THE UNIVERSITY OF CHICAGO

HIGH-DIMENSIONAL GENERATIVE MODELS: SHRINKAGE, COMPOSITION AND AUTOREGRESSION

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF STATISTICS

BY
MARC GOESSLING

CHICAGO, ILLINOIS
AUGUST 2016

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I am grateful to my principal advisor Yali Amit for his guidance, advice and ideas. I also thank him for his openness and for giving me the freedom to explore my own ideas. Moreover, I appreciate his insistence on clear statistical modeling. I thank John Lafferty and Risi Kondor for being on my committee and for providing valuable feedback on my work. I also appreciate the fruitful discussions and collaborations with the other members of the Amitgroup – Danny, Mark, Gustav, Jiajun and Kendra. I thank the faculty of the statistics department for the interesting courses they offered. Finally, I am grateful to my fellow students who made the PhD program an enjoyable experience for me.

ABSTRACT

We present several approaches to learn generative models for high-dimensional data. Estimating the underlying distribution for a given dataset is one of the most fundamental problems in statistics. Once a distribution estimate is obtained we can in principle derive any population statistic. In particular, we can synthesize data, impute missing values, detect outliers, denoise observations and perform classification.

High-dimensional distribution estimation is a challenging problem due to the curse of dimensionality which states that without appropriate model assumptions the required number of samples for accurate estimation increases exponentially in the data dimension. The main theme of this dissertation is to decompose the estimation problem into simpler subproblems. Specifically, we consider mixture models in which the data points are divided into clusters (i.e., the data matrix is split row-wise) and separate models for each data cluster are learned. We also use coarse-to-fine strategies that start from vague descriptions of the data which are then successively refined. Another modeling technique we study is the partitioning of the variables into subsets (i.e., the data matrix is split column-wise). Separate models for these smaller subsets of variables can then be learned. A related approach are compositional models in which multiple local data models are combined into a global data model according to a specified composition rule. A further model family we explore are autoregressive models which make use of the fact that high-dimensional distributions can always be decomposed into products of univariate conditional distributions.

CHAPTER 1

INTRODUCTION

Given a collection of multivariate data points $\boldsymbol{x_n} \in \mathbb{R}^D$, $n = 1, \ldots, N$, we consider the fundamental task of learning the underlying distribution $\mathbb{P}(\boldsymbol{x})$. One interesting class of high-dimensional data are images. Image data has the advantage of being straightforward to visualize. Moreover, when working with images certain spatial dependencies are known a priori and transformations like shifts and rotations can explicitly be incorporated into the models. Transformation modeling can lead to considerably better distribution estimates, especially when learning from small datasets. On the other hand, when the training set is large then additional model constraints can actually degrade the performance because the underlying assumptions are typically not exactly true. With few exceptions, the approaches which we develop in this thesis avoid domain-specific assumptions and can hence be applied to general high-dimensional datasets.

The natural quality measure of a generative model is the predictive likelihood, i.e., the probability (or density) of holdout data under the model. While the training likelihood only captures how well the model is able to reproduce seen examples, the likelihood on unseen data assesses the generalization performance of the model. Consequently, we use the predictive likelihood to perform model selection and model comparison.

1.1 Motivation

Generative models have many potential applications. We briefly mention several of them. First of all, generative models can be used for *data understanding* because the learned parameters often provide insight into the data. Since generative models discover statistical regularities, they can also be used to *extract features* for dimensionality reduction or discriminative tasks. Distribution estimates allow us to *detect anomalies*. Specifically, we can identify outliers by evaluating likelihoods. Generative models are also useful for *denoising*.

Indeed, original samples can be recovered from noisy observations by finding the most likely data point conditioned on the noisy observation. It is also possible to *synthesize data* by sampling from the learned distribution. This produces instances which are similar but not identical to the training examples. Moreover, through conditional sampling we can produce examples with desired properties. In particular, we can *impute missing values*. According to Bayes theorem we can also perform *classification* by using distribution estimates for different classes. Furthermore, generative models are important as *prior distributions* for challenging vision tasks like posture detection, scene understanding or semantic image segmentation. Finally, distribution estimates can be used to construct *optimal compression schemes*. Some of the mentioned applications will be explored in the following chapters.

1.2 The Empty Space Problem

One of the simplest families of generative models are mixture distributions [e.g., 77]

$$\sum_{k=1}^{K} \pi_k \mathbb{P}_k(\boldsymbol{x}), \quad \pi_k \geq 0, \ \sum_{k=1}^{K} \pi_k = 1.$$

In the easiest case, the mixture components $\mathbb{P}_k$ are products of parametric distributions (e.g., Gaussian or Bernoulli distributions). It is well known that such product mixtures can approximate any distribution arbitrarily well by using a large number of components. Intuitively one might expect that learning a mixture distribution with more components will always improve the generative performance of the model. A mixture model in which the number of components is equal to the number of training examples is called a kernel density estimator [98]. Such a model spreads the observation weight of each sample to a local neighborhood. However, in high dimensions kernel density estimators are known to perform very poorly. The reason for this bad performance is the so-called *empty space problem* which states that in high-dimensional spaces most samples are very different from each other. This is the case because too many possible variations exist and hence each pair of samples is

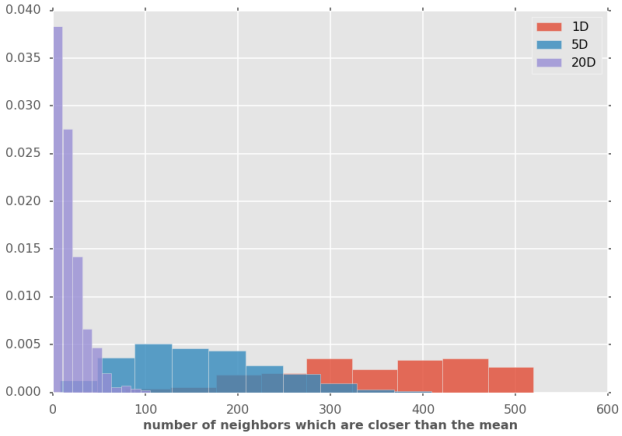

Figure 1.1: Number of neighbors (out of 999) which are closer to a given data point than the mean.

necessarily rather different.

We illustrate the empty space problem through a simple experiment. We sample $N = 1{,}000$ data points from a product normal distribution $\mathcal{N}(0, I_D)$ for different numbers of dimensions D. For each data point $\boldsymbol{x_n}$ we then count how many samples $\boldsymbol{x_m}$ are closer to it than the mean $\bar{\boldsymbol{x}}$, i.e., we compute

$$\#\{m \neq n : d(\boldsymbol{x_m}, \boldsymbol{x_n}) < d(\bar{\boldsymbol{x}}, \boldsymbol{x_n})\}.$$

Histograms for $D = 1, 5, 20$ are shown in Figure 1.1. For $D = 1$ on average around $1/3$ of the samples are closer to a given data point than the mean. For $D = 20$ only around 2% of the samples have a smaller distance than the mean. For $D = 100$ (not shown) the mean is almost always closer to a given data point than all other samples.

In Chapter 2 we quantitatively show that mixture models indeed deteriorate if too many components are learned in the standard way. We also propose a new learning method with which the performance improves the more components are used. The difference to a standard kernel density estimator is that we place the kernels closer to the high-probability regions of the distribution and not exactly at the locations of the training examples. Our approach is based on the general idea of shrinkage [56]. A shrinkage estimator "pulls" the standard

estimate towards some prior target (or targets). In our case the targets are the centers of data clusters. Specifically, we use a novel hierarchical regularization method which employs a penalized regression of observations onto the path in a hierarchical clustering. This allows us to robustly learn large mixture models. As an aside we show that the same method can also be used to regularize deep regression and classification trees.

Our hierarchical shrinkage approach can be considered a more principled variant of wavelets on trees [46]. The wavelet coefficients in that work are estimated through simple empirical averages and coefficients corresponding to sparse nodes are manually set to zero. The method we propose uses penalized regressions instead to shrink the coefficients. Moreover, the focus in Gavish et al. [46] is on semi-supervised learning while we are mostly interested in the unsupervised setting.

1.3 Distributed Representations

A mixture model essentially spreads probability mass around certain centroids in the data space. The shrinkage approach which we just mentioned is a sample-efficient method for learning large mixture models. However, the number of mixture components which can be learned through this procedure is bounded by the number of samples. In high-dimensional spaces a much larger number of centroids may be needed in order to properly cover the high-probability regions. The fundamental problem with mixture models is that only a single latent variable is employed. Indeed, for each sample exactly one mixture components is chosen and that component has to "explain" all dimensions. This severely limits the generative performance of mixture models. In the remainder of this section we outline approaches based on distributed representations [11]. This lead to models that can be understood as mixtures with exponentially many components which are learned in an efficient way.

Rather than using a mixture of product distributions we can consider a product of mixture

distributions. In this case, the joint distribution is modeled as the product

$$\mathbb{P}(\boldsymbol{x}) = \prod_{d=1}^{D} \mathbb{P}_d(\boldsymbol{x}(d))$$

where each marginal $\mathbb{P}_d(\boldsymbol{x}(d))$ is a mixture model. Such a model has D latent variables, corresponding to the chosen mixture component for each dimension. If K mixture components are used for each marginal then the joint distribution can have up to K^D modes.

Assuming that all variables are independent is a very strong assumption which in most cases is not justified. A generalization of product models with a weaker assumption are *partition models*. Such models use a manually specified partitioning of the variables into subsets

$$\{1, \ldots, D\} = \bigcup_{\ell=1}^{L} S_\ell.$$

Let $\boldsymbol{x}(S_\ell)$ denote the subset of variables $\boldsymbol{x}(d)$ with $d \in S_\ell$. Separate models $\mathbb{P}_\ell$ for each $\boldsymbol{x}(S_\ell)$ are then used and it is assumed that variables in different subsets are conditionally independent. Formally, each model $\mathbb{P}_\ell$ is controlled through a latent variable $\boldsymbol{h}(\ell)$ and conditioned on the latent state $\boldsymbol{h}$ the data vectors $\boldsymbol{x}(S_\ell)$ are independent, i.e.,

$$\mathbb{P}(\boldsymbol{x} \,|\, \boldsymbol{h}) = \prod_{\ell=1}^{L} \mathbb{P}_\ell(\boldsymbol{x}(S_\ell) \,|\, \boldsymbol{h}(\ell)). \tag{1.1}$$

To complete the model a prior distribution $\mathbb{P}(\boldsymbol{h})$ has to be specified. For image data, a regular grid that divides the image into patches provides a reasonable partitioning. Such an approach has been explored, for example, by Pal et al. [85]. In general, a good partitioning is characterized by providing weakly dependent subsets of variables so that the conditional independence assumption (1.1) is adequate and the distribution of the latent variables is easy to model.

One difficulty with partition models is that unless prior knowledge about the dependence structure is available it is hard to find a satisfactory partitioning of the variables. An even

more serious problem is that there simply might be no single fixed partitioning which works well for the whole dataset because the set of variables which are affected by different factors of variation might overlap. This restricts the scenarios in which partition models are useful. An extension that avoids fixed partitions are *compositional models* [e.g., 50] which are based on multiple "experts" [55]. An expert $\mathbb{P}_k(\boldsymbol{x})$ is simply a distribution of the full data vector $\boldsymbol{x}$ that might depend on a latent variable $\boldsymbol{h}(k)$. Given the latent activities $\boldsymbol{h}$ of the K experts the combined distribution of $\boldsymbol{x}$ is formed by composing the experts according to some rule $\mathbb{P}(\boldsymbol{x} \,|\, \boldsymbol{h}) = \gamma(\{\mathbb{P}_k(\boldsymbol{x})\} \,|\, \boldsymbol{h})$. As before, the model is completed by specifying a distribution of the expert activities $\boldsymbol{h}$. The latent state $\boldsymbol{h}$ with the highest posterior probability given a data point $\boldsymbol{x}$ is known as the *latent representation* of $\boldsymbol{x}$. Note that mixture models are a special case of compositional models that use a so-called one-hot representation. This means that exactly one expert is active for each data point. In general, compositional models use distributed representations, meaning that several experts can cooperate in order to explain the observation. With compositional models the evaluation of

$$\mathbb{P}(\boldsymbol{x}) = \sum_{\boldsymbol{h}} \mathbb{P}(\boldsymbol{x}, \boldsymbol{h}) = \sum_{\boldsymbol{h}} \mathbb{P}(\boldsymbol{h}) \cdot \mathbb{P}(\boldsymbol{x} \,|\, \boldsymbol{h})$$

is difficult. Indeed, summing over all all configurations of the latent variables is most of the time intractable. If that is the case, approximations [e.g., 94, 82] have to be used instead.

The classical approach to learn a distributed representation is principal component analysis [e.g., 57] which is an eigenvalue decomposition of the sample covariance matrix. The procedure yields K orthogonal vectors $\boldsymbol{w_k} \in \mathbb{R}^D$ corresponding to the directions of largest possible variance in the data. The latent representation $\boldsymbol{h}$ is given by the coefficients when projecting onto this basis. These coefficients are known as the principal components. The composed model $\gamma(\{\boldsymbol{w_k}\} \,|\, \boldsymbol{h})$ is a normal distribution with mean $\sum_{k=1}^{K} \boldsymbol{h}(k)\boldsymbol{w_k}$ and covariance matrix $\sigma^2 I_D$. Often a relatively small number $K \ll D$ of principal directions is sufficient to reconstruct the data well. A variant of principal component analysis are factor

models [e.g., 23]. In these models the conditional variances are not assumed to be equal for all dimensions but instead an arbitrary diagonal covariance matrix is used.

A characteristic of the eigendecomposition approach is that the representations are usually dense, i.e., the latent activations $\boldsymbol{h}(k)$ are all nonzero. Moreover, the basis vectors $\boldsymbol{w_k}$ typically have a global support. In practice this means that the factors of variation in the data are not disentangled and hence the latent distribution $\mathbb{P}(\boldsymbol{h})$ can be rather complicated. We illustrate this fact through a simple synthetic example. We consider the uniform distribution on the 32-element subset $\{(0,1),(1,0)\}^5 \subset \{0,1\}^{10}$. Note that there are five factors of variation corresponding to the state of the pairs $(\boldsymbol{x}(2\ell{-}1), \boldsymbol{x}(2\ell))$ for $\ell = 1, \ldots, 5$ with the two factor levels $(0,1)$ and $(1,0)$. Indeed, the distribution can be easily expressed (and learned) through a partition model with partitioning

$$\{1,2\} \cup \{3,4\} \cup \{5,6\} \cup \{7,8\} \cup \{9,10\}$$

and corresponding models

$$\mathbb{P}_\ell(\boldsymbol{x}(2\ell{-}1), \boldsymbol{x}(2\ell)) = \tfrac{1}{2} \cdot \mathbb{1}\{\boldsymbol{x}(2\ell{-}1) = 0,\, \boldsymbol{x}(2\ell) = 1\} + \tfrac{1}{2} \cdot \mathbb{1}\{\boldsymbol{x}(2\ell{-}1) = 1,\, \boldsymbol{x}(2\ell) = 0\}.$$

The ground-truth distribution is simply the product of these local models. Let us now consider a principal component analysis. Samples from the distribution together with the top 5 principal directions are shown in the left and middle panel of Figure 1.2. We see that each basis vector is a weighted average of all factors of variation. As a result, the latent distribution is more complicated than for the partition model, see right panel of Figure 1.2. While the variables $\boldsymbol{h}(k)$ are guaranteed to be (linearly) uncorrelated they are not independent of each other.

The example suggests that it may be beneficial to employ experts with local supports. By that we mean that the experts are uninformative about the majority of the variables. Ideally, each expert should only explain one factor of variation and for each source of variation there

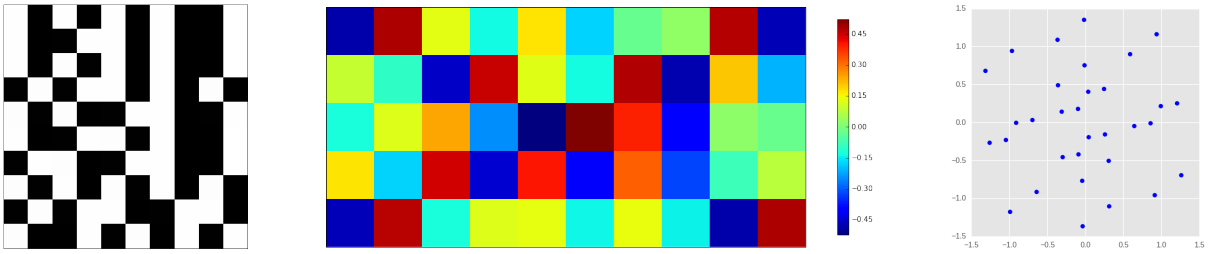

Figure 1.2: Principal component analysis for the synthetic example. Left panel: 10 random samples from the ground-truth model (rows indicate samples and columns indicate dimensions). Middle panel: First 5 principal directions obtained from 1000 samples. Right panel: Distribution of the first 2 principal components.

should exactly be one expert which focuses on it. In the signal processing community it is well-known that local supports of basis functions are related to sparse representations. Two popular representations for time signals are the Fourier transform and the wavelet transform [e.g., 74]. The difference between the two is that Fourier basis functions have global supports while wavelet basis functions have local supports. For non-stationary signals it now turns out that the wavelet representations are often (approximately) sparse but the Fourier representations are not. Even more, Olshausen and Field [84] showed that enforcing sparsity when learning a representation leads to experts which tend to have local supports. This insight triggered a lot of research in sparse dictionary learning [31]. Sparse representations are attractive because they simplify computations and increase robustness to noise or data corruption. Indeed, there is an entire field of research, called compressed sensing [30], which studies signal reconstructions using sparse representations. With sparse representations it is even possible to learn overcomplete dictionaries. This means that the number of experts can be larger than the data dimension.

An easy way to obtain a sparsely distributed representation is by using experts with binary latent variables. The latent state then simply specifies which of the experts are active, $\boldsymbol{h}(k) = 1$ means that the k-th expert is active. Such binary representations will be one of the main focuses of this thesis. A crucial aspect of every compositional model is the choice of the composition rule $\gamma(\{\mathbb{P}_k\} \,|\, \boldsymbol{h})$. In Chapter 3 we review various ways of composing

experts for binary data $\boldsymbol{x} \in \{0,1\}^D$. Building upon the work of Lücke and Sahani [72] we then introduce a novel composition rule which creates strong competition among experts. The rule discourages experts from learning similar structures and encourages local supports. In the same chapter we also introduce a new sequential initialization procedure which is based on a process of oversimplification and correction. This approach is quite different from conventional random initializations of learning algorithms. Experiments show that with our approach we can learn compact representations and we can successfully disentangle factors of variations. With compact representations the posterior distribution on the latent variables given $\boldsymbol{x}$ is highly peaked at a state $\boldsymbol{h}^\star$. We can thus use the simple approximation

$$\mathbb{P}(\boldsymbol{x}) \approx \mathbb{P}(\boldsymbol{h}^\star) \cdot \mathbb{P}(\boldsymbol{x} \,|\, \boldsymbol{h}^\star)$$

for likelihood evaluation. We perform a quantitative comparison with models that use the popular sum-of-log-odds composition rule. This rule is used for example in autoencoders [107], sparse dictionaries [31] and restricted Boltzmann machines [54]. In Chapter 3 we also consider specific methods for binary image data. In particular, we develop a model that uses experts which are invariant to shifts and rotations. Such invariances are a fundamental property which one would expect from image experts but they are not commonly implemented. By modeling these transformations we are able to learn concise and very intuitive representations for simple image classes.

In Chapter 4 we develop the approach further and introduce a more generally applicable model which in particular can deal with real-valued data $\boldsymbol{x} \in \mathbb{R}^D$. The idea is to employ for each variable d the "opinion" of the most reliable expert. Formally,

$$\mathbb{P}(\boldsymbol{x}(d) \,|\, \boldsymbol{h}) = \mathbb{P}_{k^\star(d)}(\boldsymbol{x}(d))$$

where $k^\star(d) \in \{1, \ldots, K\}$ depends on the levels of "expertise" of the active experts. Since conditioned on the latent state we obtain a partition model, we call this model a *dynamic*

partition model. We show that the model is able to learn the factors of variation in the earlier synthetic example without requiring a manual specification of the partitioning. Our approach is an alternative to Hinton's product of experts model [54] which combines experts by multiplying them together and renormalizing

$$\mathbb{P}(\boldsymbol{x} \,|\, \boldsymbol{h}) = \frac{\prod_{k:\boldsymbol{h}(k)=1} \mathbb{P}_k(\boldsymbol{x})}{\sum_{\boldsymbol{x'}} \prod_{k:\boldsymbol{h}(k)=1} \mathbb{P}_k(\boldsymbol{x'})}.$$

1.4 Autoregressive Decompositions

A limiting factor of the models discussed so far is that they assume conditional independence of the data given the latent state. Geometrically this means that probability mass is spread out in all directions, which may hurt the generative performance of the model. If dependencies between the variables are incorporated it may be possible to place probability mass more appropriately. Indeed, high-dimensional data is often concentrated on a relatively low-dimensional manifold which implies that the variables are highly dependent. The dimension D_0 of the manifold may be an order of magnitude smaller than D. Assuming independence of all variables can then lead to a log-likelihood which is a factor D_0/D smaller compared to the ground-truth distribution. An efficient way to model dependencies for high-dimensional data is to employ an *autoregressive decomposition*. A multivariate distribution can always be decomposed into a product of univariate conditional distributions

$$\mathbb{P}(\boldsymbol{x}) = \prod_{d=1}^{D} \mathbb{P}(\boldsymbol{x}(d) \,|\, \boldsymbol{x}(1{:}d{-}1)). \tag{1.2}$$

Learning the joint distribution thus reduces to estimating D univariate conditionals. An advantage of the factorization (1.2) is that likelihood evaluations and exact sampling are straightforward. Consequently, model selection and comparisons with other models are easy to perform. In the last two chapters of the thesis we utilize autoregressive models to learn

high-dimensional distributions. We perform quantitative comparisons with autoregressive models based on neural networks, for example the neural autoregressive density estimator [63] of Larochelle and Murray. The models we propose achieve state-of-the-art or better results on several standard benchmark datasets.

We explore two different approaches. In Chapter 5 we learn mixture models with sparse dependencies and shared parameters. Specifically, we use L1-penalized logistic and linear regressions to estimate the conditional distributions. Moreover, we consider a partition model in which the latent variables are interleaved with the observable variables. This model permits exact likelihood evaluations because the latent variables can be easily integrated out. The model is simple, fast to train, achieves excellent generalization performance and scales well to extremely high dimensions.

In Chapter 6 we use a nonparametric approach for high-dimensional distribution estimation. Compared to a mixture of parametric distributions these distribution estimators are more flexible but require more data and might generalize less well. Specifically, we learn autoregressive networks for binary data by training nonlinear probability estimators separately for each dimension. The conditional distributions are learned using LogitBoost [40], which is based on a sequence of decision trees [18]. Similarities and differences between our model and neural autoregressive networks are discussed in detail. The advantage of our approach compared to neural networks is that model training can be trivially parallelized over data dimensions. On the other hand, our method requires to combine the trained conditional distributions into a coherent joint model. We propose three different combination procedures for that and empirically evaluate them on several diverse datasets. Finally, we study how the ordering of the variables effects the performance of the trained model. In particular, we introduce a sorting procedure which allows us to build simple compression mechanisms.

CHAPTER 2

HIERARCHICAL SHRINKAGE

Mixture models [e.g., 77] are a flexible family for density estimation. Indeed, it is a well-known fact that Gaussian product mixtures $\sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{\mu_k}, \mathrm{diag}(\boldsymbol{\sigma_k^2}(1), \ldots, \boldsymbol{\sigma_k^2}(D))$ are universal approximators. That means they can approximate any continuous distribution arbitrarily well. For a proof see Silverman [98]. Mixture models are typically learned through either the EM algorithm [28] or through variational Bayesian methods [e.g., 13]. In both cases the responsibilities for the D-dimensional data points $\boldsymbol{x_n}$, $n = 1, \ldots, N$, are determined based on the posterior probabilities of the components. For high-dimensional data, however, the posterior distributions are often extremely peaked. Consequently, the samples are essentially divided into clusters S_k, $k = 1, \ldots, K$, and the parameters for the k-th mixture component are estimated by only using the training examples in S_k. In the case of Gaussian product mixtures the maximum-likelihood estimates are

$$\pi_k = \frac{|S_k|}{N}, \quad \boldsymbol{\mu_k}(d) = \frac{1}{|S_k|} \sum_{n \in S_k} \boldsymbol{x_n}(d), \quad \boldsymbol{\sigma_k^2}(d) = \frac{1}{|S_k| - 1} \sum_{n \in S_k} (\boldsymbol{x_n}(d) - \boldsymbol{\mu_k}(d))^2.$$

Increasing the number of mixture components decreases the bias of the model but increases the variance. Hence, the test performance in terms of likelihoods starts degrading once a certain number of components is reached. The problem of the standard approaches is that for large mixtures each component is learned from only very few training examples. In the extreme case where the number of components is equal to the number of training examples the maximum-likelihood estimates for the component means $\boldsymbol{\mu_n}$ are equal to the training examples $\boldsymbol{x_n}$ and the component variances $\boldsymbol{\sigma_n^2}(d)$ are often set to a fixed value σ_0^2 which is estimated from holdout data. Such a model is known as a kernel density estimator [98] because (Gaussian) kernels with bandwidth σ_0 are placed at the observed data points. It is well known that kernel density estimators perform very poorly in high dimensions. This is due to the empty space problem which states that high-dimensional spaces are inherently

sparse and that most data points appear as outliers. To alleviate this difficulty, in Bayesian approaches [e.g., 47] the individual mixture components are shrunk towards a prior target which leads to smoother estimates. Shrinkage [56] introduces a bias but at the same time has the potential to decrease the variance. The posterior component means in Bayesian models are usually convex combinations of the maximum-likelihood estimates and the prior mean.

In this chapter we propose a hierarchical regularization method in which the mixture templates are estimated jointly by using all the data. The idea is to learn a hierarchical clustering of the data and to compute a regression of the observations onto the path in that hierarchy. Standard Bayesian approaches can be described as flat shrinkage procedures because all components are pulled towards a single target. We implement a different "prior" by pulling the components towards multiple targets. In particular, we share parameters at multiple layers of a hierarchy. In our experiments the bias-variance tradeoff of the proposed method is much better than for flat shrinkage. In particular, with our approach the generative performance of mixture models improves the more components are used. This is a consequence of the explicit parameter sharing that we use. With existing shrinkage procedures a large number of mixture components eventually leads to overfitting.

