horn, R. A. and Johnson, C. R. (1994). *Topics in Matrix Analysis*. Cambridge University Press, New York, NY, USA.

[Hyvärinen, 1999] Hyvärinen, A. (1999). Survey on independent component analysis. *Neural Computing Surveys*, 2:94–128.

[Jojic and Frey, 2000] Jojic, N. and Frey, B. (2000). Topographic transformation as a discrete latent variable. In *Advances in Neural Information Processing Systems 12*, Cambridge, MA. MIT Press.

[Jolliffe, 1986] Jolliffe, I. T. (1986). *Principal Component Analysis*. Springer.

[Jordan, 2008] Jordan, M. (2008). *An Introduction to Probabilistic Graphical Models*. To be published.

[Kumar and Hebert, 2004] Kumar, S. and Hebert, M. (2004). Discriminative fields for modeling spatial dependencies in natural images. In Thrun, S., Saul, L., and Schölkopf, B., editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA.

[Kwok and Tsang, 2003] Kwok, J. T. and Tsang, I. W. (2003). The pre-image problem in kernel methods. In *Proc. of the Twentieth Intern. Conf. on Machine Learning*.

[Lafferty et al., 2001] Lafferty, J., McCallum, A., and Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA.

[Lawrence, 2004] Lawrence, N. D. (2004). Gaussian process latent variable models for visualisation of high dimensional data. In *Adv. in Neural Information Processing Systems 16*.

[Lawrence, 2006] Lawrence, N. D. (2006). Mocap toolbox for matlab.

[Lawrence and Quinonero-Candela, 2006] Lawrence, N. D. and Quinonero-Candela, J. (2006). Local distance preservation in the gp-lvm through back constraints. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 513–520, New York, NY, USA. ACM Press.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[LeCun and Huang, 2005] LeCun, Y. and Huang, F. (2005). Loss functions for discriminative training of energy-based models. In *Proc. of the 10-th International Workshop on Artificial Intelligence and Statistics (AIStats'05)*.

[Lee and Elgammal, 2005] Lee, C.-S. and Elgammal, A. M. (2005). Facial expression analysis using nonlinear decomposable generative models. In Zhao, W., Gong, S., and Tang, X., editors, *AMFG*, volume 3723 of *Lecture Notes in Computer Science*, pages 17–31. Springer.

[Mardia et al., 1980] Mardia, K. V., Kent, J. T., and Bibby, J. M. (1980). *Multivariate Analysis*. Academic Press, London.

[Meinicke, 2000] Meinicke, P. (2000). *Unsupervised Learning in a Generalized Regression Framework*. PhD thesis, Bielefeld University.

[Meinicke, 2002] Meinicke, P. (2002). Personal communication.

[Meinicke et al., 2005] Meinicke, P., Klanke, S., Memisevic, R., and Ritter, H. (2005). Principal surfaces from unsupervised kernel regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(9):1379–1391.

[Memisevic, 2003] Memisevic, R. (2003). Unsupervised kernel regression for nonlinear dimensionality reduction. Diplomarbeit, Universität Bielefeld.

[Memisevic, 2006a] Memisevic, R. (2006a). Dual optimization of conditional probability models. Technical report, University of Toronto, Toronto, Canada.

[Memisevic, 2006b] Memisevic, R. (2006b). An introduction to structured discriminative learning. Technical report, University of Toronto, Toronto.

[Memisevic, 2006c] Memisevic, R. (2006c). Kernel information embeddings. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 633–640, New York, NY, USA. ACM Press.

[Memisevic and Hinton, 2005] Memisevic, R. and Hinton, G. (2005). Improving dimensionality reduction with spectral gradient descent. *Neural Networks*, 18(5-6):702–710.

[Memisevic and Hinton, 2005] Memisevic, R. and Hinton, G. (2005). Multiple relational embedding. In Saul, L. K., Weiss, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 17*, pages 913–920. MIT Press, Cambridge, MA.

[Memisevic and Hinton, 2007] Memisevic, R. and Hinton, G. E. (2007). Unsupervised learning of image transformations. In *CVPR '07*. IEEE Computer Society.

[Neal, 1993] Neal, R. M. (1993). Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, University of Toronto, Toronto.

[Nene et al., 1996] Nene, S. A., Nayar, S. K., and Murase, H. (1996). Columbia object image library (coil-20). Technical report, Columbia University.

[Olshausen, 1994] Olshausen, B. (1994). *Neural Routing Circuits for Forming Invariant Representations of Visual Objects*. PhD thesis, Computation and Neural Systems.

[Olshausen and Field, 1996] Olshausen, B. A. and Field, D. J. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609.

[Palmer, 1999] Palmer, S. (1999). *Vision Science: Photons to Phenomenology*. MIT Press.

[Principe et al., 1999] Principe, J., Xu, D., and Fisher, J. (1999). Information theoretic learning. In *Unsupervised Adaptive Filtering*, pages 265–319. John Wiley & Sons.

[Rao and Ballard, 1997] Rao, R. P. and Ballard, D. H. (1997). Efficient encoding of natural time varying images produces oriented space-time receptive fields. Technical report, University of Rochester, Rochester, NY, USA.