The details of our hierarchical shrinkage method are provided in the next sections. We first focus on continuous data (Section 2.1-2.3) and then describe the necessary modifications for binary data (Section 2.4). We also explain how the same approach can be applied to the supervised setting (Section 2.5). We continue with an in-depth discussion of related work (Section 2.6) and provide theoretical support for our method (Section 2.7). We conclude with illustrative and quantitative experiments which focus on a comparison between flat and hierarchical shrinkage (Section 2.8).

2.1 Hierarchical Parameter Sharing

When learning mixture models with a large number of components it is natural to introduce some form of parameter sharing because the individual components can be quite similar and

hence should not be learned from essentially disjoint sets of training examples. Parameter sharing decreases variance at the cost of some additional bias. Using shared parameters is intuitively attractive but the crucial question is how much sharing should be imposed between the various mixture components. Certainly, similar components should share more parameters than dissimilar ones. We propose to measure similarity in terms of a top-down hierarchical clustering [e.g., 41] of the data. This leads to a tree structure which represents the geometry of the dataset. We assume that the nodes in the tree are numbered as $1, \ldots, V$ layerwise from left to right. Each leaf contains one or more data points. For similar training samples the paths from the root to the corresponding leaf will typically have a large overlap. In our approach the leaves represent the mixture components. The core idea is to model each mixture template as a sum of contributions $M_v \in \mathbb{R}^D$ along the associated path in the hierarchy. Specifically, the component means are of the form

$$\boldsymbol{\mu_k}(d) = \sum_{\ell=0}^{L_k} M_{\boldsymbol{v_k}(\ell)}(d) \tag{2.1}$$

where L_k is the depth of the k-th leaf and $\boldsymbol{v_k}(0), \ldots, \boldsymbol{v_k}(L_k)$ is the path from the root to the k-th leaf.

The additive decomposition (2.1) implies that the mixture templates are linear functions of the paths in the tree. Consequently, the parameters M_v can be estimated through a multivariate linear regression. The path information for the n-th sample can be represented by a sparse vector $\boldsymbol{a_n} = \sum_{\ell=0}^{L_n} \boldsymbol{e_{v_n(\ell)}} \in \{0,1\}^V$ where $\boldsymbol{e_i}$ is the i-th unit vector in $\mathbb{R}^V$ and $\boldsymbol{v_n}(0), \ldots, \boldsymbol{v_n}(L_n)$ denotes the path to the leaf which contains the n-th sample. The estimates $M(d) = (M_v(d))_{v=1,\ldots,V}$ can thus be obtained from the regression of the observations $\boldsymbol{x_n}(d)$ onto the span of the design matrix $A = [\boldsymbol{a_1}| \ldots |\boldsymbol{a_N}]^T$.

2.2 Regularization

We are especially interested in very large mixture models for which $K \sim N$. Since in this case the number of parameters can exceed the number of observations we need some form of regularization. Also, without regularization the contributions of all inner nodes would simply be zero and the leaf estimates would be equal to the corresponding empirical averages. We therefore use penalized regressions

$$\min_{M(d) \in \mathbb{R}^V} - \sum_{n=1}^{N} \mathrm{LL}(\boldsymbol{x_n}(d); \boldsymbol{a_n}^T M(d)) + \lambda ||M(d)||_2^2$$

where LL is the Gaussian log-likelihood (i.e., negative squared error). The penalty strength λ is a continuous tuning parameter which replaces the discrete choice of the number of mixture components. In most experiments an L2-penalty (i.e., ridge regression) worked best. In this case the parameters can be estimated by applying the smoothing matrix

$$S = (A^T A + \lambda I_D)^{-1} A^T \tag{2.2}$$

to the data vector. Alternatively, an L1-penalty (i.e., Lasso) can be used. In that case the contributions of a subset of the nodes might be exactly zero which means that the mixture has fewer than K components. Before running the regularized regressions we center the training examples (as in [45]). Moreover, it can be beneficial to normalize the scale of the variables if they are very different. In previous work [53] on mixture models (for a discriminative framework) differences between components were directly penalized rather than penalizing contributions along the tree path as in our approach. Consequently, there is no parameter sharing in those models apart from pulling all components towards the global mean.

2.3 Full Binary Trees

In principal any hierarchical clustering can be used. However, the deeper the tree is the more parameter sharing is possible. In our experiments the model performance improved monotonically with the number of parameters which we shared. Consequently, we suggest to recursively split data clusters into two until each data point forms its own cluster.[1] A simple method to obtain such a clustering is bisecting k-means [100]. The binary tree then consists of $2N-1$ nodes ($N-1$ inner nodes and N leaves). It is natural to introduce an additional symmetry constraint which reduces the number of parameters to N. For each pair of siblings (i.e., two nodes with the same parent) we enforce that their contributions sum to zero, i.e., $M_{2v} = -M_{2v+1}$ for all $v = 1, \ldots, N-1$. This has the effect that information which is common to both nodes is being pushed into the parent node. The child nodes then only represent deviations from the parent node. To incorporate this constraint into our model we change the representation of the samples to

$$\tilde{\boldsymbol{a}}_{\boldsymbol{n}} = \sum_{\ell=0}^{L_n} \boldsymbol{s_n}(\ell)\, \boldsymbol{e}_{\tilde{\boldsymbol{v}}_{\boldsymbol{n}}(\boldsymbol{\ell})}, \quad \boldsymbol{s_n}(\ell) = (-1)^{\boldsymbol{v_n}(\ell)+1}, \quad \tilde{\boldsymbol{v}}_{\boldsymbol{n}}(\ell) = \lceil (\boldsymbol{v_n}(\ell)+1)/2 \rceil.$$

This means the path information is encoded by a sparse vector with nonzero values ± 1 which indicate whether the left or the right child is visited on the path from the root to the corresponding leaf. For example, the design matrix corresponding to a complete binary tree of depth 2 is

$$\begin{bmatrix} +1 & -1 & -1 & 0 \\ +1 & -1 & +1 & 0 \\ +1 & +1 & 0 & -1 \\ +1 & +1 & 0 & +1 \end{bmatrix}.$$

The first column corresponds to the root contribution M_1 which is contained in the additive decomposition of any leaf. Note that if the tree is not balanced the design matrix will not

1. We can stop earlier if there are duplicates or if the dataset is very large.

be orthogonal.

If N components are used then our mixture model is an adaptive kernel density estimator in which the kernels are placed closer to the high-probability regions of the distribution and not exactly at the locations of the training examples. After estimating the mixture means, we also have to specify the variance of the mixture components. We do that by first training a kernel density estimator and choosing an appropriate bandwidth (any procedure can be used, see Section 2.6 for references) which will be used as the initial variance of the components. We then apply our hierarchical shrinkage method and select the right amount of regularization based on the likelihood performance on holdout data. Afterwards the variance of the mixture components can be readjusted if desired.

2.4 Binary Data

We now consider binary data $\boldsymbol{x} \in \{0,1\}^D$. First note that Bernoulli product mixtures $\sum_{k=1}^{K} \pi_k \text{Ber}(\boldsymbol{\mu_k})$ can represent any binary distribution. This follows immediately from the fact that every binary distribution can be written as a mixture of Dirac distributions. The common Bayesian approach [5] for learning a mixture of Bernoulli distributions uses a (symmetric) Beta prior for the Bernoulli probabilities. The posterior component means in this case are

$$\boldsymbol{\mu_n}(d) = \frac{1}{|S_k| + 2\epsilon} \left(\sum_{n \in S_k} \boldsymbol{x_n}(d) + \epsilon \right) \tag{2.3}$$

where $\epsilon > 0$ is the regularization parameter (and S_k is the set of samples for which the k-th component is responsible). Since the Bernoulli variance is a function of the mean, there is no separate variance estimate as for continuous data.

The necessary modifications to apply our hierarchical shrinkage approach to binary data are straightforward. We employ the same additive decomposition (2.1) but use it for the log-odds of the Bernoulli templates. The model can be trained by running penalized logistic regressions. An effect similar to centering of the training examples (which was mentioned

in Section 2.2) can be achieved by using a smaller penalty for the root contribution M_1 compared to other nodes. The hierarchical clustering itself can be determined by learning Bernoulli mixture models with two components each. However, simply using k-means on the binary data led to very similar results in our experiments.

2.5 Supervised Learning*

It is straightforward to apply our hierarchical shrinkage method to nonparametric regression problems. Given data $(\boldsymbol{x_n}, y_n) \in \mathbb{R}^{D+1}$ we can simply use the Nadaraya-Watson estimator [e.g., 99]

$$\widehat{m}(\boldsymbol{x}) = \frac{\sum_{n=1}^{N} \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu_n}, \sigma_0^2 I_D) y_n}{\sum_{n=1}^{N} \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu_n}, \sigma_0^2 I_D)}$$

for the conditional mean $\mathbb{E}[y|\boldsymbol{x}]$ where $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \Sigma)$ is the density at $\boldsymbol{x}$ for a Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance matrix Σ. In this attempt the tree structure would be learned based on the similarity of the covariates $\boldsymbol{x}$. Moreover, a regression with a D-dimensional response would be required to compute the mixture means $\boldsymbol{\mu_n}$.

For supervised problems it makes more sense to learn the tree based on the response y. Decision trees [18] provide a hierarchical partitioning of the dataset by recursively splitting data clusters into two using a supervised criterion. The problem with decision trees is similar to mixture models. Shallow decision trees often suffer from a substantial bias while deep decision trees typically result in a large variance because they partition the dataset into very small groups of samples and the parameter for each leaf is estimated by using only one of the groups. We suggest to grow the decision tree until all leaves are pure, i.e., until they contain only a single data point. The common practice is then to prune the tree [18]. Instead of that, we propose to run again a regularized linear (for regression trees) or logistic (for classification trees) regression of the response onto the path in the hierarchy. This results in a smoothing of the leaf values. The penalty strength acts as a continuous replacement for the discrete choice of tree depth.

2.6 Related Work

We now discuss related work from various fields of research.

Tree-based models

The work most closely related to our approach are wavelets on trees [46]. The authors construct a multi-scale orthonormal basis $(\boldsymbol{\psi_i})_{i=1,\dots N}$ for real-valued functions on $\{1,\dots,N\}$ based on a tree structure. Our sparse representation $\tilde{\boldsymbol{a}}_{\boldsymbol{n}}$ can be thought of as a replacement for the inner products $\langle \mathbb{1}_n, \boldsymbol{\psi_i} \rangle = \boldsymbol{\psi_i}(n)$ where $\mathbb{1}_n$ is the indicator function for the n-th sample. The main consequence of the enforced orthonormality is that regularization becomes less straightforward because the natural scale of the nodes is destroyed. In our design matrix columns corresponding to deep nodes have smaller norms (compared to nodes at higher levels) because the path to fewer samples runs through those nodes. Consequently, deep nodes experience a stronger shrinkage since their effect on the likelihood term (for the same parameter magnitude) is smaller. Gavish et al. [46] estimate coefficients based on simple empirical averages, i.e., unbiased estimators are used. Coefficients corresponding to sparse nodes are manually set to zero. Our approach is more principled and uses penalized regressions to shrink the coefficients. Moreover, Gavish et al. focus on semi-supervised learning (i.e., the unknown target function is only estimated at points which are available during training), applications to mixture modeling or decision trees are not explored.

Some related smoothing methods have been proposed for probability estimation trees. For example, the procedure of Bahl et al. [7] uses a weighted average of the frequencies along the path from the root to the leaf where the weights are determined through a forward-backward algorithm. Similarly, Ferri et al. [37] recursively compute a convex combination of the frequencies at a node and at the corresponding parent node. The weighting there is chosen in an ad-hoc manner.

Tree-based hierarchies are also popular for topic modeling [e.g., 14]. Each node in these

models corresponds to a topic and given a path in the tree a mixture of the topics along the path is formed. In contrast to our work, all nodes there correspond to mixture components and the components are not decomposed additively.

Another related model are Brownian motion trees. They were used by Felsenstein [35] to reconstruct the topological form and branching times of evolutionary trees. The difference to our work is that an additive decomposition of the covariances between leaves is used rather than an additive decomposition of the means.

Parameter sharing

In dictionary learning [e.g., 4] the response is approximated through the product $M\boldsymbol{a}$ where M is a $D\times R$ matrix of dictionary elements and $\boldsymbol{a}$ is an R-dimensional sparse (real-valued) representation vector. The learning procedure alternates between updates of M and updates of $\boldsymbol{a}$. In contrast to that, the representation $\boldsymbol{a}$ in our work is binary (or ternary) and deduced from the tree structure. Moreover, in our approach the node contributions (counterpart of the dictionary elements) experience shrinkage rather than being computed through an SVD.

A related approach to learn a mixture with shared parameters was proposed by Tang et al. [101]. Mixture components there are modeled through a sequence of linear transformations starting from an isotropic Gaussian. Learning is performed in a greedy layer-wise manner using the EM algorithm. In their experiments only relatively shallow architectures (2 or 3 layers) led to substantial improvements.

In deep autoregressive networks [52] the observable variables are regressed onto latent binary variables rather than on known paths as in our work. Since the explanatory variables are stochastic (i.e., not a deterministic function of the observable variables) a variational learning procedure is required to train the model.

Kernel density estimation

Previous work on adaptive kernel density estimation has focused on choosing appropriate bandwidths for different locations and dimensions [e.g., 2, 102, 71]. Our approach adapts the means of the kernels instead.

A robust kernel density estimator was introduced by Kim and Scott [58]. In their work, kernel density estimation is formulated as mean estimation in kernel space. Instead of using the empirical average a robustifying M-estimator is applied. This procedure however only adjusts the mixing weights of the estimator, so that outliers can be excluded. The bandwidths and the means of the kernels are unaffected. Robustness in our method comes from the fact that outliers are expensive to fit well because they deviate a lot from the common structure and the corresponding weights would not be reusable. Moreover, the symmetry constraint further suppresses unusual deviations.

2.7 Theory

The purpose of this section is to analytically illustrate the effect of parameter sharing in a simple framework and to provide an argument why multi-layer parameter sharing may be beneficial. We consider the normal means problem [108] which is an abstract framework that unifies multiple nonparametric estimation problems. The task is to estimate K normal means $\boldsymbol{\theta_k} \in \mathbb{R}^D$ from independent observations $\boldsymbol{x_k} \sim \mathcal{N}(\boldsymbol{\theta_k}, \sigma^2 I_D)$. Without further assumptions the smallest achievable estimation risk (mean squared error) for $\boldsymbol{\theta} = (\boldsymbol{\theta_1}^T, \ldots, \boldsymbol{\theta_K}^T)^T$ is as follows.

Theorem (7.28 & 7.39 in Wasserman [108])**.** *The ideal linear shrinkage estimator of the form* $(b\boldsymbol{x_1}, \ldots, b\boldsymbol{x_K})$ *with* $b \in \mathbb{R}$ *is (asymptotically) minimax and has associated risk*

$$\frac{KD\sigma^2||\boldsymbol{\theta}||^2}{KD\sigma^2 + ||\boldsymbol{\theta}||^2}.$$

However, if the means are actually close to each other then a smaller risk can be achieved.

Let $\boldsymbol{\theta_0} = \frac{1}{K}\sum_{k=1}^{K}\boldsymbol{\theta_k}$ and denote $\boldsymbol{\eta_k} = \boldsymbol{\theta_k} - \boldsymbol{\theta_0}$, $||\boldsymbol{\eta_k}|| = \delta_k$. Instead of estimating $\boldsymbol{\theta}$ directly, we can first estimate $\boldsymbol{\theta_0}$ and then estimate $\boldsymbol{\eta} = (\boldsymbol{\eta_1}^T, \ldots, \boldsymbol{\eta_K}^T)^T$. To estimate $\boldsymbol{\theta_0}$ we pool all observations and simply use the MLE[2] $\widehat{\boldsymbol{\theta}}_{\mathbf{0}} = \overline{\boldsymbol{x}}$. Estimating $\boldsymbol{\eta}$ is easier than the original problem if $||\boldsymbol{\eta}|| < ||\boldsymbol{\theta}||$.

Corollary. *With estimators $\widehat{\boldsymbol{\theta}}_{\boldsymbol{k}} = \overline{\boldsymbol{x}} + \widehat{\boldsymbol{\eta}}_{\boldsymbol{k}}$, where $\widehat{\boldsymbol{\eta}} = (\widehat{\boldsymbol{\eta}}_{\mathbf{1}}, \ldots, \widehat{\boldsymbol{\eta}}_{\boldsymbol{K}})$ is the ideal linear shrinkage estimator of the form $(b(\boldsymbol{x_1} - \overline{\boldsymbol{x}}), \ldots, b(\boldsymbol{x_K} - \overline{\boldsymbol{x}}))$ with $b \in \mathbb{R}$, the risk is*

$$D\sigma^2 + \frac{K(D-1)\sigma^2||\boldsymbol{\eta}||^2}{K(D-1)\sigma^2 + ||\boldsymbol{\eta}||^2}.$$

Proof. We can decompose the error as

$$\begin{aligned}\sum_{k=1}^{K}||\widehat{\boldsymbol{\theta}}_{\boldsymbol{k}} - \boldsymbol{\theta_k}||^2 &= \sum_{k=1}^{K}||\overline{\boldsymbol{x}} + \widehat{\boldsymbol{\eta}}_{\boldsymbol{k}} - \boldsymbol{\theta_0} - \boldsymbol{\eta_k}||^2 \\ = \sum_{k=1}^{K}\{||\overline{\boldsymbol{x}} - \boldsymbol{\theta_0}||^2 &+ 2(\overline{\boldsymbol{x}} - \boldsymbol{\theta_0})^T(\widehat{\boldsymbol{\eta}}_{\boldsymbol{k}} - \boldsymbol{\eta_k}) + ||\widehat{\boldsymbol{\eta}}_{\boldsymbol{k}} - \boldsymbol{\eta_k}||^2\}\end{aligned}$$

and hence

$$\mathbb{E}\sum_{k=1}^{K}||\widehat{\boldsymbol{\theta}}_{\boldsymbol{k}} - \boldsymbol{\theta_k}||^2 = K\frac{D\sigma^2}{K} + \frac{K(D-1)\sigma^2||\boldsymbol{\eta}||^2}{K(D-1)\sigma^2 + ||\boldsymbol{\eta}||^2}$$

because $\mathbb{E}(\overline{\boldsymbol{x}} - \boldsymbol{\theta_0})^T(\widehat{\boldsymbol{\eta}}_{\boldsymbol{k}} - \boldsymbol{\eta_k}) = \mathbb{E}(\overline{\boldsymbol{x}} - \boldsymbol{\theta_0})^T\mathbb{E}(\widehat{\boldsymbol{\eta}}_{\boldsymbol{k}} - \boldsymbol{\eta_k}) = 0$ since $\overline{\boldsymbol{x}}$, $\boldsymbol{x_k} - \overline{\boldsymbol{x}}$ are independent and $\mathbb{E}\overline{\boldsymbol{x}} = \boldsymbol{\theta_0}$. The expression for the risk of $\widehat{\boldsymbol{\eta}}$ can be derived analogously to Theorem 7.39 in Wasserman [108] using the fact that $\boldsymbol{x_k} - \overline{\boldsymbol{x}}$ has $D-1$ degrees of freedom. □

If $||\boldsymbol{\eta}||$ is small compared to $||\boldsymbol{\theta}||$ then the procedure with weight sharing outperforms the direct estimation. Indeed, the risk can be up to a factor of K smaller (in the limit as $||\boldsymbol{\eta}|| \to 0$ and $||\boldsymbol{\theta}|| \to \infty$). This provides an explanation why hierarchical parameter sharing with multiple small contributions performs better than a shallow architecture. Note that ridge regression is a linear shrinkage estimator. By choosing the regularization strength

2. This scenario is easier to analyze. With more sophisticated estimators for $\boldsymbol{\theta_0}$ the associated risk will be even smaller.

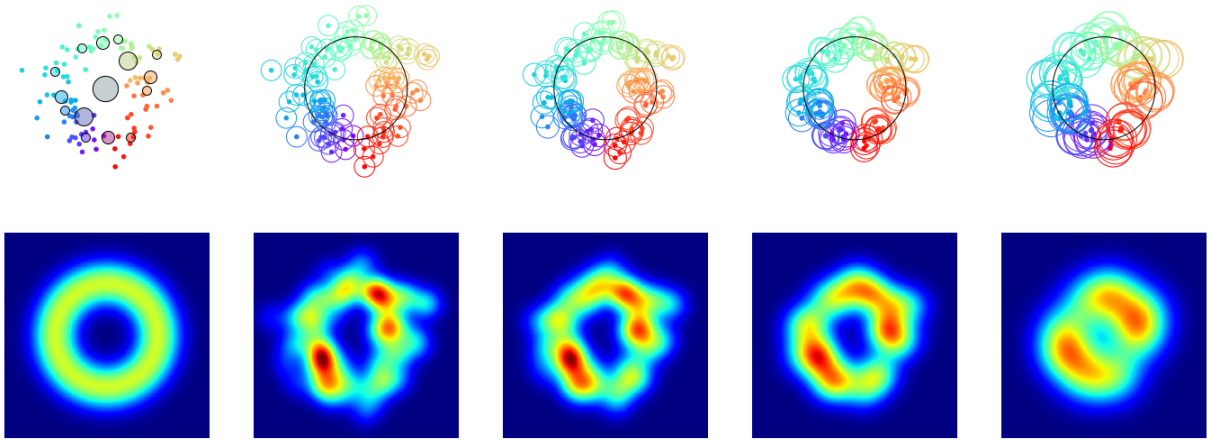

Figure 2.1: Bivariate density estimation. 1st column: Training samples together with tree nodes up to depth three (top) and true density (bottom). 2nd-5th column: Placed kernels together with underlying manifold (top) and resulting density estimate (bottom) for increasing amounts of shrinkage.

appropriately one can expect the corresponding risk to be similar to the one of the optimal linear estimator.

2.8 Experiments

2.8.1 Illustrative Example

We illustrate our proposed estimation procedure through a low-dimensional synthetic example (see Figure 2.1). The training set consists of 100 samples from a circle with Gaussian noise (1st column). The task is to estimate the underlying distribution from these data points. A common approach is to use a standard kernel density estimator (2nd column). With our hierarchical shrinkage procedure however it is possible to obtain a smoother density estimate. In the top left panel we overlaid the training samples with inner nodes from the learned hierarchical clustering tree. Hierarchical shrinkage now pulls the samples towards these targets. The resulting regularized kernel density estimators for three different penalty strengths are shown in the 3rd-5th column. Note that the training samples move closer to the underlying manifold and are not simply being pulled towards the global mean as with

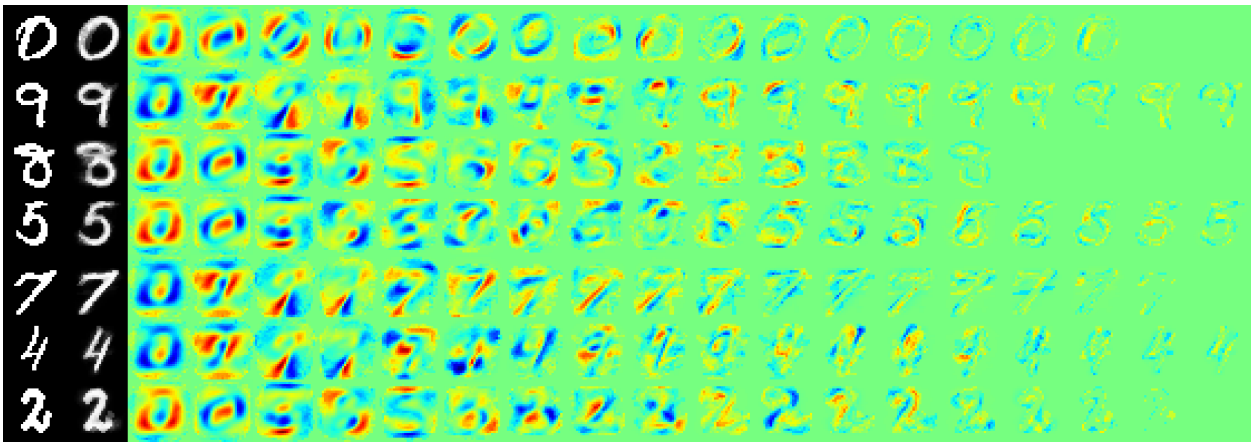

Figure 2.2: Mixture components for MNIST digits. 1st column: Training samples. 2nd column: Corresponding smoothed templates. Remaining columns: Decomposition of the templates (on the log-odds scale).

flat shrinkage procedures. However, if the amount of regularization is too large then a bias towards the data mean is actually introduced. For all estimates the kernel bandwidth was chosen based on the likelihood performance on holdout data. In low-dimensional spaces the benefits from shrinkage can be rather small. As we show next, in high dimensions hierarchical shrinkage can lead to substantially better distribution estimates.

2.8.2 MNIST Handwritten Digits

For a quantitative evaluation of our procedure we use the MNIST handwritten digits dataset [65]. There are 60,000 training samples and 10,000 test samples of dimension $28 \times 28 = 784$. We binarized the data by thresholding at 0.5. Hierarchical clustering applied to this dataset yielded a tree in which the depth varies from 9 to 30 with an average depth of 18. In Figure 2.2 we show some resulting smoothed templates when hierarchical shrinkage is applied. We also show the corresponding original samples and the coarse-to-fine decompositions. Peculiarities of the samples which cannot be found in other training examples are smoothed out while common features are retained. Note that with flat shrinkage as in equation (2.3) the templates would be regularized simply by pulling the probabilities a bit closer towards 1/2. Visually they would look no different from the original samples.

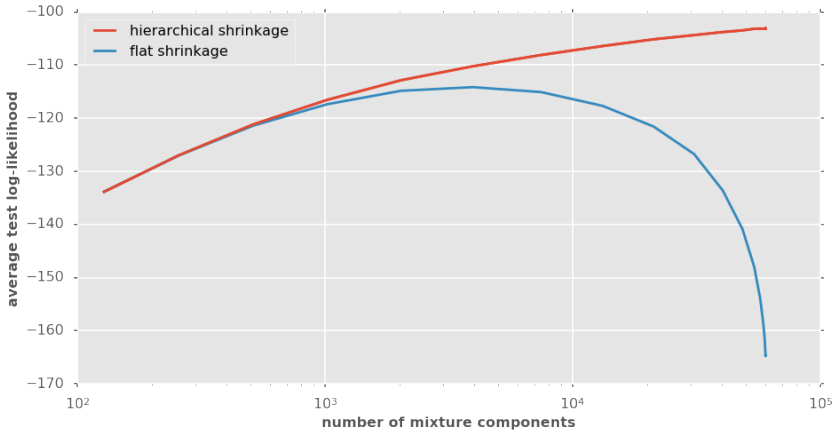

Figure 2.3: Average test log-likelihood for MNIST digits as a function of the number of mixture components.

The likelihood performance of the mixture model with hierarchical shrinkage improves the more components are used. This is in stark contrast to a mixture model with flat shrinkage which overfits when too many components are used. In Figure 2.3 we show the corresponding average log-likelihood for the test set of the MNIST digits as a function of the number of mixture components. With flat shrinkage the optimum is reached at around 3,000 components. This means that about one component per 20 samples is used. With our hierarchical shrinkage method the optimum is reached when the number of components is equal to the number of training examples. Moreover, for any number of mixture components the generative performance of our model is better than the standard approach. In particular, with hierarchical shrinkage the maximum (-103.07 nats) is substantially higher than with flat shrinkage (-114.15 nats).

Our density estimator can also be used for classification. We simply learn separate models for each class and assign new samples to the class with the highest likelihood (this is Bayes rule with a uniform prior). A kernel density estimator with flat shrinkage achieves a test accuracy of 95.68% on this data. Our density estimator with hierarchical shrinkage on the other hand achieves a test accuracy of 97.34%. For comparison, a support vector machine with Gaussian kernel (and tuned penalty parameter) trained on the same dataset achieves a test accuracy of 96.99%.

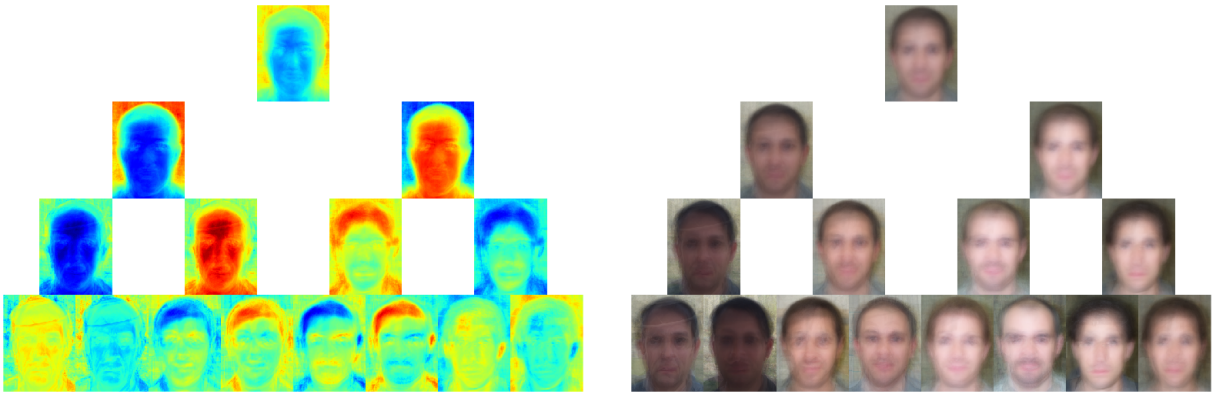

Figure 2.4: First three layers of the parameter-sharing tree for Caltech faces. Left: Node contributions M_v (shown is the average over the three color channels). Right: Aggregated templates (including the global mean) at inner nodes.

2.8.3 Caltech Faces

We now consider very high-dimensional real-valued data. The Caltech face dataset [36] consists of 450 high-resolution color images. We downsampled the images to $200 \times 150 \times 3$, which are 90,000 dimensions. We use 400 images for training and the remaining 50 images for testing. The hierarchical clustering produces a tree in which the leaves have a depth between 5 and 16. Figure 2.4 visualizes the top three layers of the parameter sharing hierarchy. Individual node contributions correspond amongst others to background brightness, face brightness, hair line and facial expression. Note that without regularization the aggregated templates at each node (i.e., the sum of contributions up to that node) would be equal to the sample average of the corresponding cluster. However, with regularization the aggregated templates are shrunk towards the aggregated templates at higher-levels. In Figure 2.5 we compare the effect of flat and hierarchical shrinkage for 10 data points. For flat shrinkage (2nd row) we computed a convex combination that uses a weight of 0.6 for the samples and a weight of 0.4 for the global mean. In order to perform a fair comparison we chose the penalty strength for hierarchical shrinkage (3rd row) such that the amount of regularization (measured in terms of squared deviations from the original samples) is the same as for flat shrinkage. The difference between the two regularization techniques is that with flat

Figure 2.5: Flat and hierarchical shrinkage for Caltech faces. 1st row: Training examples. 2nd&4th row: Smoothed templates when using flat shrinkage with sample weights of 0.6 and 0.3, respectively. 3rd&5th row: Smoothed templates when using hierarchical shrinkage with corresponding amounts of regularization.

shrinkage the templates are pulled towards the global mean while with hierarchical shrinkage they are pulled towards the underlying manifold. For example, a face under dark lighting conditions (as in the 5th column) becomes much brighter when being pulled towards the global mean. With hierarchical shrinkage however it is pulled towards the average of faces under dark lighting conditions, i.e., it remains relatively dark. Moreover, unusual artifacts are smoothed out much faster with hierarchical shrinkage. For example, the hand in front of the face (8th column) is still clearly visible when flat shrinkage is used but disappeared almost entirely with hierarchical shrinkage. To further illustrate the differences we also considered flat shrinkage (4th row) using a weight of 0.3 for the samples and a weight of 0.7 for the global mean. The resulting templates are visually still very similar to the original

samples while the lighting conditions are quite homogeneous. Hierarchical shrinkage with an equivalent amount of regularization (5th row) leads to more generic templates, i.e., a higher abstraction from the observed samples is achieved. In particular, individual characteristics are smoothed out (consider for example the 9th&10th column).

Without regularization a Gaussian product mixture trained on this dataset achieves an average test log-density of -109,854 nats. The optimal number of mixture components is 20, i.e., one component per every 20 samples is used. A kernel density estimator trained on this dataset without regularization but with tuned bandwidth achieves an average test log-density of -107,040 nats. Using the same bandwidth a kernel density estimator with hierarchical shrinkage achieves an average test log-density of -97,369 nats. The optimal bandwidth for the regularized estimator is actually somewhat smaller than for the standard kernel density estimator because the smoothed templates are closer to the manifold compared to the training examples. Readjustment of the bandwidth of our estimator leads to an average test log-density of -95,986 nats. For comparison, a kernel density estimator with flat shrinkage and optimized bandwidth obtains an average test log-density of -99,308 nats. While the optimal amounts of regularization for flat and hierarchical shrinkage are similar, the optimal bandwidth for flat shrinkage is somewhat larger than for hierarchical shrinkage.

2.8.4 Decision Trees*

We now exemplify the usefulness of our hierarchical shrinkage procedure for smoothing regression trees. In Figure 2.6 we consider noisy observations from a simple regression function. We trained a regression tree until all leaves were pure, i.e., each leaf only contained a single training sample. Using this tree to define the hierarchical clustering we performed shrinkage using an L1 penalty and an L2 penalty, respectively. With the L1 penalty the smoothed regression tree looks similar to a shallow regression tree (of depth say 3). With the L2 penalty the smoothed regression tree is comparable to a collection of 1,000 bagged regression trees but actually has a somewhat better predictive performance since it is more robust to outliers.

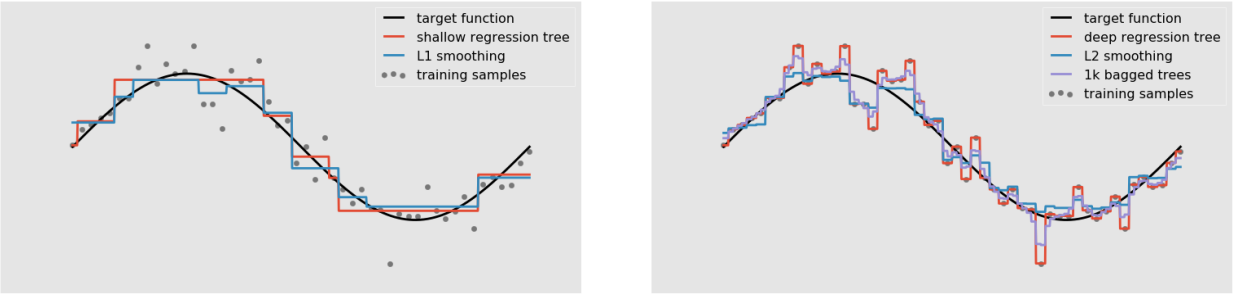

Figure 2.6: Smoothing of a regression tree using an L1 (left) and L2 (right) penalty, respectively.

Note that a flat shrinkage procedure would simply pull the estimates in each leaf towards the global average of the response. In particular, all leaf values would be changed in the same direction.

The hierarchical shrinkage procedure can also be used to smooth classification trees. In Figure 2.7 we consider a two-class classification problem. The samples for both classes are drawn from circular distributions with different radii plus some Gaussian noise. The Bayes-optimal classifier is a perfect circle. A classification tree with tuned depth performs rather poorly. Hierarchical shrinkage applied to a classification tree with pure leaves leads to a substantial improvement and performs even better than 1,000 bagged trees. Again, flat shrinkage would only pull the predicted probabilities closer to the prior class probabilities and would not change the decision boundary.

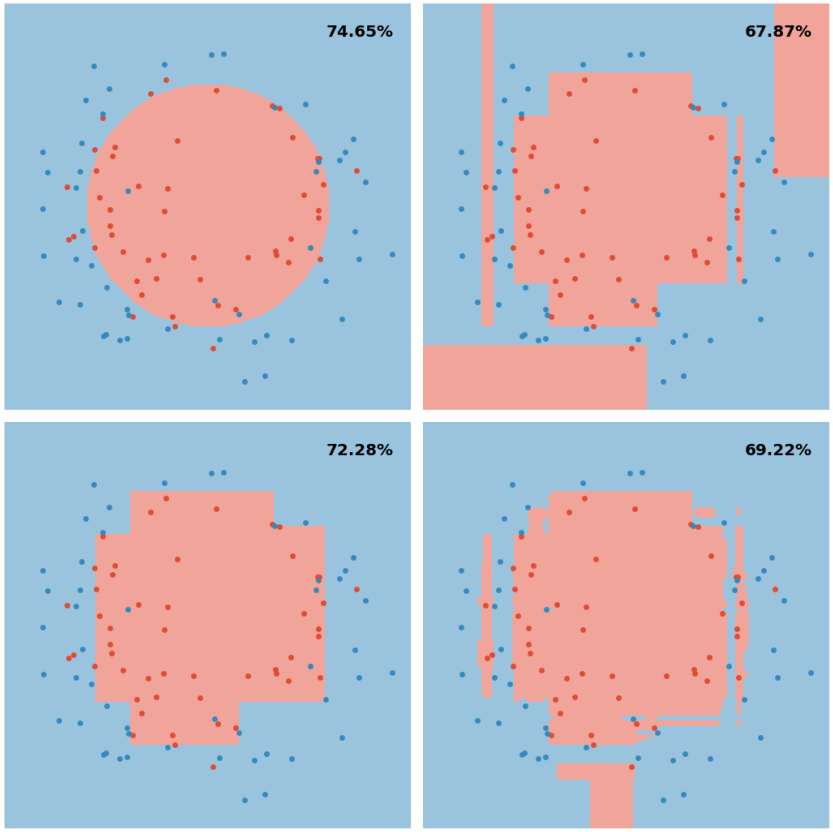

Figure 2.7: Smoothing of a classification tree. Red and blue dots indicate the training examples from the two classes. The red and blue background colors visualize the decision rule. The percentage is the classification accuracy (on the ground-truth distribution). Top-left: Bayes-optimal classifier. Top-right: Classification tree with tuned depth. Bottom-left: Smoothed classification tree. Bottom-right: 1,000 bagged trees.

CHAPTER 3

COMPACT COMPOSITIONAL MODELS

In recent years, multi-layer network architectures have drastically improved the discriminative performance in several classification tasks. These deep networks can be constructed based on desired invariances and stability properties [19] or they can be learned from usually large datasets as a cascade of nonlinear functions [e.g., 66]. While being effective for classification the used transformations are typically high-dimensional and individual features are not always semantically meaningful. Moreover, the described structures are sometimes more global than desired and often there are many features which describe similar structures. As pointed out by Bengio et al. [11] learning to disentangle the factors of variation remains a key challenge in deep networks.

Even for simple image classes, the basic task of learning a compact and interpretable representation has not yet been solved in a satisfactory manner. For example, the most natural representation of the letter T is in terms of a vertical and a horizontal bar. Consequently, this class can efficiently be represented by six coordinates, corresponding to the row/column location and the orientation of the bars. In this work, we will learn such a representation from as little as 10 examples. Apart from being intuitively appealing, low-dimensional explicit representations can be useful in order to obtain coarse descriptions for scene analysis tasks. We learn a robust representation by seeking a parsimonious set of experts corresponding to the largest stable structures in the data. We emphasize that the focus is not on achieving state-of-the-art performance in terms of classification rates or likelihoods but rather on learning simple models from few examples. The key is to define a composition rule for experts which causes strong competition over the domain. In Section 3.1 we review various ways of composing experts for binary data and discuss their impact on the resulting representation. A new rule based on extremal competition is then introduced which is particularly well suited for learning compact representations. The rule discourages experts from learning similar structures and penalizes opposing opinions strongly so that

abstaining from voting becomes more attractive. By this we formally mean that identical experts cannot be used to improve likelihoods and opposing opinions always lead to a reduction in likelihood. In Section 3.2 we present an appropriate sequential inference procedure for the type of compositional models which we consider. In Section 3.3 we describe a batch as well as online version of an EM-style learning procedure for the new composition rule. Moreover, we propose a sequential initialization method which can be described as a process of oversimplification and correction. Results for a synthetic dataset and handwritten letters are presented in Section 3.4.

3.1 Composition Rules

We model binary data $\boldsymbol{x} \in \{0,1\}^D$ through a product Bernoulli distribution $\mathbb{P}(\boldsymbol{x} \,|\, \boldsymbol{\mu})$, i.e., the variables $\boldsymbol{x}(d)$ are assumed to be conditionally independent given $\boldsymbol{\mu}$. The global template $\boldsymbol{\mu}$ is a composition of experts $\boldsymbol{\mu_1}, \ldots, \boldsymbol{\mu_K}$ where each expert is a Bernoulli template $\boldsymbol{\mu_k} \in [0,1]^D$ on the whole domain. The way in which the experts are combined in order to create the composed template is specified through a *composition rule* (also known as patchwork operation, mixing function or voting scheme). Such rules are the generative counterpart of activation functions in feed-forward neural networks. Formally, a composition rule is a function $\gamma\colon [0,1]^K \to [0,1]$ with a varying number K of arguments. The composed template $\boldsymbol{\mu}$ is obtained by applying the composition rule to the expert "opinions" for each dimension

$$\boldsymbol{\mu}(d) = \gamma(\boldsymbol{\mu_1}(d), \ldots, \boldsymbol{\mu_K}(d)).$$

Of special interest is the ability of an expert to abstain from voting. By that we mean there exists a value $q \in [0,1]$ such that

$$\gamma(q, p_1, \ldots, p_K) = \gamma(p_1, \ldots, p_K)$$

for all $p_1, \ldots, p_K \in [0, 1]$ and all K.

In the following we consider two classes of models. The first class is a natural choice for binary images and is referred to as write-black models [95]. In these models the default variable state is "off" (white pixel). Underlying factors are now able to turn variables "on" (black pixel). If multiple causes for a variable are present the state will still be "on". This type of model is appropriate, for example, for figure-ground segmentations (white corresponding to background and black corresponding to foreground) where experts describe object parts. Composition rules for write-black models encode "no vote" through $\boldsymbol{\mu_k}(d) = 0$ and we refer to them as asymmetric rules. Note that the value 0 is used for regions far away from the expert support. A template probability $\boldsymbol{\mu_k}(d) = 1/2$ on the other hand is used for variables which are close to the boundary of the support and can hence be interpreted as "not sure". In write-black models, samples from individual experts will like actual object parts.

The second class we consider are write-white-and-black models [95]. In such models experts are able to cast votes in favor of "on" as well as "off". Composition rules for write-white-and-black models encode abstention through $\boldsymbol{\mu_k}(d) = 1/2$ and we refer to them as symmetric rules. Note that "not sure" is also encoded through $\boldsymbol{\mu_k}(d) = 1/2$, so this value can have either of the two meanings. Samples from single experts for image data will not look like actual object parts because about half of the pixels in the background region will be turned on.

3.1.1 Asymmetric Rules

Noisy Or

A straightforward composition rule for write-black models is the disjunctive composition

$$\gamma(p_1, \ldots, p_K) = 1 - \prod_{k=1}^{K} (1 - p_k).$$

The composed probability is simply the probability of observing at least one success when drawing independently from Bernoulli distributions with probabilities $p_1, \ldots, p_K$. This rule was used by Saund [95].

Sum of Odds

It was argued by Dayan and Zemel [27] that the noisy-or rule offers little incentive for experts to focus on different structures in the data. A more competitive composition rule was therefore proposed which is of the form

$$\gamma(p_1, \ldots, p_K) = 1 - \left(1 + \sum_{k=1}^{K} \frac{p_k}{1 - p_k}\right)^{-1}.$$

This rule can be motivated as the probability of observing a success when drawing independently from Bernoulli distributions with probabilities p_k conditioned on observing at most one success. It is easy to see that the composed odds are just the sum of the individual odds. While in a global mixture model exactly one component is responsible for generating the whole data vector, here exactly one expert is responsible for each dimension that is turned "on". So, in contrast to the noisy-or rule the responsibility for individual variables is not shared.

Maximum

The most extreme form of competition is achieved through the max rule

$$\gamma(p_1, \ldots, p_K) = \max_{k=1,\ldots,K} p_k.$$

Such a rule was used by Lücke and Sahani [72]. Since only the strongest template matters, experts have no incentive to represent structures which are already present unless their opinion is the most extreme one. Consequently, experts tend to focus on different aspects

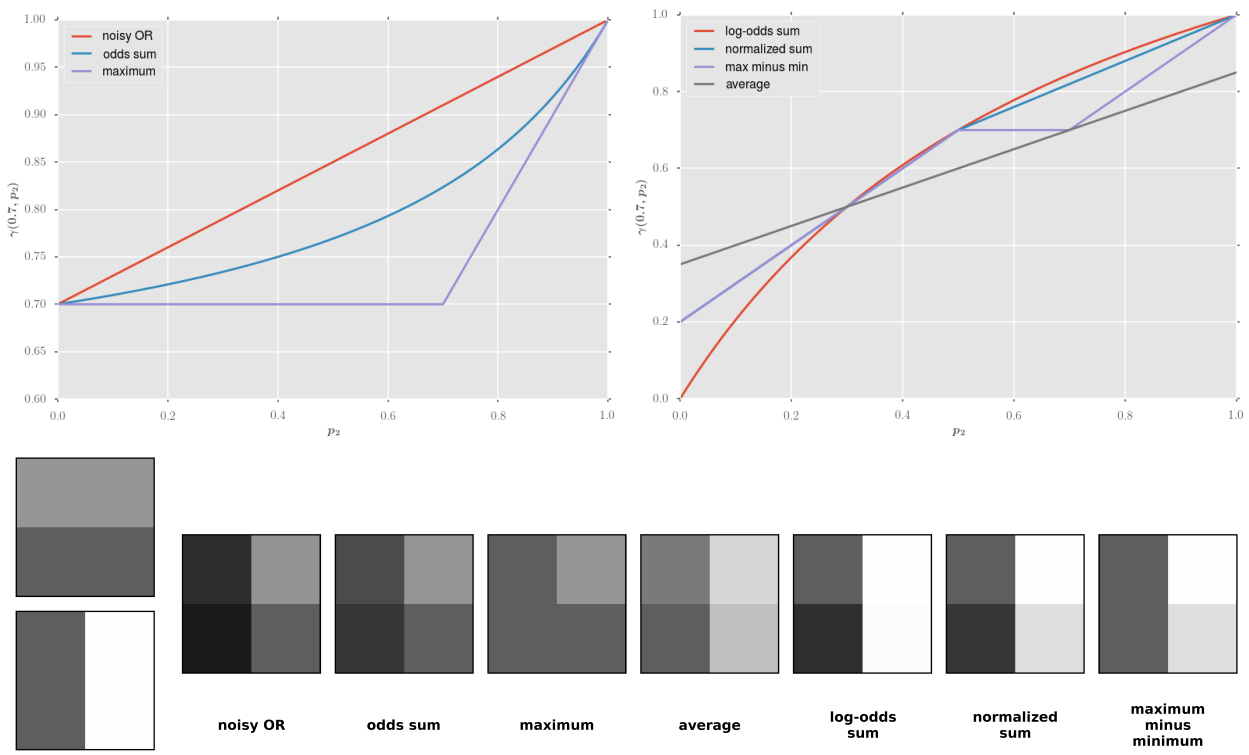

Figure 3.1: Top: Comparison of composition rules, as a function of p_2 for $p_1 = 0.7$. Bottom: Compositions of two experts (the probabilities in the first template are 0.5 and 0.7, the probabilities in the second template are 0.7 and 0.01) using different rules.

of the data. In contrast to the noisy-or and the sum-of-odds rule, likelihoods cannot be improved by using the same expert multiple times. Other than for the sum-of-odds rule, with this composition rule for each variable it is known which expert is responsible. This fact allows one to use an analytic formula in the M-step of the EM learning procedures for such models.

The left panel in Figure 3.1 visualizes the different asymmetric composition rules for two experts. We see that the max rule is flat at 0 and hence inhibits the experts from leaving the "no vote" state.

3.1.2 Symmetric Rules

Arithmetic Mean

An intuitive composition rule for write-white-and-black models is the simple average

$$\gamma(p_1, \ldots, p_K) = \frac{1}{K} \sum_{k=1}^{K} p_k.$$

The composition can be interpreted as an equal mixture of the individual expert templates. This rule was used by Amit and Trouvé [6]. Note however that with this composition rule it is impossible for an expert to abstain from voting. Consequently, the support of the experts has to be restricted manually.

Sum of Log-Odds

An often used composition rule is the sum of log-odds

$$\gamma(p_1, \ldots, p_K) = \sigma \left(\sum_{k=1}^{K} \log \frac{p_k(x)}{1 - p_k(x)} \right)$$

where $\sigma(t) = (1+e^{-t})^{-1}$ is the logistic function (the inverse of the logit function). This type of composition is used in restricted Boltzmann machines [54] and sigmoid belief networks [83]. A sum followed by the logistic link function is also used in generalized linear models for binary data. This includes logistic PCA [96], latent trait models [8], exponential family sparse coding [67] and binary matrix factorization [79]. Abstention is expressed through log-odds of 0, i.e., probabilities of 1/2. Since the first step is to sum up the individual votes (casted in terms of log-odds), opinions of experts voting in the opposite direction can be completely canceled out. Indeed, as seen in Figure 3.1, even if $p_1 = 0.7$ the composed probability approaches 0 as p_2 approaches 0. As a result, there is little pressure to abstain from voting. At the same time, two very similar experts may complement each other without

reducing the likelihood compared to a single strong expert.

Normalized Sum

A more competitive composition rule was proposed by Saund [95], which for active disagreement results in a net uncertainty. For $p_k \in \{0, 1/2, 1\}$ the rule is

$$\gamma(p_1, \ldots, p_K) = \frac{1}{2}\left(\frac{\sum_{k=1}^{K}(p_k - \frac{1}{2})}{\sum_{k=1}^{K}|p_k - \frac{1}{2}|} + 1\right)$$

and linear interpolation is used for values in between. However, due to the computational cost for linear interpolations in K dimensions a more tractable approximation was actually used for the experiments in that work.

Maximum Minus Minimum

We propose a new composition rule which reduces redundancy among experts and incentivizes abstention at the same time. Analogously to the max composition, we would like to rule out the possibility to increase the likelihood just by using the same expert multiple times. This can be achieved by using only the most extreme opinion. On the other hand, similarly to the normalized sum, opposing opinions should result in a limitation of the maximum achievable likelihood. These considerations naturally lead to the max-minus-min rule

$$\gamma(p_1, \ldots, p_K) = q + \left(\max_{k=1,\ldots,K} p_k - q\right)_+ - \left(\min_{k=1,\ldots,K} p_k - q\right)_-$$

where the subscript $+/-$ denotes the positive/negative part of a real number. We set $q = 1/2$ but other values could also be used. To our knowledge, such a composition rule has not been used before.

A comparison of the symmetric composition rules is shown in the right panel of Figure 3.1. To illustrate the difference between the sum of log-odds composition and the max-minus-

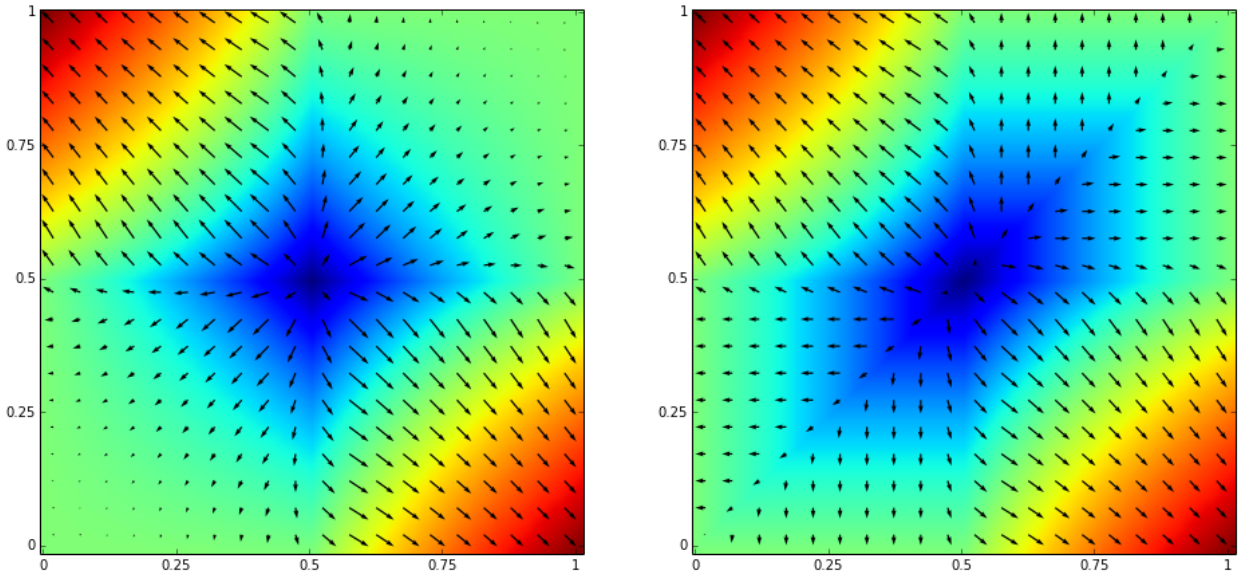

Figure 3.2: Log-likelihood functions for a two-expert model with sum-of-log-odds composition (left) and max-minus-min composition (right).

min composition we compute the corresponding log-likelihood functions in a simple example. Consider the ground-truth model which creates completely white images with probability 1/4, completely black images with probability 1/4 and random images (in which each pixel is drawn independently from a Bernoulli-1/2 distribution) with probability 1/2. We attempt to learn this model by training two experts which can be combined using the sum-of-log-odds or the max-minus-min composition rule, respectively. For simplicity we reduce the expert templates to a single parameter $\boldsymbol{\mu_k}(x) = p_k$, $k = 1, 2$, i.e., each pixel has the same chance of being turned on. We show the resulting (expected) log-likelihood functions in Figure 3.2 in the limit of a large image resolution. There are two equivalent global maxima, one at $(1, 0)$ and one at $(0, 1)$, each of them corresponds to one expert for black images and one expert for white images. If the initial parameters in the sum-of-log-odds model are on the same side of 1/2 (top-right or bottom-left quadrant) gradient descent will change both parameters in the same direction (i.e., it will increase both values or it will decrease both values). This partially explains why restricted Boltzmann machines [54] with randomly initialized weights have a tendency to yield multiple similar experts. The situation is very different for the

max-minus-min composition. If both parameters are on the same side of 1/2 then moving along the gradient direction would only change the more extreme value while the other expert would remain close to the "no vote" state (and would hence be "available" in later stages of the learning process). Note however that we are not using gradient descent to train max-minus-min models but rather employ the EM algorithm (see Section 3.3). The example also suggests that it could be beneficial to use a sequential initialization procedure in which additional experts take care of structures which cannot be explained by the existing ones.