[Reichardt, 1969] Reichardt, W. (1969). Movement perception in insects. In Reichardt, W., editor, *Processing of optical data by organisms and by machines*. Academic Press, New York.

[Roth and Black, 2005a] Roth, S. and Black, M. J. (2005a). Fields of experts: A framework for learning image priors. In *CVPR '05*, Washington, DC, USA. IEEE Computer Society.

[Roth and Black, 2005b] Roth, S. and Black, M. J. (2005b). On the spatial statistics of optical flow. In *ICCV '05*, Washington, DC, USA. IEEE Computer Society.

[Roweis and Saul, 2000] Roweis, S. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(9):533–536.

[Sammon, 1969] Sammon, J. W. (1969). A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, 18:401–409.

[Schmitt, 2002] Schmitt, M. (2002). On the complexity of computing and learning with multiplicative neural networks. *Neural Computation*, 14(2):241–301.

[Schölkopf et al., 1998] Schölkopf, B., Smola, A., and Mueller, K.-R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319.

[Schölkopf and Smola, 2001] Schölkopf, B. and Smola, A. J. (2001). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA.

[Scott, 1992] Scott, D. (1992). *Multivariate Density Estimation: Theory, Practice, and Visualization*. John Wiley & Sons, New York.

[Sejnowski, 1986] Sejnowski, T. (1986). Higher-order Boltzmann machines. In Denker, J., editor, *Neural Networks for Computing*, pages 398–403. American Institute of Physics.

[Stewart, 2005] Stewart, L. (2005). Structure learning in sequential data. Master's thesis, University of Toronto.

[Taskar et al., 2004] Taskar, B., Guestrin, C., and Koller, D. (2004). Max-margin markov networks. In Thrun, S., Saul, L., and Schölkopf, B., editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA.

[Teh et al., 2005] Teh, Y., Seeger, M., and Jordan, M. (2005). Semiparametric latent factor models. In *Proceedings of the International Workshop on Artificial Intelligence and Statistics*, volume 10.

[Tenenbaum and Freeman, 2000] Tenenbaum, J. and Freeman, W. (2000). Separating Style and Content with Bilinear Models. *Neural Computation*, 12(6):1247–1283.

[Tenenbaum et al., 2000] Tenenbaum, J. B., de Silva, V., and Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, pages 2319–2323.

[Tijl De Bie and Cristianini, 2003] Tijl De Bie, M. M. and Cristianini, N. (2003). Efficiently learning the metric using side-information. In *Proc. of the 14th International Conference on Algorithmic Learning Theory*.

[Tipping and Bishop, 1999] Tipping, M. E. and Bishop, C. M. (1999). Probabilistic principal component analysis. *Journal Of The Royal Statistical Society Series B*, 61(3):611–622.

[Torkkola, 2003] Torkkola, K. (2003). Feature extraction by non-parametric mutual information maximization. *Journal of Machine Learning Research*, 3:1415–1438.

[Urtasun et al., 2006] Urtasun, R., Fleet, D. J., and Fua, P. (2006). 3d people tracking with gaussian process dynamical models. In *CVPR (1)*, pages 238–245. IEEE Computer Society.

[van Hateren and Ruderman, 1998] van Hateren, L. and Ruderman, J. (1998). Independent component analysis of natural image sequences yields spatio-temporal filters similar to simple cells in primary visual cortex. *Proc. R. Soc. Lond. B*, 265:2315–2320.

[Vasilescu and Terzopoulos, 2002] Vasilescu, M. A. O. and Terzopoulos, D. (2002). Multilinear analysis of image ensembles: Tensorfaces. In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part I*, pages 447–460, London, UK. Springer-Verlag.

[Vetter and Poggio, 1997] Vetter, T. and Poggio, T. (1997). Linear object classes and image synthesis from a single example image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):733–742.

[Viola et al., 1996] Viola, P., Schraudolph, N. N., and Sejnowski, T. J. (1996). Empirical entropy manipulation for real-world problems. In *Adv. in Neural Information Processing Systems 8*.

[Waibel et al., 1989] Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. J. (1989). Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37:328–339.

[Wang et al., 2006] Wang, J., Fleet, D., and Hertzmann, A. (2006). Gaussian process dynamical models. In Weiss, Y., Schölkopf, B., and Platt, J., editors, *Advances in Neural Information Processing Systems 18*, pages 1441–1448. MIT Press, Cambridge, MA.

[Weinberger and Saul, 2006] Weinberger, K. Q. and Saul, L. K. (2006). Unsupervised learning of image manifolds by semidefinite programming. *Int. J. Comput. Vision*, 70(1):77–90.

[Welling et al., 2005] Welling, M., Rosen-Zvi, M., and Hinton, G. (2005). Exponential family harmoniums with an application to information retrieval. In *Advances in Neural Information Processing Systems 17*. MIT Press, Cambridge, MA.

[Xing et al., 2003] Xing, E. P., Ng, A. Y., Jordan, M. I., and Russell, S. (2003). Distance metric learning with application to clustering with side-information. In *Advances in Neural Information Processing Systems 15*, pages 505–512. MIT Press, Cambridge, MA.