3.2 Sequential Inference

Given experts $\boldsymbol{\mu_k}$ the inference task is to determine the posterior distribution on expert configurations given the observation $\boldsymbol{x}$, or at least finding the expert configuration which is most likely to have generated the data. For our purposes it is important that the inference procedure yields compact representations. We mention a few possible approaches before we present a sequential procedure which we call *likelihood matching pursuit*. In restricted Boltzmann machines [54] experts are evaluated independently. This is computationally efficient but activates all experts that match the data sufficiently well, rather than providing the sparsest possible activation which can explain the data. In other works, the indicator variables for expert presence are relaxed to real values in $[0, 1]$. For example, Dayan and Zemel [27] and Vincent et al. [107] use simple mean-field approximations. Saund [95] uses gradient descent starting from a point with all coordinates equal to 1/2. This iterative procedure becomes inefficient however if the number of possible experts is much larger than the typical number of active experts. Lücke and Sahani [72] use a truncated search which evaluates all expert configurations with a small number of active components.

We propose a simple sequential inference procedure for max and max-minus-min compositions. The data is first explained by the expert $\boldsymbol{\mu} = \boldsymbol{\mu_{k_1}}$ which yields the highest likelihood for the observation $\boldsymbol{x}$. Additional experts $\boldsymbol{\mu_k}$ are then evaluated by forming the composition (according to the chosen rule) of the current global template $\boldsymbol{\mu}$ with the candidate experts

and computing the new likelihood. The best expert is added and the global template is updated accordingly. The procedure ends when the likelihood cannot be improved anymore. This yields a sparse activation in a single sequential pass because experts are only added if they are able to explain structures which have not been explained before. Note that this procedure is quite similar to matching pursuit [75] for sparse coding. Instead of minimizing the squared error, we maximize the Bernoulli likelihood. This sequential fitting works well for write-white-and-black models (using the max-minus-min rule). However, since write-black models (using the max rule) encode abstention through a probability of 0 rather than 1/2, the value of $P(\boldsymbol{x} \,|\, \boldsymbol{\mu_k})$ will heavily depend on which structures other than the one described by $\boldsymbol{\mu_k}$ are present in the observation $\boldsymbol{x}$. Consequently, it is easy to run into situations where

$$P(\boldsymbol{x} \,|\, \boldsymbol{\mu_{k_1}}) > P(\boldsymbol{x} \,|\, \boldsymbol{\mu_{k'_1}})$$

but

$$P(\boldsymbol{x} \,|\, \gamma(\boldsymbol{\mu_{k_1}}, \boldsymbol{\mu_{k_2}}, \ldots, \boldsymbol{\mu_{k_J}})) \ll P(\boldsymbol{x} \,|\, \gamma(\boldsymbol{\mu_{k'_1}}, \boldsymbol{\mu_{k_2}}, \ldots, \boldsymbol{\mu_{k_J}})).$$

Hence, the choice of the first expert may lead to a poor local maximum. The problem is that the first expert tries to explain the entire data vector. As a simple fix we propose to use truncated templates

$$\tilde{\boldsymbol{\mu}}_{\boldsymbol{k}}(d) = \max\left(\frac{1}{2},\, \boldsymbol{\mu_k}(d)\right)$$

instead. This eliminates the impact of the data in the background region of the expert. Indeed, $P(\boldsymbol{x} \,|\, \tilde{\boldsymbol{\mu}}_{\boldsymbol{k}})$ only depends on variables $\boldsymbol{x}(d)$ for those d which satisfy $\boldsymbol{\mu_k}(d) > 1/2$. That means the likelihood of the truncated template only depends on the data in the support of expert k. A similar idea to robustify templates was used by Williams and Titsias [109]. They mix the original templates with a uniform distribution. In the case of binary data this amounts to $\alpha\boldsymbol{\mu}(d) + (1-\alpha)/2$ for some $\alpha \in (0,1)$, i.e., a convex combination between $\boldsymbol{\mu}$ and 1/2 is formed. For a write-black model this is a less effective transformation than our truncation. We illustrate the effectiveness of the proposed modification through a simple

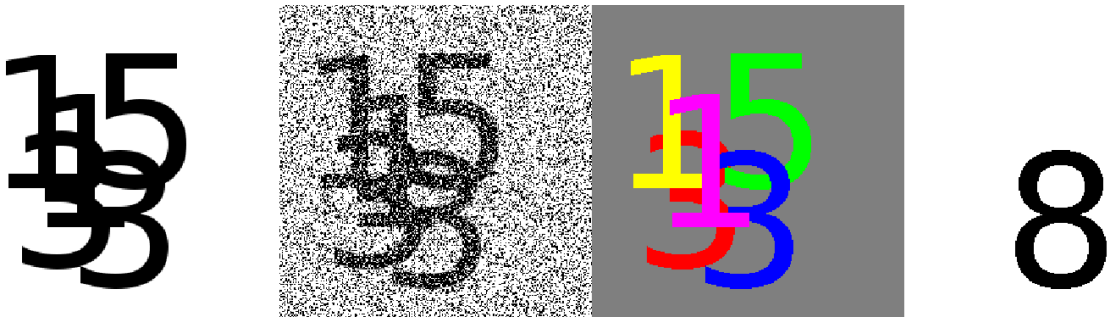

Figure 3.3: 1st panel: Scene to be analyzed. 2nd panel: Noisy version. 3rd panel: Resolved scene using robustified templates (the digits are detected in the order red, green, blue, yellow, magenta). 4th panel: First detected digit using the original templates.

scene analysis example. Five digits are placed at random locations in the image according to a write-black model and the task is to recover the identity and locations of the digits in a noisy version of this scene. Using the robustified templates the scene can be resolved perfectly, as shown in Figure 3.3. However, with the original templates the first digit which is placed down corresponds to the largest structure in the image that can be explained by a single template through a moderately well fit. Note that the suggested fix is not the only possible solution. If we allow for an iterative procedure then the scene can also be resolved with the original templates. The modification is simply meant as a robustification of the greedy inference procedure.

3.3 EM Learning

We now describe an approximate EM procedure [28] for learning max-minus-min models. When working with image data, we explicitly model transformations like shifts and rotations. Each template $\boldsymbol{\mu_k}$ then provides multiple transformed versions $\boldsymbol{\mu_{kt}} = \Phi_t(\boldsymbol{\mu_k})$ where t denotes the transformation. This allows us to share parameters among all transformed versions. Since our templates will describe rather large structures, we assume that only one of the transformed versions is present in each image. In the (hard) E-step we use the likelihood matching pursuit procedure from the previous section which yields for each observation $\boldsymbol{x}$ a

representation $(k_1, t_1), \ldots, (k_J, t_J)$. We define

$$k^\star(d) = \operatorname*{argmax}_{j=1,\ldots,J} \boldsymbol{\mu}_{\boldsymbol{k_j t_j}}(d), \qquad \ell^\star(d) = \operatorname*{argmin}_{j=1,\ldots,J} \boldsymbol{\mu}_{\boldsymbol{k_j t_j}}(d)$$

provided that the maximum is larger than q and the minimum is smaller than q, respectively (otherwise we leave $k^\star(d)$ or $\ell^\star(d)$ undefined). In words, $k^\star(d)$ and $l^\star(d)$ are the active experts with the most extreme opinions for variable $\boldsymbol{x}(d)$. In the M-step we then simply compute

$$\boldsymbol{\mu_k}(d) = \frac{\sum_{n=1}^{N} \mathbb{1}\{k{=}k_n^\star(d) \text{ or } k{=}\ell_n^\star(d)\}\, \Phi_{t_{nk}}^{-1}(\boldsymbol{x_n})(d) + \epsilon}{\sum_{n=1}^{N} \mathbb{1}\{k{=}k_n^\star(d) \text{ or } k{=}\ell_n^\star(d)\} + 2\epsilon} \tag{3.1}$$

where $k_n^\star$, $l_n^\star$ are the maximizers respectively minimizers and t_{nk} is the transformation corresponding to k-th expert in the representation of the n-th training example $\boldsymbol{x_n}$. The pseudocount $\epsilon > 0$ is added for regularization purposes and can be interpreted as a Beta(ϵ, ϵ) prior on the expert probabilities. In all our experiments we use $\epsilon = 1$ which corresponds to a uniform prior. The update formula (3.1) is not the exact maximizer for the expert templates but a very good heuristic. The reason why we can use such a simple analytic expression is that with extremal composition rules the responsibility for individual variables $\boldsymbol{x}(d)$ is not split among many experts. With other composition rules gradient descent methods are often required in order to perform the M-step. Also note that for $q = 0$ the max-minus-min model reduces to the max model. In that special case the update formula (3.1) only involves $k^\star$ but not $\ell^\star$.

3.3.1 Online Learning and Sequential Initialization

It is straightforward to provide an online version of the learning procedure. The M-step updates simply are

$$\boldsymbol{\mu_k}(d) \leftarrow \frac{N_k(d)\boldsymbol{\mu_k}(d) + \mathbb{1}\{k{=}k_n^\star(d) \text{ or } k{=}\ell_n^\star(d)\}\, \Phi_{t_{nk}}^{-1}(\boldsymbol{x_n})(d)}{N_k(d) + \mathbb{1}\{k{=}k_n^\star(d) \text{ or } k{=}\ell_n^\star(d)\}},$$

$$N_k(d) \leftarrow N_k(d) + \mathbb{1}\{k{=}k_n^\star(d) \text{ or } k{=}\ell_n^\star(d)\}.$$

The variable $N_k(d)$ counts how many training examples have been used to compute the current estimate for the d-th dimension of expert k. The online version is attractive because it allows us to use a sequential initialization scheme. Since the learning problem is non-convex, it is crucial to have a good initialization. In accordance with our attempt to learn a parsimonious representation we start off with a single global template derived from the first training example and add more experts later on. The idea is to use "oversimplified" models in the sense that they try to explain new examples through the experts learned so far. The models are then "corrected" by appending additional templates to the collection of experts. Define $\tilde{\boldsymbol{\mu}}_{\boldsymbol{K+1}}(d) = (\boldsymbol{x_n}(d) + \epsilon)/(1 + 2\epsilon)$. When using the max-minus-min composition the additional template can be initialized as

$$\boldsymbol{\mu_{K+1}}(d) = \begin{cases} \frac{1}{2} & \text{if } \mathbb{P}(\boldsymbol{x_n}(d) \,|\, \boldsymbol{\mu}(d)) \geq \mathbb{P}(\boldsymbol{x_n}(d) \,|\, \tilde{\boldsymbol{\mu}}_{\boldsymbol{K+1}}(d)) \\ \tilde{\boldsymbol{\mu}}_{\boldsymbol{K+1}}(d) & \text{otherwise} \end{cases}$$

where $\boldsymbol{\mu}$ is the composed template using the existing experts. This means that the new expert abstains from voting for dimensions which are already well explained by the other

experts. For max compositions we suggest to use

$$\boldsymbol{\mu_{K+1}}(d) = \begin{cases} \frac{1}{2} & \text{if } \mathbb{P}(\boldsymbol{x_n}(d) \,|\, \boldsymbol{\mu}(d)) \geq \frac{1}{2} \\ \tilde{\boldsymbol{\mu}}_{\boldsymbol{K+1}}(d) & \text{otherwise} \end{cases}.$$

This makes sure that in the background region $\boldsymbol{\mu_{K+1}}$ is close to 0 (rather than equal to 1/2). Our successive refinement is in stark contrast to the typical bottom-up grouping of local structures in part-based compositional models. For example, Fidler and Leonardis [38] and Zhu et al. [111] start with elementary edge features and combine simple structures into more complex ones through a hierarchical clustering procedure.

3.3.2 Geometric Component

For image data, the spatial arrangement of the experts is modeled through a joint Gaussian distribution for shifts and rotations. Consequently, learning only requires us to compute the mean and covariance of the training configurations provided by the inference procedure. As emphasized by Bruna and Mallat [19], a very desirable property of a representation is that intraclass deformations are linearized. As the experiment in Section 3.4.2 confirms, our representation transforms the complex deformation orbit (in the original space) into a linear space in which a Gaussian distribution satisfactorily describes the deformations.

3.4 Experiments

3.4.1 A Synthetic Write-White-and-Black Model

We compare the max-minus-min model to denoising autoencoders [107] and restricted Boltzmann machines [54] on a synthetic model for binary images of size 6×6 pixels. The image grid is divided into a regular grid of four quadrants. Each quadrant is independently activated with probability 1/2. An activated quadrant is either entirely black or entirely white,

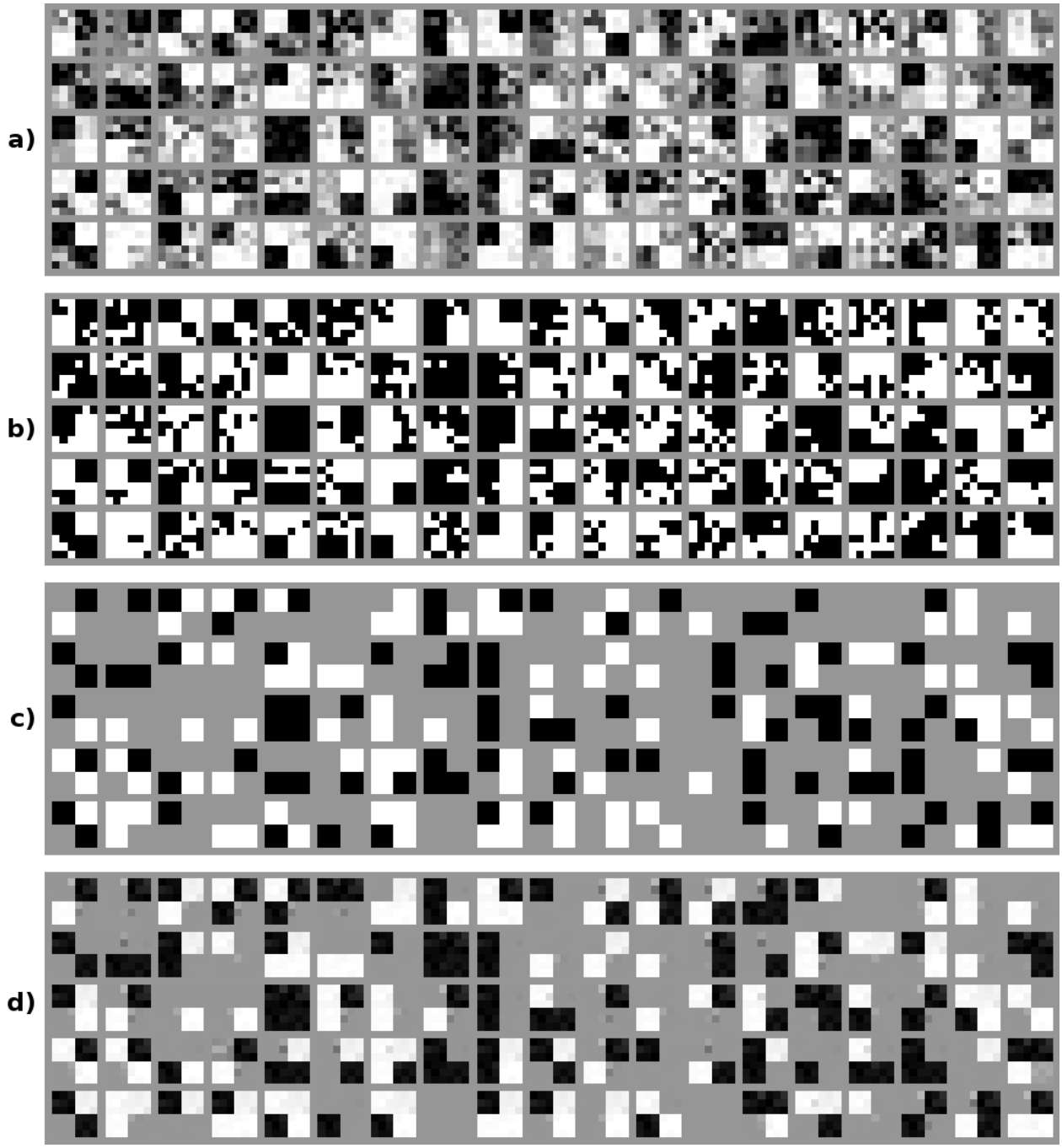

Figure 3.4: a) Sparse coding reconstruction. b) 100 samples from the synthetic model. c) Ground-truth templates for the samples above. d) Reconstruction of the max-minus-min model.

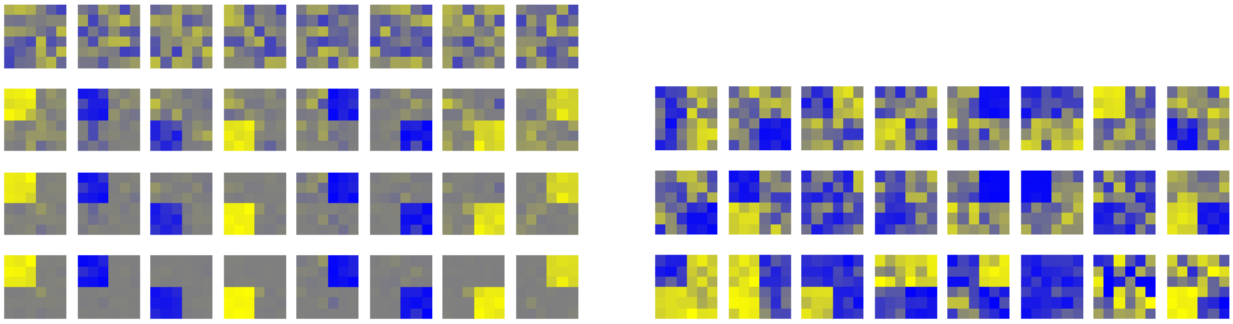

Figure 3.5: Left: Initialization (1st row) and learned experts after 1, 2 and 5 EM iterations (2nd-4th row) for the max-minus-min model trained on 100 examples from the synthetic model. Right: Experts obtained from a denoising autoencoder (top), a restricted Boltzmann machine (center) and dictionary learning for sparse coding (bottom) using 100 samples.

each with probability 1/2. For non-activated quadrants the pixels are drawn independently from a Bernoulli-1/2 distribution (samples from this model can be found in Figure 3.4). The task is to recover the 8 underlying experts corresponding to the four quadrants with two polarities. The learning algorithm for the max-minus-min model typically converged after a few iterations. In the left panel of Figure 3.5 we visualize the obtained templates after 1, 2 and 5 iterations when using 100 training examples. As we see, our model is able to almost perfectly recover the original experts. The denoising autoencoder and the restricted Boltzmann machine were run for 100 epochs and the learning rates were tuned. For the denoising autoencoder we also tuned the corruption level. Figure 3.6 shows a comparison with the max-minus-min model in terms of the cross-entropy (i.e., the negative log-likelihood of the composed templates) on a test set of 1,000 samples. The reconstruction error for the max-minus-min model is much lower. This is because with the sum-of-log-odds composition multiple ground-truth experts are mixed up. The right panel of Figure 3.5 shows that the templates are indeed combinations of multiple ground-truth experts, i.e., the factors of variation have not been disentangled successfully. In order to make a fair comparison we have left out the sequential initialization procedure (as well as the transformation modeling) and initialized all models with completely random templates. That means the performance gain is in fact due to the more competitive expert interaction. As a further comparison we

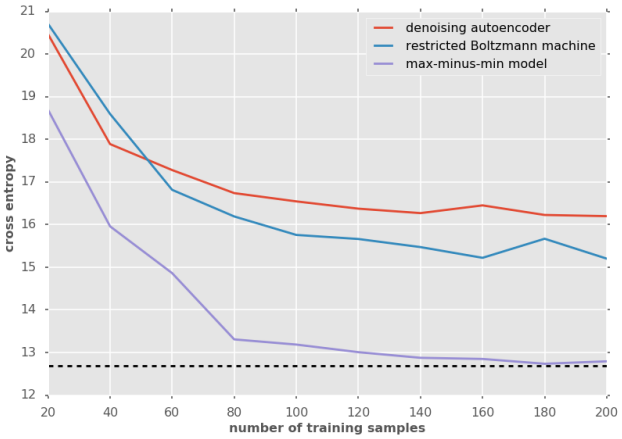

Figure 3.6: Cross-entropy reconstruction error for different models and training sizes (lower is better). The dashed black line is the cross-entropy of the ground-truth model.

learned a sparse coding dictionary [73]. Note that the data is then treated as real-valued and the task is to find basis vectors whose linear combinations allow for good reconstructions in a squared error sense. The basis vectors learned from 100 samples are visualized in the right panel of Figure 3.5. Again, the ground-truth experts are not recovered. However, the basis vectors look more structured than the experts learned by denoising autoencoders or restricted Boltzmann machines. In terms of L_2 reconstruction error this dictionary is actually even better than the ground-truth generative model. Figure 3.4 visualizes the reconstructions obtained via sparse coding and via the learned max-minus-min model. The sparse coding reconstruction is visually closer to the data but much more noisy than the reconstruction of our model. Indeed, the max-minus-min reconstruction is almost identical to the ground-truth templates. The reason for this different behavior is that the squared error is more forgiving for small deviations compared to cross-entropy and penalizes harder than cross-entropy if the deviation is large (around 1/2).

3.4.2 Handwritten Letters

We now train experts on the letter classes from the TiCC handwritten characters dataset [106] while modeling shifts and rotations. The images are of size 56×56 pixels and each

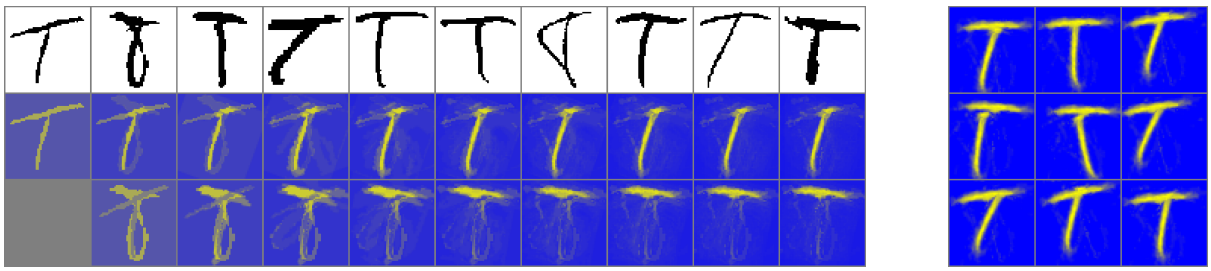

Figure 3.7: Online learning for the letter T. Left panel: The 10 samples used for training (1st row) and the two expert templates (2nd&3rd row) at step $i = 1, \ldots, 10$ (blue/yellow corresponds to probabilities of 0/1). Right panel: 9 sampled expert configurations based on a multivariate Gaussian distribution of the spatial expert arrangement using the final templates.

sample provides a label for the writer of the corresponding character. Since "on" pixels in this dataset correspond to black ink it is natural to use the max composition rule. The purpose of this experiment is to illustrate the effectiveness of our sequential initialization procedure from Section 3.3.1. The left panel in Figure 3.7 visualizes the online learning process for the letter T, using the first sample of the first 10 writers. The first expert is initialized by the first training example. The second expert is initialized by the characteristics of the second example which cannot be explained through the first expert. Every additional image then updates both experts. After 10 examples the learning process has converged to a vertical and a horizontal bar. Samples (Figure 3.7, right panel) from the learned spatial distribution look realistic and cover the principal deformations of the class. We also learned up to four experts for the other[1] letters, using the first sample of the first 20 writers. The results are shown in Figure 3.8. Most experts correspond to natural elements of the character class. Note that in contrast to Lake et al. [62] no motion information was necessary to learn these character primitives.

1. The letter X is missing from the dataset.

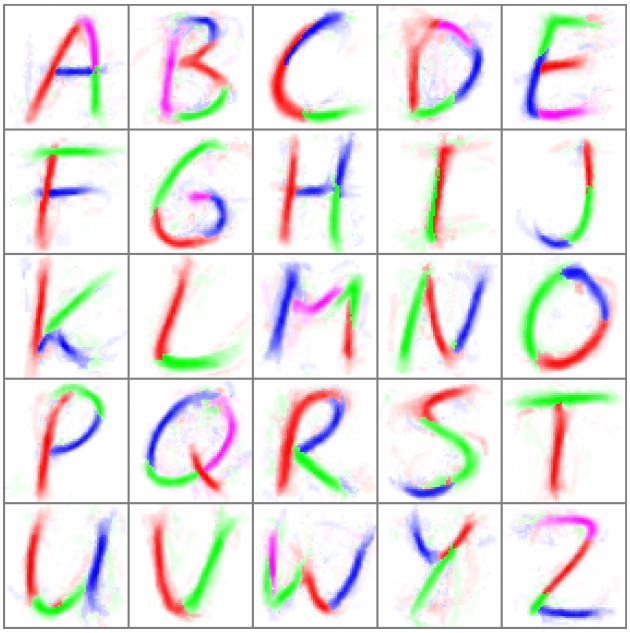

Figure 3.8: Experts learned from 20 examples per class. Each expert is plotted at its mean location with mean orientation. For each pixel the color (red, green, blue, magenta) indicates the maximum expert and the intensity visualizes the template value (white corresponding to 0, color corresponding to 1).

CHAPTER 4

DYNAMIC PARTITION MODELS

In a distributed representation a collection of experts $k = 1, \ldots, K$ is used to explain the data points $\boldsymbol{x}$. Specifically, each latent state $\boldsymbol{h}$ corresponds to a distribution on the data space. The goal is to obtain a good conditional likelihood of the data given the latent variables. If the encoding is binary then the latent state simply specifies which experts are active. A popular model family with latent binary representation are products of experts [54]. In such a model individual distributions $\mathbb{P}_k$ are multiplied together and renormalized

$$\mathbb{P}_{\mathrm{PoE}}(\boldsymbol{x} \mid \boldsymbol{h}) = \frac{\prod_{k:\boldsymbol{h}(k)=1} \mathbb{P}_k(\boldsymbol{x})}{\sum_{\boldsymbol{x'}} \prod_{k:\boldsymbol{h}(k)=1} \mathbb{P}_k(\boldsymbol{x'})}.$$

Computing the normalizing constant is in general intractable though. A special case in which it is possible are restricted Boltzmann machines [54]. In these models the experts are product Bernoulli distributions with templates $\boldsymbol{\mu_k} \in [0,1]^D$. The composed distribution is then also a product Bernoulli distribution

$$\mathbb{P}_{\mathrm{PoE}}(\boldsymbol{x} \mid \boldsymbol{h}) = \prod_{d=1}^{D} \boldsymbol{\mu}_{\mathrm{PoE}}(d)^{\boldsymbol{x}(d)} (1 - \boldsymbol{\mu}_{\mathrm{PoE}}(d))^{1-\boldsymbol{x}(d)}$$

where the composed template is

$$\boldsymbol{\mu}_{\mathrm{PoE}}(d) = \sigma\left(\sum_{k:\boldsymbol{h}(k)=1} \boldsymbol{w_k}(d) \right).$$

Here, $\boldsymbol{w_k}(d) = \log(\boldsymbol{\mu_k}(d)/(1 - \boldsymbol{\mu_k}(d)) \in \mathbb{R}$ are the log-odds of the experts and $\sigma(t) = (1 + \exp(-t))^{-1}$ is the logistic function. A related model are autoencoders [e.g., 64], which can be considered as a mean-field approximation of restricted Boltzmann machines that uses latent variables $\boldsymbol{h}(k)$ with values in $[0, 1]$. The sum-of-log-odds composition arises naturally from

generalized linear models for binary data because the log-odds are the canonical parameter of the Bernoulli family. However, it is not the only possible choice. Various other composition rules are discussed in Chapter 3. The main problem with summing up the log-odds is that the corresponding learning procedures often yield many similar and highly dependent experts. For more details on this problem see Chapter 3.

In this Chapter we propose a new approach for learning experts which are individually meaningful and which have disjoint responsibilities. We start by presenting the basic idea in Section 4.1 and introduce a smoothed model which is easier to train. In Section 4.2 and 4.3 we present inference and learning procedures for our model. The extension to continuous data is discussed in Section 4.4. Experiments on binary and real-valued data are performed in Section 4.5.

4.1 Model Definition

The main idea is to use expert opinions $\boldsymbol{\mu_k} \in [0,1]^D$ with associated levels of expertise $\boldsymbol{e_k} \in \mathbb{R}_+^D$. For each dimension the composed template then simply consists of the expert opinion with the highest level of expertise

$$\tilde{\boldsymbol{\mu}}(d) = \boldsymbol{\mu}_{\boldsymbol{k^\star(d)}}(d), \quad k^\star(d) = \operatorname*{argmax}_{k:\boldsymbol{h}(k)=1} \boldsymbol{e_k}(d).$$

Consequently, each variable $\boldsymbol{x}(d)$ is explained by only a single expert. The partitioning into expert supports $S_k(\boldsymbol{h}) = \{d \in \{1,\ldots,D\} : k^\star(d){=}k\}$ is determined dynamically based on the latent configuration $\boldsymbol{h}$. We hence call our model a *dynamic partition model*. One advantage of this model over approaches which combine multiple experts for individual variables is that the estimation of the expert opinions is trivial. Indeed, the maximum-likelihood estimate for $\boldsymbol{\mu_k}(d)$ is simply the average over all observations $\boldsymbol{x_n}(d)$ for which the k-th expert was

responsible

$$\widehat{\boldsymbol{\mu}}_{\boldsymbol{k}}(d) = \frac{\sum_{n=1}^{N} \mathbb{1}\{k_n^\star(d)=k\}\boldsymbol{x_n}(d)}{\sum_{n=1}^{N} \mathbb{1}\{k_n^\star(d)=k\}}. \tag{4.1}$$

Here, $k^\star(d)$ denotes the expert with the highest level of expertise $\boldsymbol{e_k}(d)$ among all k with $\boldsymbol{h_n}(k) = 1$. In contrast to that, models that sum up expert votes require some kind of regression during the training phase.

A related composition rule was studied by Lücke and Sahani [72]. In their model the composed template is simply the maximum of the individual templates $\boldsymbol{\mu_k}$. This rule is only useful in special cases where the presence of at least one causal factor implies that $\boldsymbol{x}(d) = 1$. A generalization was considered in Chapter 3 where the composed probability depends on the maximum and the minimum of the expert probabilities $\boldsymbol{\mu_k}(d)$. While the motivation for that rule was similar, our maximum-expertise rule is more principled and can be applied to continuous data.

Using the maximum-expertise composition rule makes it is easy to learn expert opinions, however the log-likelihood is not a continuous function of $\boldsymbol{e_k}$. We hence consider a smoother variant for which the levels of expertise are easier to train. Maximum likelihood estimation for the smoothed model will converge to a dynamic partition model. Rather than using the opinion corresponding to the highest level of expertise, we compute a weighted average of the opinions $\boldsymbol{\mu_k}(d)$ where the weight is proportional to the level of expertise $\boldsymbol{e_k}(d)$. Thus, the smoothed composition rule is

$$\boldsymbol{\mu}(d) = \sum_{k:\boldsymbol{h}(k)=1} \boldsymbol{r_k}(d)\boldsymbol{\mu_k}(d), \quad \boldsymbol{r_k}(d) = \frac{\boldsymbol{e_k}(d)}{\sum_{k':\boldsymbol{h}(k')=1} \boldsymbol{e_{k'}}(d)}.$$

Note that in contrast to a classical mixture model [e.g., 77] we use a different mixture for each dimension. The weight $\boldsymbol{r_k}(d)$ is the degree of responsibility of expert k for dimension d and depends on the latent state $\boldsymbol{h}$. An expert with a medium level of expertise assumes

full responsibility if no other reliable expert is present and takes on a reduced degree of responsibility otherwise. This is different from the sum of log-odds composition for which the votes $\boldsymbol{w_k}$ are not adjusted depending on the latent state. Decomposing votes into opinions and levels of expertise also avoids ambiguities. For example, a vote $\boldsymbol{w_k}(d) \approx 0$ could mean that $\boldsymbol{\mu_k}(d) \approx 1/2$ or that $\boldsymbol{e_k}(d) \approx 0$.

Amit and Trouvé [6] used a simple average (i.e., an equal mixture) of the individual templates. With such a composition rule all experts are equally responsible for each of the variables and hence specialization on local structures is not possible. To circumvent this problem, in that work $\boldsymbol{e_k}(d)$ was manually set to 1 for some subset of the dimensions (depending on a latent shift variable) and to 0 otherwise.

In a product of experts, the variance of the composition is typically smaller than the variance of the experts. For example, in a restricted Boltzmann machine this is the case if for a fixed dimension d the votes $\boldsymbol{w_k}(d)$ have the same sign. As a consequence, there is a tendency to employ many (potentially very similar) experts for each dimension. Even with an L1-penalty on the votes $\boldsymbol{w_k}$ the responsibility for individual variables $\boldsymbol{x}(d)$ is typically still shared among many experts. This is because under the constraint $\sum_k \boldsymbol{w_k}(d) = \boldsymbol{w}(d)$ the quantity $\sum_k |\boldsymbol{w_k}(d)|$ is minimized whenever $\boldsymbol{w_k}(d)$ has the same sign for all k.

On the other hand, the variance of a mixture is always larger than the average variance of the components (this can be seen from the total variance formula). Consequently, the likelihood of the model is maximized if only one expert is used for each variable. Note that the overall variance of the composed template $\boldsymbol{\mu}$ is of course smaller than for any of the expert templates. This is because the experts only have small variance for some subset of the variables while the variance of the other variables is large.

4.2 Inference

In the inference step we try to find for each data point $\boldsymbol{x_n}$ the subset of experts which maximizes $P(\boldsymbol{x_n} \,|\, \boldsymbol{h_n})$. We suggest to sequentially add the expert which most improves

the likelihood. This approach is called *likelihood matching pursuit* and was used already in Chapter 3. The greedy search works well for our model because we are working with a small set of experts which focus on very different structures in the data. Indeed, with compact representations the posterior distribution on the latent variables given $\boldsymbol{x_n}$ is often highly peaked at a state $\boldsymbol{h_n^\star}$.

For comparison, the inference procedure for restricted Boltzmann machines independently activates experts based on their inner product with the data point. In particular, not just the most probable expert configuration is determined but the whole posterior distribution on latent states given the data is explored. This is necessary because in models with sum-of-log-odds composition various different latent states could have generated the given data point since many similar experts exist.

4.3 Learning

We use an EM-style algorithm [28] to learn our model. The sequential inference procedure is used as the E-step and provides the latent representations $\boldsymbol{h_n}$ for the data points $\boldsymbol{x_n}$. The corresponding responsibilities when using these experts are denoted by $\boldsymbol{r_{nk}}$ and we define $\boldsymbol{r_{nk}} = 0$ if $\boldsymbol{h_n}(k) = 0$. In the M-step we have to update the experts opinions $\boldsymbol{\mu_k}$ and the levels of expertise $\boldsymbol{e_k}$.

The log-likelihood

$$\sum_{n=1}^{N} \left[\boldsymbol{x_n}(d) \log \left(\sum_{k=1}^{K} \boldsymbol{r_{nk}}(d) \boldsymbol{\mu_k}(d) \right) + (1 - \boldsymbol{x_n}(d)) \log \left(1 - \sum_{k=1}^{K} \boldsymbol{r_{nk}}(d) \boldsymbol{\mu_k}(d) \right) \right]$$

is a concave function of $\boldsymbol{\mu_k}(d)$, see Section 4.6. We could therefore in principal use Newton's method. There are a few complications though. One problem is that the second derivative is proportional to the squared responsibility and hence close to 0 if the level of expertise is small. Consequently, template updates in regions with low expertise would be unstable. To deal with that we could add a penalty on the squared log-odds for example. Another

problem is that the Newton steps may lead to probability estimates outside of $[0, 1]$. This can be dealt with by pulling the estimates back into the interval. Note that working on the log-odds scale is not possible because the log-likelihood of our model is not convex in the expert log-odds. Since maximum likelihood estimation for expertise-weighted compositional models converges to dynamic partition models, we use a variant of the update equation (4.1) as a simple, fast and robust heuristic instead of Netwon's method. Concretely, we employ a responsibility-weighted average of the training samples

$$\boldsymbol{\mu_k}(d) = \frac{\sum_{n=1}^{N} \boldsymbol{r_{nk}}(d)\boldsymbol{x_n}(d) + \epsilon\boldsymbol{\mu_0}}{\sum_{n=1}^{N} \boldsymbol{r_{nk}}(d) + \epsilon}.$$

For stability we added a term that shrinks the updated template towards some target $\boldsymbol{\mu_0}$ if the total responsibility of the expert is small. In our experiments we set $\boldsymbol{\mu_0}$ to the average over all training examples. Note that such an update rule would not work for sum-of-log-odds compositions because the composed probabilities have to be equal to the empirical frequencies and not the individual expert probabilities.

We now turn to the updates of the levels of expertise. The log-likelihood is a rather complex function of $\boldsymbol{e}_k$. Using gradient descent is problematic because the derivatives with respect to $\boldsymbol{e_k}$ can have very different scales which makes it difficult to choose an appropriate learning rate and hence the convergence could be slow. Moreover, exact optimization is not necessary because at the time of convergence only the order of the levels of expertise matters. Consequently, we propose to adjust $\boldsymbol{e_k}(d)$ based on the sign of the gradient. We simply multiply or divide the current value by a constant C. If the gradient is very close to 0 we leave $\boldsymbol{e_k}(d)$ unchanged. For all our experiments we used $C = 2$. Larger values can speed up the convergence but sometimes lead to a somewhat worse optimum.

In the learning procedure we perform the expertise update first. We then recompute the responsibilities using these new levels of expertise and update the expert opinions. The EM

algorithm typically converges after about 10 iterations.

4.4 Continuous Data

For continuous data $\boldsymbol{x} \in \mathbb{R}^D$ we again model the expert composition through a responsibility-weighted mixture. However, rather than working with mixtures of Gaussians we use a single Gaussian distribution whose mean is equal to the mixture mean and whose variance is equal to the mixture variance. To avoid confusion with the logistic function we denote the expert variances by $\boldsymbol{v_k}$ and the composed variance by $\boldsymbol{v}$. According to the total variance formula the composition rule for the variance can be expressed as

$$\boldsymbol{v}(d) = \sum_{k:\boldsymbol{h}(k)=1} \boldsymbol{r}_k \boldsymbol{v_k} + \sum_{k:\boldsymbol{h}(k)=1} \boldsymbol{r}_k (\boldsymbol{\mu_k} - \boldsymbol{\mu})^2 = \sum_{k:\boldsymbol{h}(k)=1} \boldsymbol{r}_k (\boldsymbol{v_k} + \boldsymbol{\mu_k^2}) - \boldsymbol{\mu}^2.$$

Note that simply using a responsibility-weighted average of the variances would not be appropriate because two experts with the same variance but different means would then be equivalent to a single expert with the average mean. In order to encourage compact representations it is important to have an increase in the variance in such a situation.

The learning procedure for the expert means and the levels of expertise is exactly the same as before. For the variance update we again use a responsibility-weighted average

$$\boldsymbol{v_k}(d) = \frac{\sum_{n=1}^{N} \boldsymbol{r_{nk}}(d)(\boldsymbol{x_n}(d) - \boldsymbol{\mu_k}(d))^2 + \epsilon \boldsymbol{v_0}}{\sum_{n=1}^{N} \boldsymbol{r_{nk}}(d) + \epsilon}$$

where $\boldsymbol{v_0}$ is the average variance over all training samples.

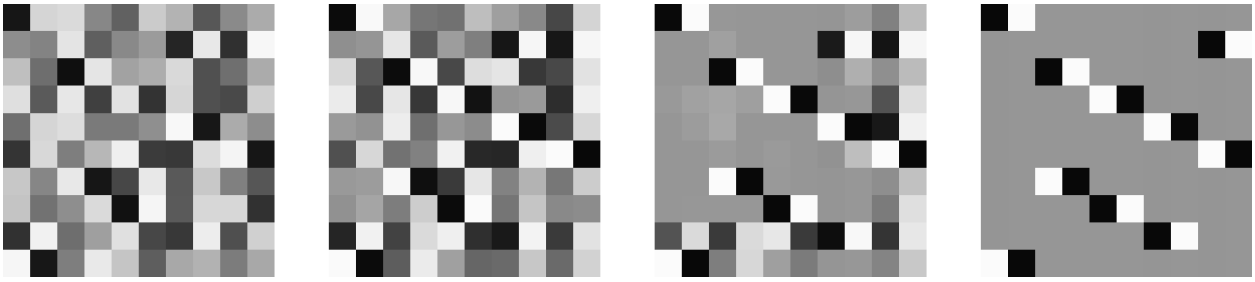

Figure 4.1: Trained experts for the synthetic dataset. Each panel shows the probabilities (white/black corresponds to $\boldsymbol{\mu_k}(d) = 0/1$) of the 10 experts (rows) for the 10 dimensions (columns). 1st panel: Random initialization. 2nd-4th panel: Our learning procedure after 3, 5 and 15 iterations.

4.5 Experiments

4.5.1 Synthetic Data

We start by running the synthetic experiment from the thesis introduction to show that our learning procedure is capable of disentangling factors of variation. Let us recall the setup. We consider the uniform distribution on the 32-element subset $\{(0,1),(1,0)\}^5 \subset \{0,1\}^{10}$. There are five factors of variation corresponding to the state of the pairs $(\boldsymbol{x}(d), \boldsymbol{x}(d+1))$ for $d = 1,3,5,7,9$ with the two factor levels $(0,1)$ and $(1,0)$. It is possible to perfectly reconstruct all data points as a dynamic partition using $5 \cdot 2 = 10$ experts. Here, the total number of experts we need happens to be equal to the number of dimensions. However, in other cases the number of experts could be smaller or larger than D. Using the 32 elements as our training set we ran our learning algorithm for 15 iterations starting from a random initialization of the experts. The resulting templates after 3, 5 and 15 iterations are shown in Figure 4.1. We see that each of the final experts specializes in exactly two dimensions d and $d+1$. Its opinion for these variables are close to 0 and 1 respectively while the opinion for the remaining variables is about 1/2. Every data point can now be (almost) perfectly reconstructed by using exactly five of these experts.

For comparison we trained various other models with 10 experts which use a sum of log-odds composition. We started with an autoencoder [e.g., 64] which in principle could adopt

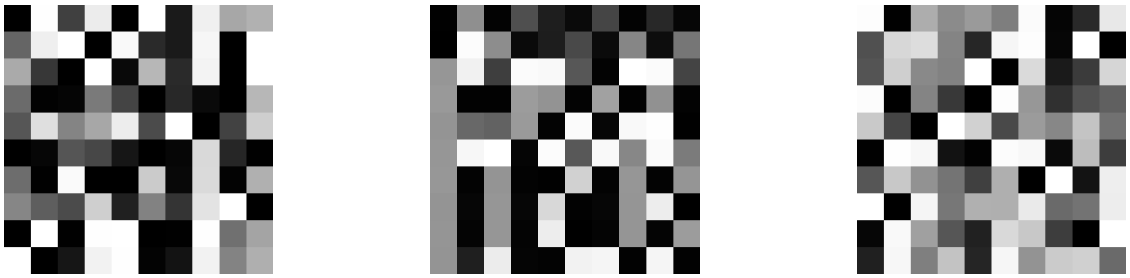

Figure 4.2: Autoencoder (1st panel), sparse dictionary (2nd panel) and restricted Boltzmann machine (3rd panel) after 1,000 iterations.

the identity map because it uses (in contrast to our model) a bias term for the observable and latent variables. However, the gradient descent algorithm with tuned learning rate yielded a different representation (Figure 4.2, 1st panel). While the errors of the reconstructions are rather low, they are not perfect and the factors of variations have not been disentangled. Next, we considered a dictionary [e.g., 31] with a sparse representation. The sparsity penalty was adjusted so that the average number of active dictionary elements was around 5. The learning algorithm again yielded highly dependent experts (Figure 4.2, 2nd panel). Finally, we trained a restricted Boltzmann machine through batch persistent contrastive divergence [103] using a tuned learning rate. Note that a restricted Boltzmann machine in principle only requires 5 experts to model the data appropriately because it uses bias terms. However, we again learned 10 experts which are shown in the 3rd panel of Figure 4.2. While the results look better than for the previous 2 models they are still far from optimal.

4.5.2 MNIST Digits

We now consider the MNIST digits dataset [65] which consists of 60,000 training samples and 10,000 test samples of dimension $28 \times 28 = 784$. We ran our learning algorithm for 10 iterations and trained 100 experts. The obtained experts are shown in Figure 4.3. We see that some experts specialize on local structures while others focus on more global ones. In Figure 4.4 we visualize the inference procedure for some test samples using these 100 learned experts. On average 12 experts were employed for each data point. For easier

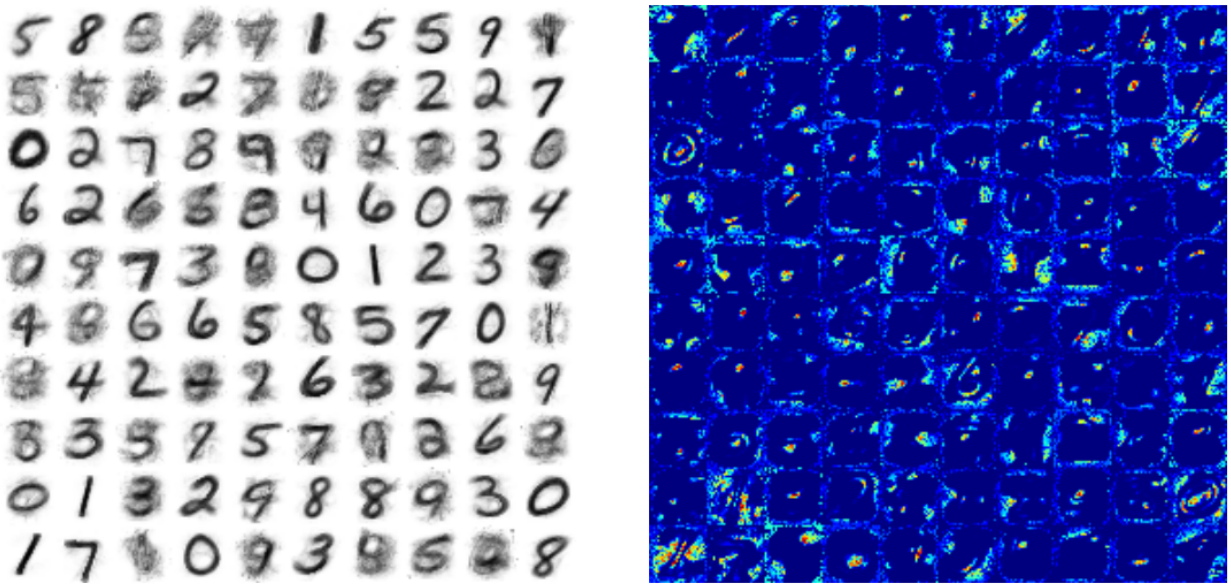

Figure 4.3: Trained experts for MNIST digits. Left: Expert probabilities (white/black corresponds to $\boldsymbol{\mu_k}(d) = 0/1$). Right: Levels of expertise (blue/red corresponds to small/large values).

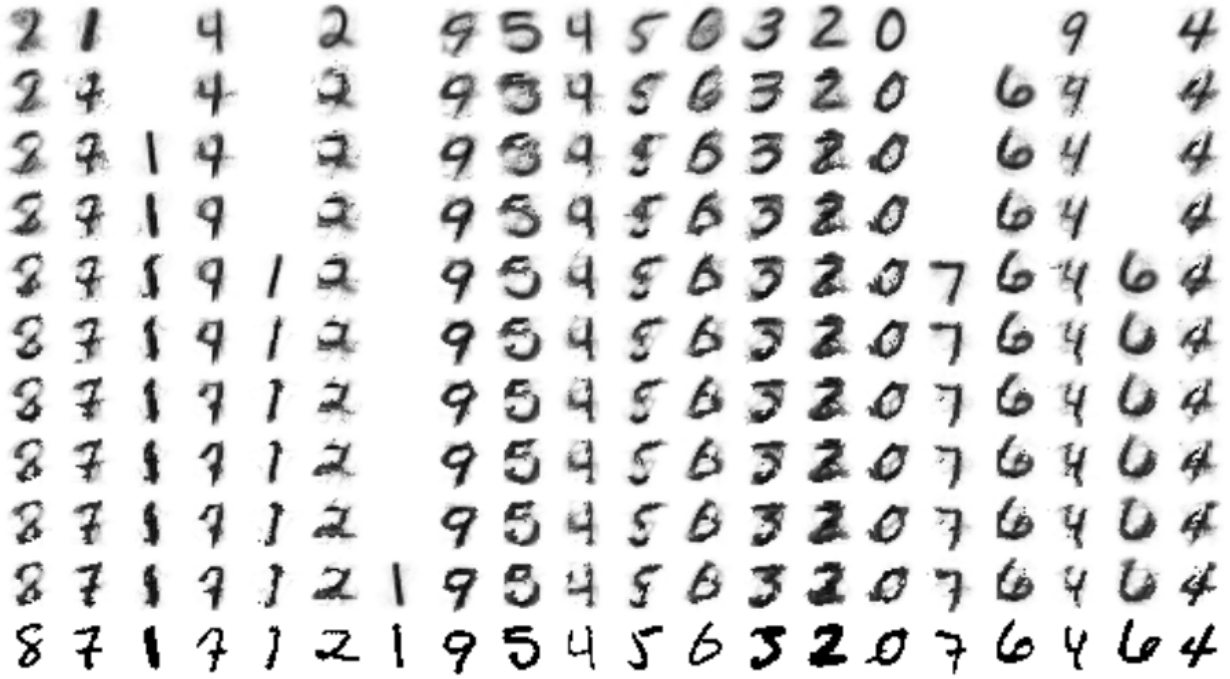

Figure 4.4: Reconstruction of MNIST test examples using likelihood matching pursuit. Each column visualizes the composed Bernoulli templates during the sequential inference procedure (top to bottom) for one sample . The last row are the original data points.

Figure 4.5: Dynamic supports for five MNIST experts. Left column: Expert probabilities. Remaining columns: Composed Bernoulli templates for 10 latent configurations. The cast opinion of the expert is shown in shades of red (white/red corresponds to $\boldsymbol{\mu_k}(d) = 0/1$).

visualization we show at most 10 iterations of the likelihood matching pursuit algorithm. The reconstructions are overall accurate and peculiarities of the samples are smoothed out. To illustrate the dynamically adapting expert supports we show in Figure 4.5 the cast opinion of some experts for different latent representations. Depending on which other experts are present the support can vary quite a bit.

4.5.3 Weizmann Horses

The following experiment shows the ability of our model to cope with very high-dimensional data. The Weizmann horse dataset [15] consists of 328 binary images of size 200×240. We used the first 300 images for training and the remaining 28 images for testing. We trained 20 experts which are visualized in Figure 4.6. Some of the experts are responsible for the background and the central region of the horse while the other experts focus on local structures like head posture, legs and tail. In Figure 4.7 we illustrate the partitioning of the test examples into expert opinions. For simplicity we used exactly four experts to reconstruct each sample. Not all characteristics of the samples are perfectly reconstructed but the general pose is correctly recovered. The same dataset was used to evaluate the shape Boltzmann machine [32] where 2,000 experts were learned. For those experiments the images were downsampled to 32×32 pixels. This is a factor 50 smaller than the full resolution of

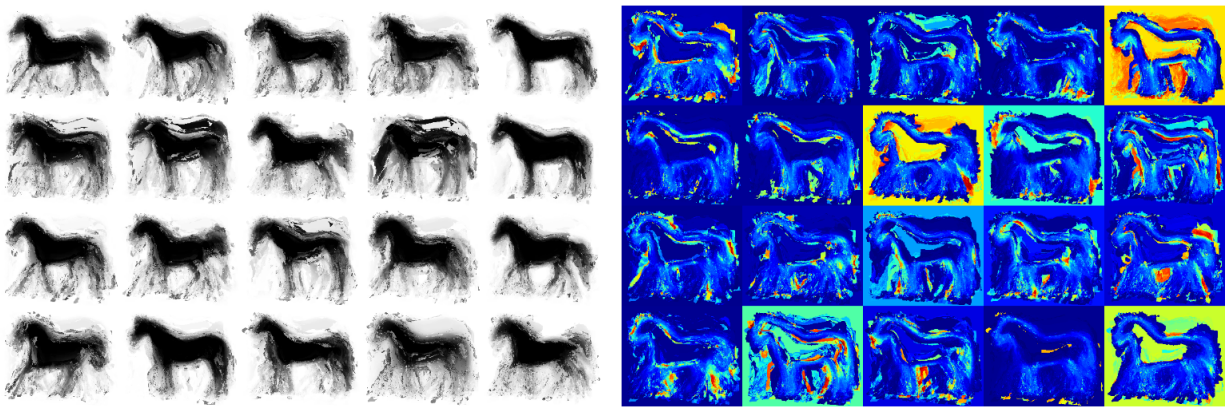

Figure 4.6: Trained experts for Weizmann horses. Left: Expert probabilities (white/black corresponds to $\boldsymbol{\mu_k}(d) = 0/1$). Right: Levels of expertise (blue/red corresponds to small/large values).

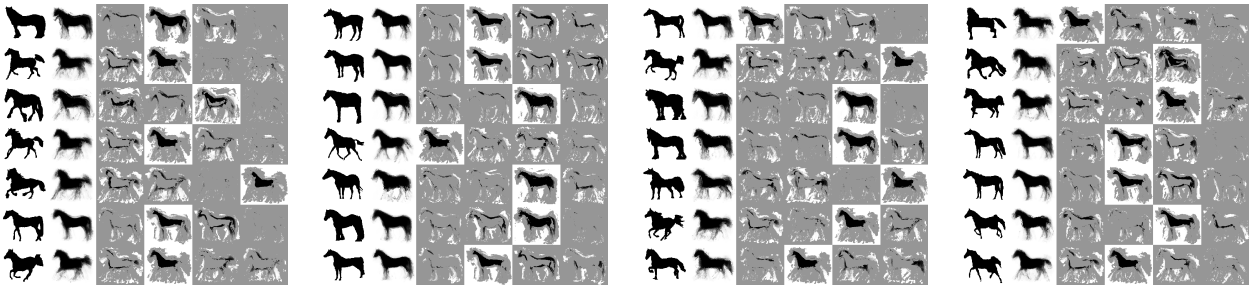

Figure 4.7: Decomposition of the test examples from the Weizmann horse dataset. 1st column: Original data points. 2nd column: Reconstructions (shown are the composed Bernoulli templates). 3rd-6th column: Partitioning into experts opinions (white/black corresponds to $\boldsymbol{\mu_k}(d) = 0/1$, gray indicates regions for which the expert is not responsible).

48,000 dimensions which we use.

4.5.4 Caltech Motorcycles

We also experimented with real-valued data using the Caltech-101 motorcycle dataset [34] which consists of 798 images of size 100×180. The first 750 images were used for training and the remaining 48 images for testing. We trained 50 experts by running our learning procedure for 10 iterations. Figure 4.8 visualizes the reconstructed test examples. The reconstructions are a bit blurry since we use a fairly sparse binary representation. For each data point on average 7 experts were employed. Note that the shapes of the motorcycles are

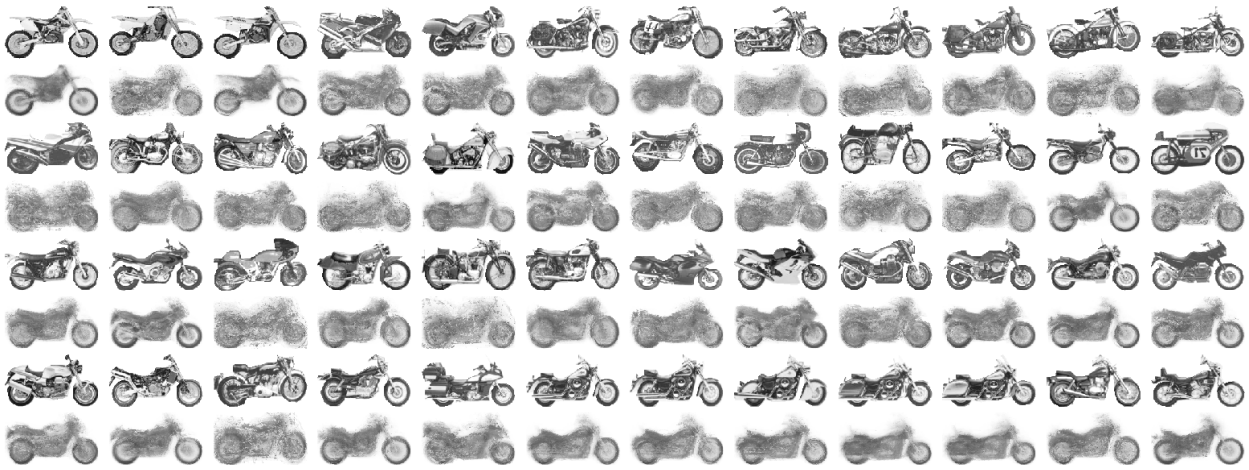

Figure 4.8: Reconstructions of the test examples from the Caltech motorcycle dataset. Odd rows: Original data. Even rows: Reconstructions (shown are the composed Gaussian means).

reconstructed quite accurately.

4.6 Derivatives

We provide here the derivatives of the log-likelihood with respect to the expert parameters.

4.6.1 Bernoulli Model

The Bernoulli log-likelihood is

$$f(\mu) = x \log \mu + (1 - x) \log(1 - \mu).$$

The composition rule for the probability is

$$\mu = \sum_k r_k \mu_k, \quad r_k = \frac{e_k}{\sum_{k'} e_{k'}}.$$

Derivatives With Respect to the Composed Probability

The first and second derivative of the log-likelihood with respect to the composed probability are

$$\frac{df}{d\mu} = \frac{x}{\mu} - \frac{1-x}{1-\mu} = \frac{x-\mu}{\mu(1-\mu)},$$

$$\frac{d^2 f}{d\mu^2} = -\frac{x}{\mu^2} - \frac{1-x}{(1-\mu)^2} = -\frac{(x-\mu)^2}{\mu^2(1-\mu)^2} = -\left(\frac{df}{d\mu}\right)^2.$$

Derivatives With Respect to the Expert Probability

The first and second derivative of the composed probability with respect to the expert probability are

$$\frac{d\mu}{d\mu_k} = r_k, \quad \frac{d^2\mu}{d\mu_k^2} = 0.$$

Consequently, the derivatives of the log-likelihood with respect to the expert probability are

$$\frac{df}{d\mu_k} = \frac{df}{d\mu} \cdot \frac{d\mu}{d\mu_k} = r_k \frac{x-\mu}{\mu(1-\mu)},$$

$$\frac{d^2 f}{d\mu_k^2} = \frac{d^2 f}{d\mu^2} \cdot \left(\frac{d\mu}{d\mu_k}\right)^2 + \frac{df}{d\mu} \cdot \frac{d^2\mu}{d\mu_k^2} = -r_k^2 \frac{(x-\mu)^2}{\mu^2(1-\mu)^2}.$$

We see that $d^2 f / d\mu_k^2 \leq 0$ for $\mu \in (0, 1)$, i.e., the log-likelihood is a concave function of μ_k.

Derivative With Respect to the Level of Expertise

The derivative of the composed probability with respect to the level of expertise is

$$\frac{d\mu}{de_k} = \frac{\mu_k E - \sum_{k'} e_{k'} \mu_{k'}}{E^2} = \frac{\mu_k - \mu}{E}$$

where $E = \sum_{k'} e_{k'}$. The derivative of the log-likelihood with respect to the level of expertise can be computed as

$$\frac{df}{de_k} = \frac{df}{d\mu} \cdot \frac{d\mu}{de_k}.$$

4.6.2 Gaussian Model

The Gaussian log-likelihood is

$$f(\mu, v) = -\frac{(x-\mu)^2}{2v} - \frac{1}{2}\log(v) - \frac{1}{2}\log(2\pi).$$

The composition rule for the mean and variance is

$$\mu = \sum_k r_k \mu_k, \quad v = \sum_k r_k(v_k + \mu_k^2) - \mu^2, \quad r_k = \frac{e_k}{\sum_{k'} e_{k'}}.$$

Derivative With Respect to the Composed Mean and Variance

The derivative of the log-likelihood with respect to the composed mean and variance are

$$\frac{df}{d\mu} = \frac{x-\mu}{v}, \quad \frac{df}{dv} = \frac{(x-\mu)^2}{2v^2} - \frac{1}{2v} = \frac{(x-\mu)^2 - v}{2v^2}.$$

Derivative With Respect to the Level of Expertise

The derivative of the composed mean and variance with respect to the level of expertise are

$$\frac{d\mu}{de_k} = \frac{\mu_k E - \sum_{k'} e_{k'}\mu_{k'}}{E^2} = \frac{\mu_k - \mu}{E},$$

$$\frac{dv}{de_k} = \frac{q_k E - \sum_{k'} e_{k'} q_{k'}}{E^2} - 2\mu\frac{d\mu}{de_k} = \frac{q_k - q}{E} - 2\mu\frac{\mu_k - \mu}{E} = \frac{v_k - v + (\mu_k - \mu)^2}{E}$$

where $E = \sum_{k'} e_{k'}$ and $q_k = v_k + \mu_k^2$, $q = v + \mu^2$. The derivative of the log-likelihood with respect to the level of expertise can be computed as

$$\frac{df}{de_k} = \frac{df}{d\mu} \cdot \frac{d\mu}{de_k} + \frac{df}{dv} \cdot \frac{dv}{de_k}.$$

CHAPTER 5

MIXTURES OF SPARSE AUTOREGRESSIVE NETWORKS

So far we assumed conditional independence of the observable variables $\boldsymbol{x}(d)$ given the latent variables. This simplification allows for very efficient computations but also limits the generative performance of the models. In the following two chapters we consider autoregressive networks which make use of the decomposition

$$\mathbb{P}(\boldsymbol{x}) = \prod_{d=1}^{D} \mathbb{P}(\boldsymbol{x}(d) \,|\, \boldsymbol{x}(1{:}d{-}1)).$$

Learning the joint distribution then reduces to estimating D univariate conditionals. Such a representation is known as a fully-connected left-to-right graphical model [9]. Likelihood evaluations and exact sampling are straightforward with this factorization. Note that the decomposition requires us to chose an ordering of the variables. In this chapter (with the exception of Section 5.2.1) we simply use the ordering in which the data is provided.

Many different models for the conditional distributions in autoregressive networks have been proposed. We discuss a few approaches in Section 5.1. The focus in this chapter is on sparse conditional models. That means the conditional distribution of $\boldsymbol{x}(d)$ given $\boldsymbol{x}(1{:}d{-}1)$ actually only depends on a small subset of the predictors. In Section 5.2 we consider some special cases of sparse autoregressive networks with very small numbers of predictors for each variable, including tree-structured networks and grid dependence structures for image data. We then describe how more general sparse dependence structures can be learned through L1-penalized logistic and linear regressions. Sparse linear/logistic autoregressive networks have excellent generalization abilities and require only relatively small datasets for training. To increase the capacity of the models we consider in Section 5.3 large mixtures of such networks and discuss ways to share parameters among mixture components. In Section 5.4 we extend this approach through a new architecture based on a partitioning of the variables. In Section 5.5 we present quantitative and qualitative results for several experiments.

5.1 Autoregressive Networks

For the overview in this section we assume that we are dealing with binary data $\boldsymbol{x} \in \{0,1\}^D$.

A simple approach [24] to learn the conditional distributions is to perform a greedy search for a set $\text{par}_d \subset \{1,\dots,d{-}1\}$ of predictors for each dimension d. The conditional probabilities $P(\boldsymbol{x}(d) \,|\, \boldsymbol{x}(\text{par}_d))$ can then simply be estimated through the corresponding empirical frequencies. However, since the number of parameters that have to be learned grows exponentially in the number of predictors only very small subsets of $\{1,\dots,d{-}1\}$ can be used. Otherwise most parent configurations will be unobserved and the corresponding conditional probabilities cannot be estimated.

Another way to learn the conditionals is to use decision trees [18]. When being employed for probabilistic modeling they are also known as probability estimation trees [87]. In a tree the probabilities are modeled through piecewise-constant functions

$$\mathbb{P}(\boldsymbol{x}(d){=}1 \,|\, \boldsymbol{x}(1{:}d{-}1)) = \sum_{j=1}^{J} p_j \mathbb{1}(\boldsymbol{x}(1{:}d{-}1) \in R_j)$$

where the regions R_j partition the predictor space $\{0,1\}^{d-1}$ and $p_j \in (0,1)$. Since the covariates are binary, each region is defined by specifying the value of some of the variables $\boldsymbol{x}(1),\dots,\boldsymbol{x}(d{-}1)$. Probability estimation trees can be learned greedily by starting with a single region and recursively dividing the region which leads to the largest improvement in terms of likelihoods. Decision trees as models for conditional distributions have been used for example in [17, 43]. While being simple, the performance of decision trees is often unsatisfactory because shallow trees have a large bias and deep trees have a large variance.

A different approach is to approximate the conditional distributions through parametric models. For example, Frey [39] considered logistic autoregressive networks, which are also called fully visible sigmoid belief nets. In these networks the log-odds of $\boldsymbol{x}(d) = 1$ is modeled

as a linear function of $\boldsymbol{x}(1{:}d{-}1)$, i.e.,

$$\mathbb{P}(\boldsymbol{x}(d){=}1 \,|\, \boldsymbol{x}(1{:}d{-}1)) = \sigma(\alpha_1 \boldsymbol{x}(1) + \ldots + \alpha_{d-1} \boldsymbol{x}(d-1))$$

where $\sigma(t) = (1 + \exp(-t))^{-1}$ is the logistic function. If the true conditional distributions are highly nonlinear then the performance of these simple networks is limited.

A recently very popular model family are neural autoregressive networks [9] which represent the conditional distributions $\mathbb{P}(\boldsymbol{x}(d) \,|\, \boldsymbol{x}(1{:}d{-}1))$ through neural networks. In these models a different set of hidden units is used for each data dimension. Larochelle and Murray [63] proposed to use the same hidden units for all conditionals. Such a weight sharing among conditional distributions leads to a significant improvement in terms of the generalization performance [10, 104] but makes it hard to parallelize the learning procedure. Several variants of this idea have been proposed [105, 89, 51]. Since the hidden units in these models are deterministic, the associated learning algorithms are relatively simple and exact likelihood evaluations are feasible. Neural networks with stochastic hidden units [52, 16] on the other hand require sophisticated inference procedures and exact likelihood evaluations are then intractable. Model complexity in neural networks is commonly controlled through early stopping. Other typical regularization techniques are weight decay [104, 105, 89], dropout [51] or adaptive weight noise [52]. Weight decay means that the L2-norm of the model parameters is penalized. None of these regularization methods yields sparse parameters. In Section 5.5 we perform a quantitative comparison between neural autoregressive networks and mixtures of sparse autoregressive networks.

The conditional distributions in autoregressive networks can also be modeled through LogitBoost [40]. This means that probabilities are estimated through boosted trees [42], which are collections of relatively shallow decision trees. The bias-variance tradeoff for boosted trees is often much better than for a single decision tree. Autoregressive networks based on LogitBoost are the topic of the last chapter of this thesis.

5.2 Sparse Autoregressive Networks

For high-dimensional problems it is very reasonable to assume that only a small number of the variables $\boldsymbol{x}(1), \ldots, \boldsymbol{x}(d-1)$ are relevant for predicting $\boldsymbol{x}(d)$. We present a few approaches based on this assumption and compare their performance.

5.2.1 Tree-Structured Networks

A simple way to introduce dependence among variables is to learn a tree structure. That means a graphical model [e.g., 59] is used in which each variable has (at most) one parent. The tree distribution which is closest in Kullback-Leibler divergence [61] to the empirical distribution is called a Chow-Liu tree model [22]. To learn such a model one can simply run Kruskal's maximum-weight spanning tree algorithm [60] on the pairwise mutual information of the variables. Note that in this case the ordering of the variables is determined as part of the learning process.

5.2.2 Networks With Grid Dependence Structure

For image data it is customary to use grid dependence structures which allow one to model local smoothness. A popular model are Markov random fields [49, 69]. In these models the joint distribution is defined in terms of local characteristics. Specifically, a model of the form

$$\mathbb{P}(\boldsymbol{x}) = \frac{1}{Z} \prod_{C} \phi_C(\boldsymbol{x}(C))$$

is used where C are the maximal cliques (fully connected subsets of vertices) in an undirected graph. Most commonly, 4- or 8-neighborhood systems are employed which means that each pixel is connected to its four respectively eight nearest neighbors. With four neighbors the maximal cliques are just horizontal and vertical edges. With eight neighbors the maximal cliques are quadruples $\{(i-1, j-1), (i-1, j), (i, j-1), (i, j)\}$ where $i = 2, \ldots, H$, $j = 2, \ldots, W$

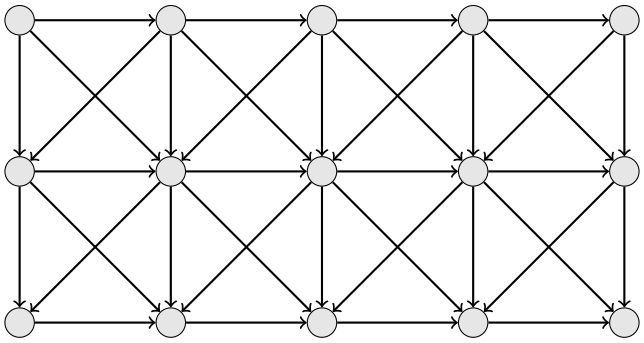

Figure 5.1: Grid dependence structure with four parents. Arrows are pointing from the parents to the children.

and H and W are the height and width of the image, respectively. The problem with this approach is that the normalizing constant Z (also known as the partition function) is typically inaccessible. Consequently, maximum-likelihood estimation and exact likelihood evaluations are intractable. Furthermore, only approximate sampling procedures can be used.

With directed graphical models [e.g., 29] these difficulties are eliminated because the joint distribution is the product of all conditional distributions given the parents in an acyclic directed graph. In Markov mesh models [1] the parents of a pixel (i, j) are all in the top-left quadrant with origin at (i, j). For instance, in a second-order Markov mesh model the parents are $(i-1, j)$ and $(i, j-1)$. In a third-order Markov mesh model there is an additional third parent $(i-1, j-1)$. Both models factorize over cliques $\{(i-1, j-1), (i-1, j), (i, j-1), (i, j)\}$ and are hence special Markov random fields with 8-neighborhood system. Since all factors are normalized, evaluation of likelihoods and sampling are straightforward in Markov mesh models. However, the unilateral dependence structure (flow of causality from the top-left corner to the bottom-right corner) limits the performance of such models. Cressie and Davidson [25] considered models with more sophisticated dependence structures, called partially ordered Markov models. Typically, a raster scan [e.g., 110] (or lexicographic) ordering is used. That means the ordering of the pixels is $(1, 1), (1, 2), \ldots, (1, W), (2, 1), \ldots, (H, W)$. The adjacent lower neighborhood for each pixel with respect to this ordering consists of the top-left, top, top-right and left neighbor (if they exist). Figure 5.1 visualizes this dependence

structure. The corresponding image model is

$$\mathbb{P}(\boldsymbol{x}) = \prod_{i=1}^{H} \prod_{j=1}^{W} \mathbb{P}(\boldsymbol{x}(i,j) \,|\, \boldsymbol{x}(i{-}1,j{-}1), \boldsymbol{x}(i{-}1,j), \boldsymbol{x}(i,j{+}1), \boldsymbol{x}(i,j{-}1)).$$

For binary image data $\boldsymbol{x_n} \in \{0,1\}^{H \times W}$ it is easy to learn the model parameters. We simply have to specify the conditional Bernoulli probability $\boldsymbol{\mu_{ij}}(y_1, y_2, y_3, y_4)$ of $\boldsymbol{x}(i,j) = 1$ for each configuration (y_1, y_2, y_3, y_4) of the parents. Note that the model has (roughly) 16 times as many model parameters as a Bernoulli model. The conditional probabilities can be estimated by counting co-occurrences of the variables. Specifically, using a Beta(ϵ, ϵ) prior with $\epsilon > 0$ for regularization the natural estimator is

$$\hat{\boldsymbol{\mu}}_{\boldsymbol{ij}}(y_1, y_2, y_3, y_4) = \frac{D}{E} \tag{5.1}$$

for $y_1, \ldots, y_4 \in \{0,1\}$ where

$$\begin{aligned} D = \#\{n : \boldsymbol{x_n}(i{-}1,j{-}1){=}y_1,\ \boldsymbol{x_n}(i{-}1,j){=}y_2,& \\ \boldsymbol{x_n}(i{-}1,j{+}1){=}y_3,\ \boldsymbol{x_n}(i,j{-}1){=}y_4,\ \boldsymbol{x_n}(i,j){=}1\} + \epsilon,& \\ E = \#\{n : \boldsymbol{x_n}(i{-}1,j{-}1){=}y_1,\ \boldsymbol{x_n}(i{-}1,j){=}y_2,\ \boldsymbol{x_n}(i{-}1,j{+}1){=}y_3,\ \boldsymbol{x_n}(i,j{-}1){=}y_4\} + 2\epsilon.& \end{aligned}$$

Since in grid models the relevant parents for each pixel consist of only a few adjacent pixels it is impossible to capture long-range dependencies. This often leads to visible artifacts in samples form such models (see Section 5.2.4).

5.2.3 Networks With L1-Penalized Linear Conditionals

As we will see in the examples in Section 5.2.4 using a larger number of parents for each variable can substantially improve the generative performance. In order to be able to learn such dependence structures we have to restrict the functional form of the conditional proba-

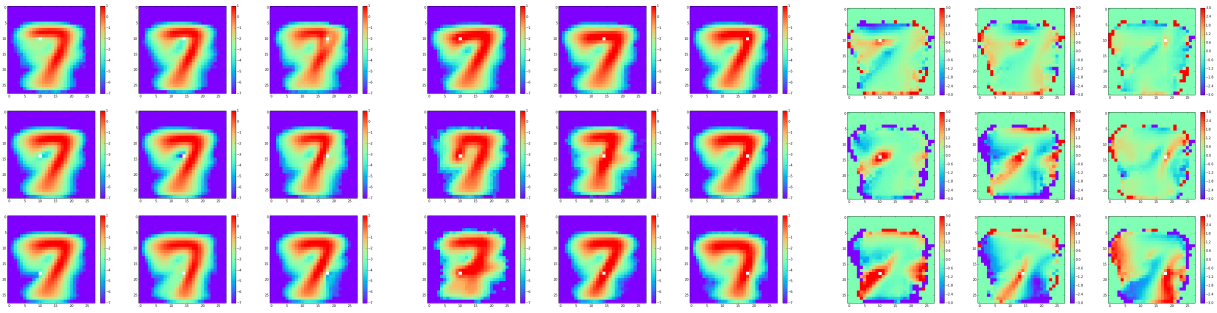

Figure 5.2: Dependencies between variables for handwritten samples of the digit 7. Left: Empirical log-odds $\log[\mathbb{P}(\boldsymbol{x}(d)=1 \,|\, \boldsymbol{x}(d')=0) \,/\, \mathbb{P}(\boldsymbol{x}(d)=0 \,|\, \boldsymbol{x}(d')=0)]$ of examples with $\boldsymbol{x}(d)'=0$ for $d' \in \{10, 14, 18\} \times \{10, 14, 18\}$. Center: The same for examples with $\boldsymbol{x}(d')=1$. Right: Logarithm of the likelihood ratios $\mathbb{P}(\boldsymbol{x}(d)=1 \,|\, \boldsymbol{x}(d')=1) \,/\, \mathbb{P}(\boldsymbol{x}(d)=1 \,|\, \boldsymbol{x}(d')=0)$. In all panels red/blue indicates large/small values. Green indicates a value close to 0.

bilities. One simple way to do this is to model the conditional distributions through logistic or linear regressions [39] which require only one parameter per parent.

Figure 5.2 illustrates the dependencies between variables for handwritten samples of the digit 7 by showing the logarithm of the likelihood ratios for $\boldsymbol{x}(d) = 1$ conditioned on $\boldsymbol{x}(d') = 0$ and $\boldsymbol{x}(d') = 1$, respectively. If the logarithm of the likelihood ratio is close to 0 then $\boldsymbol{x}(d')$ is not helpful for determining whether $\boldsymbol{x}(d) = 1$. As seen in the right panel of Figure 5.2 only a small number of the possible predictors have a likelihood ratio which is significantly different from 0. We hence use an L1-penalty for regularization since this induces sparsity. In Section 5.5 we experimentally confirm that an L1-penalty indeed yields much better results than an L2-penalty. We also tried elastic nets [112], which use an L1- as well as an L2-penalty. But the performance was typically worse than with just an L1-penalty. Since the effect of regularization depends on the scale of the variables, we encode binary data as ± 1 in order not to create a systematic bias. Similarly, for real-valued data we normalize the variables to have mean zero and variance one. The conditional distributions in our autoregressive network are then modeled through L1-penalized logistic or linear regressions which are fitted separately for each dimension via coordinate descent.

Concretely, for binary data $\boldsymbol{x} \in \{-1, 1\}^D$ we minimize

$$\sum_{n=1}^{N} \log(1 + \exp(-\boldsymbol{x_n}(d)[\boldsymbol{\alpha_d}(0) + \boldsymbol{\alpha_d}(1{:}d{-}1)^T \boldsymbol{x_n}(1{:}d{-}1)])) + \lambda_0 |\boldsymbol{\alpha_d}(0)| + \lambda ||\boldsymbol{\alpha_d}(1{:}d{-}1)||_1$$

where $\lambda_0, \lambda > 0$. The intercept $\boldsymbol{\alpha_d}(0)$ has a special meaning and is hence regularized differently than the dependency weights $\boldsymbol{\alpha_d}(i)$, $i{=}1, \ldots, d{-}1$. Shrinkage for the intercept is needed to avoid potentially degenerated probabilities (in case the empirical variance of $\boldsymbol{x}(d)$ is zero) but also improves the generalization performance. The penalty strength λ_0 for the intercept can be much smaller than λ though. The ratio λ/λ_0 is known as the intercept scaling factor. We do not expect the intercepts to be mostly zero, so a different type of penalty could be used in principle. But for uniformity we penalize the absolute value.

For continuous data $\boldsymbol{x} \in \mathbb{R}^D$ we minimize

$$\frac{1}{2} \sum_{n=1}^{N} (\boldsymbol{x_n}(d) - \boldsymbol{\alpha_d}(1{:}d{-}1)^T \boldsymbol{x_n}(1{:}d{-}1))^2 + \lambda ||\boldsymbol{\alpha_d}(1{:}d{-}1)||_1$$

where $\lambda > 0$. The intercept is not needed here since we center all variables in advance. The parameters $\boldsymbol{\alpha_d}(1{:}d{-}1)$ specify the conditional mean of $\boldsymbol{x}(d)$ given $\boldsymbol{x}(1{:}d{-}1)$, which we denote by $\boldsymbol{\mu}(d) \,|\, \boldsymbol{x}(1{:}d{-}1)$. The conditional standard deviation $\boldsymbol{\sigma}(d)$ is assumed to be fixed (i.e., it does not depend on $\boldsymbol{x}(1{:}d{-}1)$) and is estimated from the residuals $\boldsymbol{x_n}(d) - \boldsymbol{\mu}(d) \,|\, \boldsymbol{x}(1{:}d{-}1)$. If each conditional distribution $\boldsymbol{x}(d) \,|\, \boldsymbol{x}(1{:}d{-}1)$ is Gaussian with a constant variance then $\boldsymbol{x}$ has a multivariate normal distribution. As an alternative, we could model the variance as a, say, quadratic function of the predicted value $\boldsymbol{\mu}(d) \,|\, \boldsymbol{x}(1{:}d{-}1)$ in which case $\boldsymbol{x}$ would no longer be multivariate Gaussian. In any case, the conditional distributions in our model are unimodal. We deal with multimodality by learning mixtures of autoregressive networks (Section 5.3). In contrast to that, Bishop [12], Davies and Moore [26] model the conditional distributions through mixtures but only learn a single network.

L1-penalized logistic and linear regressions for neighborhood selection have been used before in undirected graphical models [81, 93]. A fully Bayesian approach for deep sigmoid

belief nets with a sparsity-inducing prior has been used by Gan et al. [44]. In their work the posterior distribution on model parameters is approximated through a variational Bayes procedure. The quantitative performance of that method however is significantly worse than our L1-penalized regression approach (see Section 5.5).

Since we do not share weight between different conditional distributions our learning procedure can be perfectly parallelized over the data dimensions. We are hence able to train models for extremely high-dimensional data.

5.2.4 Examples

We compare the mentioned sparse autoregressive models on two examples.

Binary Ellipses

As a first simple example we consider a synthetic dataset of (axis-aligned) binary ellipses with various shapes placed at different locations in images of size 50×50 pixels. The first row in Figure 5.3 shows examples from the ground truth model. Samples from a Bernoulli model trained on this dataset (2nd row) look very noisy and the probability for observing an actual ellipse under this model is extremely low. The samples from a trained Chow-Liu tree model (3rd row) are more continuous but due to the tree structure strong dependencies only exist along the typical object boundaries and not across. Consequently, many samples are disconnected. Samples from the models with grid dependence structure (4th-6th row) on the other hand are all connected. However, because only local dependencies exist most samples are somewhat globally distorted. The quality improves the more parents are used. The learned conditional distributions for the grid model with four parents are shown in Figure 5.4. Parent configurations which have never been observed lead to conditional probabilities of $1/2$ due to the symmetric prior we are using for estimation (5.1). With a sparse logistic autoregressive network the data distribution can be fitted very closely (last row in Figure 5.3).

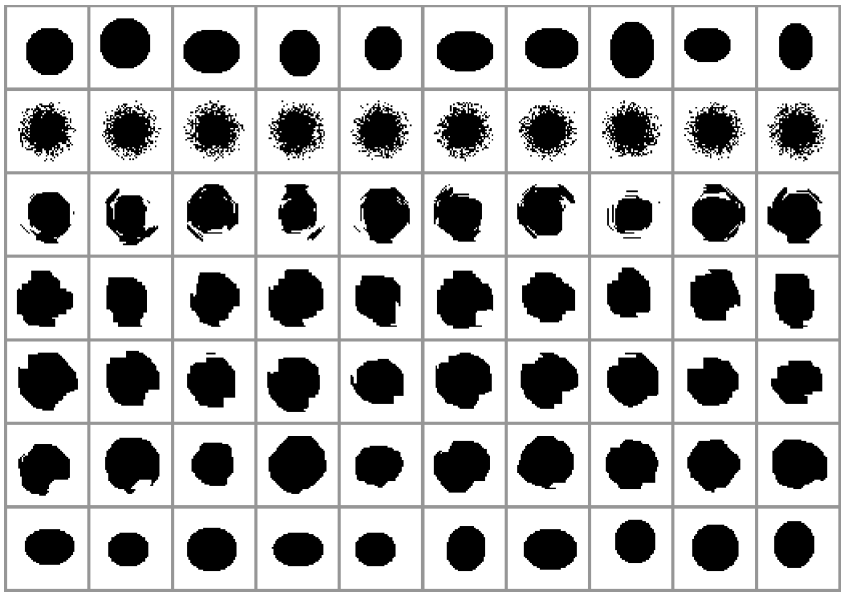

Figure 5.3: 1st row: Examples from the ellipse dataset. 2nd row: Samples from a trained Bernoulli model. 3rd row: Samples from a trained Chow-Liu tree model. 4th-6th row: Samples from trained grid models with two, three and four parents. 7th row: Samples from a trained sparse logistic autoregressive network. For all samples white/black indicates $\boldsymbol{x}(d) = 0/1$.

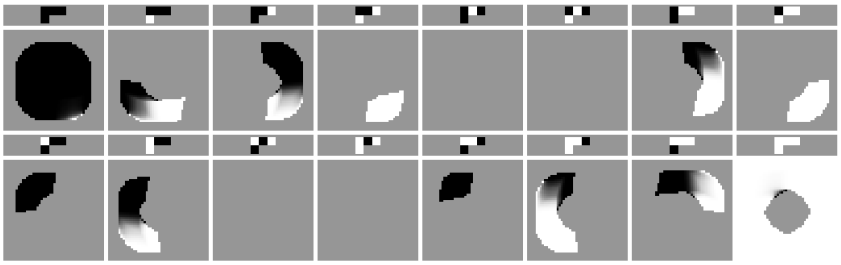

Figure 5.4: Conditional distributions in a grid model with four parents learned from the ellipse dataset. Shown are the parent configurations and the corresponding conditional probabilities at every pixel in the image grid. White/black indicates a probability of 0/1.

Caltech Faces

We now consider very high-dimensional real-valued data. The Caltech face dataset [36] consists of 450 high-resolution color images. We downsampled the images to $200 \times 150 \times 3$, which are 90,000 dimensions. Samples from a sparse linear autoregressive network with constant conditional variance are presented in the left panel of Figure 5.5. The samples are diverse and show substantial abstraction from the training examples. We also learned

Figure 5.5: Left: Samples from a sparse linear autoregressive network trained on Caltech faces (1st-5th column) and closest training examples for the adjacent sample (6th column). Right: Samples from a grid model with four parents.

a network in which the conditional variance was modeled as a quadratic function of the predicted value. This improved the likelihood only slightly and the corresponding samples were visually indistinguishable from the samples of the network with constant variance. For comparison, the quality of samples using a grid model with four parents is clearly inferior and artifacts due to the local dependence structure are visible.

5.3 Mixtures of Autoregressive Networks

For large datasets a single sparse autoregressive network is often not competitive with more sophisticated networks. Since learning requires only relatively few training examples it is natural to consider mixtures of the form

$$\mathbb{P}(\boldsymbol{x}) = \sum_{k=1}^{K} \mathbb{P}(h{=}k) \prod_{d=1}^{D} \mathbb{P}(\boldsymbol{x}(d) \,|\, \boldsymbol{x}(1{:}d{-}1), h)$$

where the conditional distributions are sparse linear/logistic regressions. The latent variable $h \in \{1, \ldots, K\}$ indicates the active mixture component. It is straightforward to learn such a mixture using the EM-algorithm [28]. A good initialization can be obtained by first learning a mixture of product (Bernoulli or Gaussian) distributions for the data. In previous work by

Meila and Jordan [80] mixtures of tree-structured networks were considered but not mixtures of general sparse autoregressive networks.

5.3.1 Without Parameter Sharing

In a classical mixture model no parameters are shared among the components. The form[1] of the component conditionals in this case is

$$\mathbb{P}(\boldsymbol{x}(d){=}1 \mid \boldsymbol{x}(1{:}d{-}1), h) = \sigma(\boldsymbol{\beta_{dh}}(0) + \boldsymbol{\beta_{dh}}(1{:}d{-}1)^T \boldsymbol{x}(1{:}d{-}1)).$$

Consequently, separate autoregressive networks are trained for different data clusters. Without parameter sharing a typical number of mixture components before overfitting occurs is around one per one thousand training examples.

5.3.2 With Parameter Sharing

A reasonable simplification is to assume that the dependence structure is universal, i.e., it does not change (much) for the different mixture components. With shared dependency weights a much larger mixture can be trained because only separate intercepts have to be learned. The form of the component conditionals with parameter sharing is

$$\mathbb{P}(\boldsymbol{x}(d){=}1 \mid \boldsymbol{x}(1{:}d{-}1), h) = \sigma(\boldsymbol{\beta_{dh}}(0) + \boldsymbol{\alpha_{d}}(1{:}d{-}1)^T \boldsymbol{x}(1{:}d{-}1)).$$

The same sharing of dependency weights is used in autoregressive sigmoid belief networks [52], which are large implicit mixtures of (dense) autoregressive networks. Note that this kind of parameter sharing across mixture components is very different from the mentioned weight sharing in neural autoregressive networks [e.g., 63] where weights are shared across conditionals for different dimensions. With shared dependency weights a typical number

1. We use here the notation for binary data, the corresponding expressions for continuous data are evident.

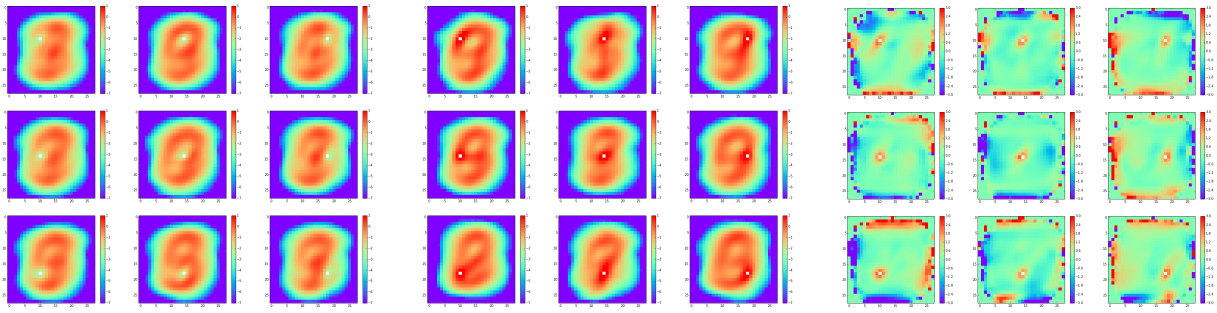

Figure 5.6: Dependencies between variables for handwritten samples of all digit classes. Left: Empirical log-odds $\log[\mathbb{P}(\boldsymbol{x}(d)=1 \,|\, \boldsymbol{x}(d')=0) \;/\; \mathbb{P}(\boldsymbol{x}(d)=0 \,|\, \boldsymbol{x}(d')=0)]$ of examples with $\boldsymbol{x}(d)'=0$ for $d' \in \{10, 14, 18\} \times \{10, 14, 18\}$. Center: The same for examples with $\boldsymbol{x}(d')=1$. Right: Logarithm of the likelihood ratios $\mathbb{P}(\boldsymbol{x}(d)=1 \,|\, \boldsymbol{x}(d')=1) \;/\; \mathbb{P}(\boldsymbol{x}(d)=1 \,|\, \boldsymbol{x}(d')=0)$. In all panels red/blue indicates large/small values. Green indicates a value close to 0. Compare with Figure 5.2 where samples from only one digit class are used.

of mixture components before overfitting occurs is around one per one hundred training examples. Note that if the dependence structure varies a lot among subsets of the data then shared dependency weights can be rather limited. For example, consider the likelihood ratios for handwritten samples from all digit classes in Figure 5.6. The number of nonzero dependencies among variables is much smaller than for a single digit class (cf. Figure 5.2). As we will see in Section 5.5 depending on the dataset the quantitative performance with parameter sharing can be better or worse than without parameter sharing.

5.3.3 Automatic Parameter Sharing

Rather than manually deciding which parameters to share we can let the amount of sharing be part of the learning process. Specifically, we can use a global bias term and global dependency weights, and let the component parameters describe deviations from these global parameters. The form of the component conditionals in this case is

$$\mathbb{P}(\boldsymbol{x}(d)=1 \,|\, \boldsymbol{x}(1{:}d{-}1), h) = \sigma\left(\boldsymbol{\alpha_d}(0) + \boldsymbol{\beta_{dh}}(0) + [\boldsymbol{\alpha_d}(1{:}d{-}1) + \boldsymbol{\beta_{dh}}(1{:}d{-}1)]^T \boldsymbol{x}(1{:}d{-}1)\right).$$

We want the component-specific parameters to be nonzero only if there is substantial gain from untying the weights from the global parameter. To achieve this, we use an L1-penalty

$$\lambda_0|\boldsymbol{\alpha_d}(0)| + \lambda_0 \sum_{h=1}^{K} |\boldsymbol{\beta_{dh}}(0)| + \lambda||\boldsymbol{\alpha_d}(1{:}d{-}1)||_1 + \lambda \sum_{h=1}^{K} ||\boldsymbol{\beta_{dh}}(1{:}d{-}1)||_1.$$

A similar penalty for deviations from a shared parameter has been used for multi-task learning with SVMs [33]. The optimal number of mixture components with automatic sharing is between the optimal numbers of components for mixtures with and without parameter sharing, respectively. In our experiments, mixtures with automatic parameter sharing always performed better than the mixtures with or without parameter sharing.

5.4 Sequence of Mixtures

In distributed representations [11] the traditional approach is to use a top-down architecture in which the latent variables are generated first and the distribution of the observable variables is modeled conditioned on the latent state. Such an organization of the variables makes inference and learning difficult. Moreover, exact likelihood evaluations are then usually intractable because the required computations involve a sum over all configurations of the latent variables. We propose a simple partition-based representation for which inference is trivial and exact likelihood evaluations are tractable. The basic idea is to divide the data dimensions into chunks and to learn local models for each of these subsets of variables. Specifically, we partition the D dimensions into L disjoint intervals $S_\ell = [d_{\ell-1}{+}1 : d_\ell]$ where

$$0 = d_0 < d_1 < \ldots < d_L = D.$$

We then use mixtures of sparse autoregressive networks for each of the data vectors $\boldsymbol{x}(S_\ell)$, $\ell = 1, \ldots, L$, utilizing the data $\boldsymbol{x}(1{:}d_{\ell-1})$ from previous intervals as additional predictors,

i.e.,

$$\mathbb{P}(\boldsymbol{x}(S_\ell) \,|\, \boldsymbol{x}(1{:}d_{\ell-1}), \boldsymbol{h}(\ell)) = \prod_{d=d_{\ell-1}+1}^{d_\ell} \mathbb{P}(\boldsymbol{x}(d) \,|\, \boldsymbol{x}(1{:}d{-}1), \boldsymbol{h}(\ell)).$$

Associated with each mixture is a latent variable $\boldsymbol{h}(\ell) \in \{1, \ldots, K_\ell\}$ which specifies the component that is active for interval S_ℓ. By choosing different configurations of mixture components global variation is achieved while the autoregressive networks themselves capture local variation of the data. For each choice of latent variables, the combined model is an autoregressive network for $\boldsymbol{x}$.

Partition-based representations are simple and interpretable distributed representations in which the area of responsibility is explicitly defined for the local models. The main drawbacks of existing partition-based approaches for image modeling [85, 3] are discontinuities and misalignments along the partition boundaries. Since in our approach each local model is an autoregressive network that uses the data from all previous dimensions as predictors (not just the data within its own interval S_ℓ) these problems are largely eliminated, see the experimental results in Section 5.5.

What remains to be specified is the distribution on the latent variables. In top-down approaches, for example deep autoregressive networks [52], the joint distribution on latent and observable variables is factored as $\mathbb{P}(\boldsymbol{h}, \boldsymbol{x}) = \mathbb{P}(\boldsymbol{h}) \cdot \mathbb{P}(\boldsymbol{x} \,|\, \boldsymbol{h})$. This makes it easy to evaluate $\mathbb{P}(\boldsymbol{h})$ but hard to evaluate $\mathbb{P}(\boldsymbol{x})$. Rather than first generating all latent variables we propose to interleave them with the observable variables. Formally, the joint distribution is factored as

$$\mathbb{P}(\boldsymbol{h}, \boldsymbol{x}) = \prod_{\ell=1}^{L} \mathbb{P}(\boldsymbol{h}(\ell) \,|\, \boldsymbol{x}(1{:}d_{\ell-1})) \cdot \mathbb{P}(\boldsymbol{x}(S_\ell) \,|\, \boldsymbol{x}(1{:}d_{\ell-1}), \boldsymbol{h}(\ell)).$$

where $\mathbb{P}(\boldsymbol{h}(\ell) \,|\, \boldsymbol{x}(1{:}d_{\ell-1}))$ are multiclass logistic regressions. In particular, this means that given $\boldsymbol{x}(1{:}d_{\ell-1})$ the distribution for $\boldsymbol{h}(\ell)$ does not depend on $\boldsymbol{h}(1{:}\ell{-}1)$. Since the posterior distribution $\mathbb{P}(\boldsymbol{h} \,|\, \boldsymbol{x})$ factorizes over the latent variables, inference is trivial with our decomposition. In contrast to that, hidden Markov models [88] define the distribution of $\boldsymbol{h}(\ell)$ in terms of $\boldsymbol{h}(\ell{-}1)$. Our architecture also makes it easy to integrate out the latent variables.

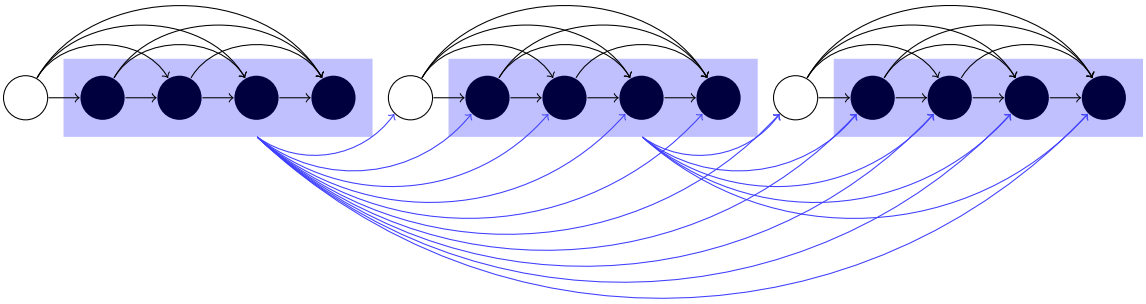

Figure 5.7: Graphical representation of the sequence of mixtures. Solid nodes represent observable variables and empty nodes represent latent variables. Blue edges denote dependence on a block of variables.

Indeed, it follows that

$$\mathbb{P}(\boldsymbol{x}) = \prod_{\ell=1}^{L} \sum_{\boldsymbol{h}(\ell)=1}^{K_\ell} \mathbb{P}(\boldsymbol{h}(\ell) \,|\, \boldsymbol{x}(1{:}d_{\ell-1})) \cdot \mathbb{P}(\boldsymbol{x}(S_\ell) \,|\, \boldsymbol{x}(1{:}d_{\ell-1}), \boldsymbol{h}(\ell)).$$

A graphical representation of this sequence of mixtures is shown in Figure 5.7. The proposed model family is a special subfamily of sum-product networks [86] which is conceptually simple and particularly easy to train. In fact, training our model reduces to learning L separate mixtures of sparse autoregressive networks because the full data likelihood $P(\boldsymbol{x})$ factorizes over ℓ. The following pseudocode can be used:

FOR $\ell = 1, \ldots, L$:

1) Learn a Bernoulli mixture model with K_ℓ components for $\boldsymbol{x}(S_\ell)$.
2) Transform the Bernoulli mixture into a mixture of sparse autoregressive networks for $\boldsymbol{x}(S_\ell)$ given $\boldsymbol{x}(1{:}d_{\ell-1})$ using the responsibilities $\mathbb{P}_{\text{Ber}}(\boldsymbol{h}(\ell) \,|\, \boldsymbol{x}(S_\ell))$.
3) Optionally update the model by running additional EM iterations using the responsibilities $\mathbb{P}(\boldsymbol{h}(\ell) \,|\, \boldsymbol{x}) \propto \mathbb{P}(\boldsymbol{h}(\ell) \,|\, \boldsymbol{x}(1{:}d_{\ell-1})) \cdot \mathbb{P}(\boldsymbol{x}(S_\ell) \,|\, \boldsymbol{x}(1{:}d_{\ell-1}), \boldsymbol{h}(\ell))$.
4) Fit a logistic regression model for $\boldsymbol{h}(\ell)$ given $\boldsymbol{x}(1{:}d_{\ell-1})$.

5.5 Experiments

We quantitatively evaluated our sparse autoregressive networks (SpARN) on various high-dimensional datasets. For all experiments we used an intercept scaling factor (see Section 5.2.3) of 10, meaning the penalty on the intercept is a factor 10 smaller than for the dependency weights. The exact choice of the factor is not crucial, other values between 10 and 100 lead to very similar results. We used likelihood evaluations on the validation set to choose the number of mixture components and to select the penalty strength λ. The same penalty strength was used for every dimension. We also performed a qualitative evaluation of the sequence of mixtures on a very high-dimensional dataset.

5.5.1 Caltech-101 Silhouettes

The first dataset we consider are the Caltech-101 silhouettes [76]. There are 4,100 training samples, 2,264 validation samples and 2,307 test samples of dimension $28 \times 28 = 784$. Each data point is a binary mask for an object from one of 101 categories. Since the similarity between samples is of semantic nature a large degree of abstraction is required to generalize well to unseen examples. Previous results for this dataset are shown in Table 5.1 (left). The current state of the art is a logistic autoregressive network, also called a fully visible sigmoid belief net (FVSBN), with sparse parameters [44]. The difference to our approach is that Gan et al. [44] learn their model through a variational Bayes procedure. Our network on the other hand is based on L1-penalized logistic regressions and trained through coordinate descent. The average test log-likelihood we achieve is 4.5 nats higher. An autoregressive network with L2-penalized logistic regressions performs significantly worse. This means that the sparsity which is induced by the L1-penalty is indeed crucial. We also trained mixtures of sparse autoregressive networks in an unsupervised manner (i.e., without using any category labels). The mixture with shared dependency weights (tied) improves over the single network by more than 2 nats. The optimal number of mixture components is 100. This makes sense

Caltech-101 silhouettes	
RBM [89]	≈-107.78
NADE-5 [89]	-107.28
RWS-NADE [16]	≈-104.30
FVSBN (VB) [44]	≈-96.40
ARN (L2-penalty)	-95.95
SpARN	-91.80
MoSpARN (100 comp, auto)	**-88.48**
MoSpARN (100 comp, tied)	-89.59

Binarized MNIST digits			
FVSBN (VB) [44]	≈-100.76	SpARN	-97.34
FVSBN (SGD) [63]	-97.45	MoSpARN (20 comp, untied)	-89.43
NADE [63]	-88.86	MoSpARN (50 comp, auto)	-87.95
MADE [51]	-86.64	MoSpARN (500 comp, tied)	-91.32
RBM (CD-25) [63]	≈-86.34	MoSpARN (4×50 comp, auto)	-87.40
DARN [52]	≈ **-84.13**	MoSpARN (4×500 comp, tied)	-88.63

Table 5.1: Average log-likelihoods (in nats) per test example using different models. The best result for each of the two datasets is shown in bold.

because that is about the number of different object categories in the dataset. When trained without intercept scaling the mixture essentially reduced to a single sparse network. Hence, it was important to have a weaker penalty on the intercepts. The mixture with automatic parameter sharing (auto) further improves the test performance by about 1 nat. The mixture without parameter sharing (untied) was not better than the single network.

5.5.2 Binarized MNIST Digits

The next dataset we consider are the MNIST digits [94] which were binarized through random sampling based on the original intensity value. This is a relatively large dataset consisting of 50,000 training samples, 10,000 validation samples and 10,000 test samples again of dimension $28 \times 28 = 784$. Previous results for this dataset are shown in Table 5.1 (right). A fully visible sigmoid belief net trained by stochastic gradient descent and regularized through early stopping achieves about the same performance as a sparse logistic autoregressive network. This is because the dataset is rather large and hence the need for regularization is small. For mixtures without parameter sharing the optimal number of components is 20. When automatic parameter sharing is used the optimal number of components is 50 and with shared dependency weights it is 500. All mixtures substantially improve over the single network. For this dataset, the mixture without sharing performs better than the mixture with shared parameters. We believe that this is the case because compared to the Caltech-101 silhouettes the distribution of MNIST digits has fewer modes and more training samples

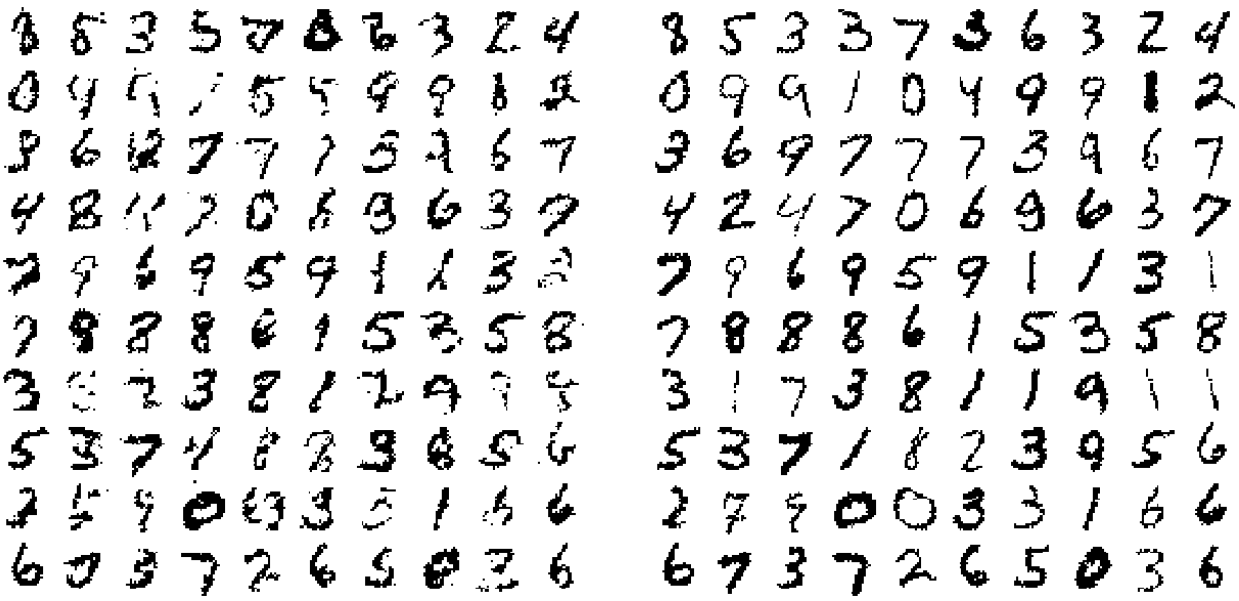

Figure 5.8: Left: Samples from a sequence of mixtures trained on MNIST digits. Right: Closest training examples.

are available to learn from. We also trained a sequence of mixtures. For that the image grid was divided into four quadrants of size 14×14 pixels and a mixture model was learned for each of the quadrants using the same number of components as before. With parameter sharing the sequence of mixtures improves the performance of the corresponding mixture by 2.5 nats. Note that the number of parameters for the two models is almost the same. The only additional parameters come from the multiclass logistic regressions for the hidden variables. When automatic parameter sharing is used the gain through a sequence of mixtures is smaller. Without parameter sharing there is no gain from the sequence of mixtures. Samples from the sequence of mixtures with automatic parameter sharing as well as nearest training examples are shown in Figure 5.8. As we see the model puts the probability mass in the right region of the data space and it does not simply memorize the training examples.

	Adult	**Connect4**	**DNA**	**Mushrooms**	**NIPS-0-12**	**OCR-letters**	**RCV1**	**Web**
train	*5,000*	*16,000*	*1,400*	*2,000*	*400*	*32,152*	*40,000*	*14,000*
valid	*1,414*	*4,000*	*600*	*500*	*100*	*10,000*	*10,000*	*3,188*
test	*26,147*	*47,557*	*1,186*	*5,624*	*1,240*	*10,000*	*150,000*	*32,561*
dim	*123*	*126*	*180*	*112*	*500*	*128*	*150*	*300*
Ber. Mix.	-20.44	-23.41	-98.19	-14.46	-290.02	-40.56	-47.59	30.16
FVSBN (SGD)	-13.17	-12.39	-83.64	-10.27	-276.88	-39.30	-49.84	-29.35
NADE	-13.19	-11.99	-84.81	-9.81	-273.08	-27.22	-46.66	-28.39
DARN	-13.19	-11.91	-81.04	-9.55	-274.68	≈-28.17	≈-46.10	≈-28.83
MADE	-13.12	-11.90	-79.66	-9.68	-277.28	-28.34	-46.74	-28.25
RWS-NADE	≈-13.16	**≈-11.68**	≈-84.26	≈-9.71	≈-271.11	**≈-26.43**	≈-46.09	≈-27.92
MoSpARN (untied)	**-13.04**	-12.04	-79.32	-9.58	-271.13	-28.48	-45.55	-27.98
MoSpARN (auto)	**-13.04**	-11.98	**-79.05**	**-9.38**	**-270.29**	-27.96	**-45.11**	**-27.50**
MoSpARN (tied)	**-13.04**	-12.14	-79.26	-9.39	-270.53	-31.53	-45.50	-28.12
std. error	*0.02*	*0.01*	*0.21*	*0.01*	*0.52*	*0.11*	*0.06*	*0.10*
comp (untied)	*1*	*10*	*1*	*2*	*1*	*100*	*200*	*100*
comp (auto)	*1*	*10*	*3*	*10*	*20*	*200*	*1,000*	*200*
comp (tied)	*1*	*10*	*3*	*20*	*20*	*5,000*	*5,000*	*200*

Table 5.2: Average log-likelihoods (in nats) per test example for various models and datasets. The best result for each dataset is shown in bold. Baseline results are taken from Germain et al. [51], Bornschein and Bengio [16]. Dataset sizes, standard errors and used numbers of mixture components are shown in italic.

5.5.3 Binary UCI Datasets

We performed additional quantitative evaluations on eight standard benchmark datasets from the UCI Machine Learning Repository [70] using the same splits into training, validation and test sets as in previous work. The datasets have various sample sizes and dimensions, and come from many different domains. The ratio of training samples to dimensionality varies between 1 and 300. Table 5.2 reports the obtained test log-likelihoods for mixtures of sparse autoregressive networks together with previous results. The table also shows the optimal numbers of mixture components for the different kinds of parameter sharing and the standard error of the test log-likelihood (when automatic parameter sharing was used). On smaller datasets mixtures with parameter sharing tend to perform a bit better than mixtures without parameter sharing, and vice versa on larger datasets. The performance of our mixtures of sparse autoregressive networks with automatic parameter sharing is better than the best previously reported result on six of the eight datasets (all improvements except for the NIPS-0-12 dataset are statistically significant). For the other two datasets the performance is close to the state of the art.

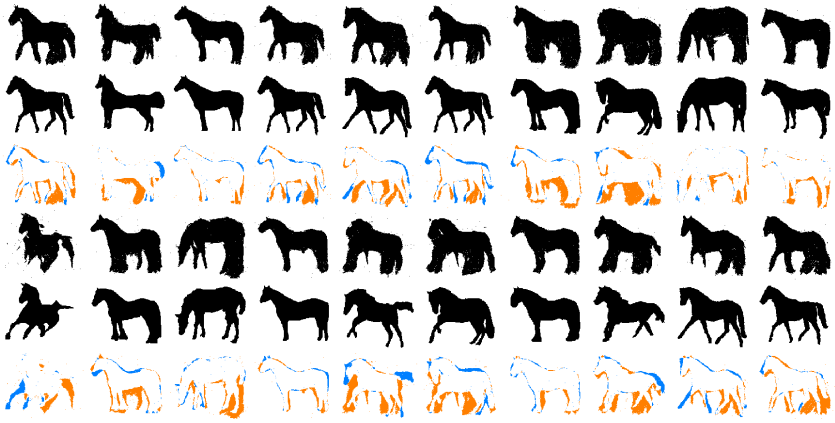

Figure 5.9: 1st&4th row: Samples from the sequence of mixtures trained on Weizmann horses. 2nd&5th row: Closest training examples. 3rd&6th row: Symmetric differences between the synthetic sample and the closest training example (pixels in blue are only "on" for the training example and pixels in orange are only "on" for the synthetic sample).

5.5.4 Weizmann Horses

The following experiment shows the ability of our model to deal with very high-dimensional binary data. The Weizmann horse dataset [15] consists of 328 binary images of size 200×240. We divided the image grid into four quadrants of size 100×120 and learned a mixture of sparse autoregressive networks for each quadrant. We decided to use five components each for the top two quadrants (i.e., horse head and back) and ten components each for the bottom two quadrants (i.e., horse legs) because there is more variability in the lower half of the image. Samples from the sequence of mixtures are presented in Figure 5.9 together with the closest training examples. Our model creates overall realistic samples which differ from the training examples. In particular, no discontinuities along the borders of the quadrants are visible. We emphasize that our model was trained using the full resolution of the images, each data point is a 48,000 dimensional binary vector. Eslami et al. [32] used the same dataset to learn a Boltzmann machine. In their experiments the images were downsampled to 32×32 pixels, which is a factor 50 smaller than in our case. This was probably necessary in order to speed up computations and to avoid overfitting. Refer to Figure 5(d) in their paper to compare the quality of the model samples.

CHAPTER 6

LOGITBOOST AUTOREGRESSIVE NETWORKS

In the previous chapter we estimated distributions through mixtures of autoregressive networks with simple parametric conditionals. In this chapter we use a nonparametric approach and learn a single network in which each conditional distribution is modeled through a nonlinear function of the predictors. Specifically, we explore the use of LogitBoost [40] for learning distributions of high-dimensional binary data. LogitBoost is a state-of-the-art probability estimator based on boosted trees [42]. Boosted trees are collections of relatively shallow decision trees that are trained in a way as to complement each other. The bias-variance tradeoff for an ensemble of boosted trees is often much better than for a single decision tree. We call the proposed model a *LogitBoost autoregressive network* (LBARN). Our contributions are as follows:

1. We propose LogitBoost as a learning procedure for the conditionals in an autoregressive network. In existing work [97] boosting was applied to continuous, stationary time series which means that only a single conditional distribution is learned. Our network on the other hand trains one conditional distributions for each data dimension.

2. Neural autoregressive networks are currently the state-of-the-art for nonparametric distribution estimation in high-dimensions (say $D > 50$). We show that there is a simpler alternative which works equally well. The similarities and differences between our approach and neural methods are discussed in detail.

3. In contrast to neural autoregressive networks our model does not use any weight sharing among conditionals for different dimensions. Consequently, our training procedure can be perfectly parallelized over the D dimensions, i.e., each conditional distribution can be learned on a separate processor without requiring any communication. On the other hand, our approach requires a method to combine all the conditionals into a co-

herent joint model. We propose three different combination procedures and empirically evaluate them on several diverse datasets.

4. Finally, we study how the ordering of the variables effects the performance of the trained model. In particular, we introduce a sorting procedure which allows us to build simple compression mechanisms.

In Section 6.1 we provide a brief overview of LogitBoost. In Section 6.2 we describe the details of the proposed autoregressive network. In particular, we discuss model selection, model refitting and contrast our model with neural autoregressive networks. In Section 6.3 we present results for several common datasets and consider different orderings of the variables.

6.1 LogitBoost

LogitBoost [40] is a forward stagewise procedure that fits additive logistic regression models

$$\mathbb{P}_t(y{=}1 \,|\, \boldsymbol{x}) = \sigma(f_1(\boldsymbol{x}) + \ldots + f_t(\boldsymbol{x}))$$

for $t = 1, \ldots, T$ to training data $(\boldsymbol{x_n}, y_n)$, $n = 1, \ldots, N$, by maximum likelihood. The optimization is based on a second-order Taylor expansion of the Bernoulli log-likelihood

$$L_t = \sum_{n=1}^{N} \log \mathbb{P}_t(y_n \,|\, \boldsymbol{x_n}).$$

Friedman et al. [40] derived that

$$\frac{\partial L_t}{\partial f_t} = \sum_{n=1}^{N} (y_n - p_{n,t-1}), \quad \frac{\partial^2 L_t}{\partial f_t^2} = -\sum_{n=1}^{N} p_{n,t-1}(1 - p_{n,t-1})$$

where

$$p_{n,t-1} = \mathbb{P}_{t-1}(y = 1 \,|\, \boldsymbol{x_n})$$

is the conditional success probability given $\boldsymbol{x_n}$ as predicted by the model after $t-1$ rounds of boosting. The Newton step results in a weighted least-squares regression of the current pseudoresiduals $y_n - p_{n,t-1}$ onto the predictors $\boldsymbol{x_n}$. The base learners are typically J-terminal regression trees

$$f_t(\boldsymbol{x}) = \sum_{j=1}^{J} \alpha_j \mathbb{1}\{\boldsymbol{x} \in R_j\}$$

where $\alpha_j \in \mathbb{R}$. It can be shown [68] that the weighted regression problem is then equivalent to fitting a decision tree with a simple splitting criterion. When learning f_t the gain of splitting region R_j using variable $\boldsymbol{x}(d)$ is

$$\begin{aligned}\mathrm{Gain}(j,d) = &\frac{\left[\sum_{n=1}^{N}(y_n - p_{n,t-1})\mathbb{1}(\boldsymbol{x_n} \in R_j,\, \boldsymbol{x_n}(d) = 0)\right]^2}{\sum_{n=1}^{N} p_{n,t-1}(1-p_{n,t-1})\mathbb{1}(\boldsymbol{x_n} \in R_j,\, \boldsymbol{x_n}(d) = 0)} \\ &+ \frac{\left[\sum_{n=1}^{N}(y_n - p_{n,t-1})\mathbb{1}(\boldsymbol{x_n} \in R_j,\, \boldsymbol{x_n}(d) = 1)\right]^2}{\sum_{n=1}^{N} p_{n,t-1}(1-p_{n,t-1})\mathbb{1}(\boldsymbol{x_n} \in R_j,\, \boldsymbol{x_n}(d) = 1)} - \frac{\left[\sum_{n=1}^{N}(y_n - p_{n,t-1})\mathbb{1}(\boldsymbol{x_n} \in R_j)\right]^2}{\sum_{n=1}^{N} p_{n,t-1}(1-p_{n,t-1})\mathbb{1}(\boldsymbol{x_n} \in R_j)}.\end{aligned}$$

Once a J-terminal tree is learned the value of the leaves is set to

$$\alpha_j = \frac{\sum_{n=1}^{N}(y_n - p_{n,t-1})\mathbb{1}(\boldsymbol{x_n} \in R_j)}{\sum_{n=1}^{N} p_{n,t-1}(1-p_{n,t-1})\mathbb{1}(\boldsymbol{x_n} \in R_j)}. \tag{6.1}$$

The number J of leaves impacts the depth of the trees and hence determines the order of interactions which can be modeled. While LogitBoost is rather robust to overfitting for classification tasks, overfitting does occur for probability estimates if too many rounds of boosting are performed [78]. It is therefore customary to choose the number of trees based on the performance on holdout data. Better results can often be achieved by introducing shrinkage since this facilitates the use of somewhat deeper trees which otherwise would not

generalize well. The log-odds of $y = 1$ in this case is modeled as $\nu \sum_{t=1}^{T} f_t$ where $\nu \in (0, 1)$ is a shrinkage parameter. Smaller values of ν almost always lead to better generalization performance. However, the smaller ν is the more trees are needed which leads to more expensive computations.

6.2 Model Details

For each data dimension $d = 1, \ldots, D$ we use LogitBoost to learn the conditional distribution $\mathbb{P}(\boldsymbol{x}(d) \,|\, \boldsymbol{x}(1{:}d{-}1))$ using up to T boosted trees. This yields D sequences of models $\mathbb{P}_t(\boldsymbol{x}(d) \,|\, \boldsymbol{x}(1{:}d{-}1))$, $t = 1, \ldots, T$, with increasing complexity. In the following we discuss different ways to combine these sequences into a single joint model. We also describe how the model parameters can be updated when additional data becomes available, without changing the learned functional structure.

6.2.1 Model Selection

For each dimension we have to choose one of the T models. The easiest way to do this is to perform model validation separately for each dimension d. This means we compute the likelihood of holdout data $\boldsymbol{x}(d)$ given $\boldsymbol{x}(1{:}d{-}1)$ under each of the T models. Let $\mathbb{P}_{t^\star(d)}(\boldsymbol{x}(d) \,|\, \boldsymbol{x}(1{:}d{-}1))$ be the model with the highest validation likelihood for the d-th dimension. The joint distribution is then estimated through

$$\mathbb{P}_{\text{sep}}(\boldsymbol{x}) = \prod_{d=1}^{D} \mathbb{P}_{t^\star(d)}(\boldsymbol{x}(d) \,|\, \boldsymbol{x}(1{:}d{-}1)).$$

If the validation set is rather small then this approach can actually overfit the validation set since we are making D model choices. We hence considered two alternative approaches which only perform a single model selection.

Conservative model selection is possible by using the entire data vector $\boldsymbol{x}$ for validation and selecting a common number of trees for all dimensions. We can simply compute the

validation likelihood for the models $\prod_{d=1}^{D} P_t(\boldsymbol{x}(d) \,|\, \boldsymbol{x}(1{:}d{-}1))$, $t = 1, \ldots, T$, and find the maximizer $t^\star$. The joint distribution is then estimated through

$$\mathbb{P}_{\text{com}}(\boldsymbol{x}) = \prod_{d=1}^{D} \mathbb{P}_{t^\star}(\boldsymbol{x}(d) \,|\, \boldsymbol{x}(1{:}d{-}1)).$$

The problem with this approach is that the same model complexity for each conditional distribution has to be used. This can lead to poor results if the data dimensions are heterogeneous.

A third approach is to construct a linear ordering of all learned trees and to perform validation on the corresponding sequence of global models. This can be achieved by starting with "empty" models for each dimension (i.e., using zero trees) and greedily adding a tree to the dimension which gives rise to the largest improvement in terms of training likelihood. Alternatively, the sorting procedure can be performed in a backward manner by starting with "full" models for each dimension (i.e., using all T trees) and greedily removing the tree which gives rise to the smallest decrease of training likelihood. These procedures create a sequence of models $\mathbb{P}_s(\boldsymbol{x})$, $s = 1, \ldots, D \cdot T$. We find the maximizer $s^\star$ of the validation likelihood and estimate the joint distribution through

$$\mathbb{P}_{\text{lin}}(\boldsymbol{x}) = \mathbb{P}_{s^\star}(\boldsymbol{x}).$$

This method makes only a single model selection while allowing for different model complexities in each dimension.

6.2.2 Model Refitting

A usual step after model selection is refitting of the parameters using the pooled training and validation data. In our model this is easy to perform. For each dimension we keep the structures of all selected trees fixed, i.e., we use the same partitionings into regions.

We then simply rerun the updates of the leaf values as in equation (6.1). This refitting is computationally efficient because the most expensive part of training, namely deciding which split to perform, can be skipped.

6.2.3 Comparison With Neural Networks

We now discuss similarities and differences between our model and autoregressive models based on neural networks. In both approaches the log-odds of $\boldsymbol{x}(d) = 1$ are modeled through a learned nonlinear function of $\boldsymbol{x}(1{:}d{-}1)$. The main difference is that we learn separate models for each dimension while weight sharing among conditionals is an important regularizer for neural networks. Indeed, the particular choice of parameter sharing introduced by Larochelle and Murray [63] greatly improves the generalization performance compared to earlier neural networks which were susceptible to overfitting [10, 104].

Another difference lies in the optimization. Neural networks use stochastic gradient descent starting from a random initialization of all parameters. Our model starts with log-odds of zero for each conditional and uses a second-order Taylor approximation of the log-likelihood for the parameter updates. In both cases the model complexity increases with additional iterations of gradient descent or additional rounds of boosting. The learning rate for neural networks has a very similar role as the shrinkage factor ν for boosting. The gradient descent learning algorithms typically also require one to choose a momentum and a mini-batch size. Model selection for neural networks is performed by keeping track of the validation likelihood over iterations of gradient descent and returning the model with the best performance on the validation set. This is similar to stopping after a certain number of trees have been learned. Since with neural networks all conditionals are learned at once only a single model selection is performed. The most analogous selection procedure for our model would be the one where we create a linear ordering of all trees based on training performance, see Section 6.2.1.

A further difference is that neural networks have a fixed architecture which is chosen by

trial and error. This includes the number of hidden layers, the number of units per layer and the connections between them. Only the weights for the connections are learned. In contrast to that, our model also learns the structure of the nonlinear functions. Indeed, each regression tree produces a function which depends only on a small subset of the variables. As described in the previous subsection this facilitates a simple refitting procedures which is not available for neural networks. In principle, neural networks could be relearned on the combined data by running gradient descent for the selected number of iterations. However, this does not really specify a particular model complexity and it is hence questionable if this would lead to good results. In our model such a retraining would correspond to learning new tree structures which is much slower than simply updating the parameters.

Apart from the learning rate the only hyperparameter that has to be specified in order to train our model is the number J of leaves for each tree. This number determines what degree of interactions can be incorporated in our model. Such a hyperparameter does not exist in neural networks. Instead the number of hidden units and the architecture of the network determine how complex the conditional distributions can become.

The computational complexity to evaluate the exact log-likelihood of a test sample under our model is $\mathcal{O}(DTL)$ where L is the average tree depth. If the trees are balanced then $L = \log J$. The simplest autoregressive model based on neural networks is NADE [63] which consists of a single layer of hidden units. Likelihood evaluations for NADE can be performed with $\mathcal{O}(DH)$ operations where H is the number of hidden variables. Deeper versions of NADE [105] with multiple layers of hidden units require $\mathcal{O}(DH^2)$ operations. Autoencoder networks [51] with more than one layer of hidden units require $\mathcal{O}(DH+H^2+D^2)$ operations. For ensemble networks [105] the complexities scale multiplicatively with the number of different orderings of the data dimensions. For neural networks that use stochastic hidden units [52, 16] the computations are exponential in H because a sum over all possible configurations of the hidden variables has to be taken. Importance sampling can then be used as an approximation. A comprehensive list of the computational complexities was provided

by Germain et al. [51]. In summary, likelihood evaluation in our model scales more gracefully to high dimensions compared to the sophisticated variants of neural autoregressive networks.

6.3 Experiments

We evaluated LogitBoost autoregressive networks (LBARN) on nine standard benchmark datasets (which we already used in Chapter 5) using the same splits into training, validation and test sets as in previous work. The ratio of sample size and dimensionality in these datasets varies between 1 and 300, see Table 6.1 for details. The MNIST dataset [94] was binarized through random sampling based on the original intensity values. The OCR-letters dataset comes from the Stanford AI Lab[1] and the remaining seven datasets come from the UCI Machine Learning Repository [70]. Our primary interest is a quantitative comparison with neural autoregressive models. Previous results for the used datasets were reported by Germain et al. [51] and Bornschein and Bengio [16].

LBARN was trained using up to $T = 1{,}000$ rounds of boosting per dimension. The number of trees to be used was then selected individually for each dimension using the validation set. For the number of leaves per tree we considered $J \in \{2, 4, 8, 16, 32, 64, 128\}$ (this corresponds to balanced trees of depth 1-7) and selected the best value based on validation performance. The shrinkage factor was set to $\nu = 0.02$. This value turned out to be small enough to allow us to train trees with a large number of leaves. Even smaller values of ν typically did not lead to better results and only slowed down the learning process. In cases where convergence had not occurred after 1,000 iterations we increased ν accordingly. This was only necessary for two datasets.

Table 6.1 reports our obtained test log-likelihoods together with previous results. The table also shows the selected number J of leaves for each dataset and the used shrinkage factor ν. For datasets with somewhat larger sample size (relative to the data dimension) deeper trees turned out to be beneficial. Overall LBARN is among the best performing

1. `http://ai.stanford.edu/~btaskar/ocr/`

	Adult	Connect4	DNA	MNIST	Mushrooms	NIPS-0-12	OCR-letters	RCV1	Web
train	*5,000*	*16,000*	*1,400*	*50,000*	*2,000*	*400*	*32,152*	*40,000*	*14,000*
valid	*1,414*	*4,000*	*600*	*10,000*	*500*	*100*	*10,000*	*10,000*	*3,188*
test	*26,147*	*47,557*	*1,186*	*10,000*	*5,624*	*1,240*	*10,000*	*150,000*	*32,561*
dim	*123*	*126*	*180*	*784*	*112*	*500*	*128*	*150*	*300*
Ber. Mix.	-20.44	-23.41	-98.19	-137.64	-14.46	-290.02	-40.56	-47.59	30.16
FVSBN	-13.17	-12.39	-83.64	-97.45	-10.27	-276.88	-39.30	-49.84	-29.35
NADE	-13.19	-11.99	-84.81	-88.33	-9.81	-273.08	-27.22	-46.66	-28.39
EoNADE	-13.19	-12.58	-82.31	-85.10	-9.69	-272.39	-27.32	-46.12	-27.87
DARN	-13.19	-11.91	-81.04	**≈-84.13**	**-9.55**	-274.68	≈-28.17	≈-46.10	≈-28.83
MADE	-13.12	-11.90	-79.66	-86.64	-9.68	-277.28	-28.34	-46.74	-28.25
RWS-NADE	≈-13.16	**≈-11.68**	≈-84.26	≈-85.23	≈-9.71	**≈-271.11**	≈-26.43	≈-46.09	≈-27.92
single tree	-13.32	-13.83	-80.72	-93.41	-9.59	-283.14	-26.95	-49.00	-29.65
LBARN									
separate	-13.09	-12.10	-78.79	-86.69	-9.62	-272.95	**-24.60**	**-45.51**	**-27.60**
common	-13.29	-12.54	**-78.64**	-87.29	-9.71	-271.53	-24.74	-45.56	-27.67
linearized	**-13.07**	-12.14	-78.91	-86.86	-9.71	-271.72	-24.63	-45.54	-27.65
LBARN refit	-13.07	-12.06	-77.93	-86.49	-9.54	-272.32	-24.47	-45.35	-27.40
std. error	*0.023*	*0.007*	*0.320*	*0.215*	*0.023*	*0.488*	*0.085*	*0.058*	*0.103*
J	*2*	*4*	*16*	*32*	*8*	*2*	*64*	*128*	*32*
ν	*0.1*	*0.5*	*0.02*	*0.02*	*0.02*	*0.02*	*0.02*	*0.02*	*0.02*

Table 6.1: Average log-likelihoods (in nats) per test example for various models and datasets. The best result (without refitting) for each dataset is shown in bold. Dataset sizes, standard errors (when validating separately for each dimension) and selected hyperparameters are shown in italic. J is the selected number of leaves per tree and ν is the shrinkage factor.

autoregressive models. On five datasets our results are better than the best reported result for neural autoregressive networks. On the other four dataset the performance is competitive with the state of the art. For MNIST digits our model performs better than a neural network with a single layer of hidden units (NADE) and as good as a two-layer autoencoder with randomized connectivity pattern (MADE). The models which perform better than LBARN use a large ensemble of models for various orderings of the data dimensions (EoNADE) or rely on stochastic hidden units which make exact likelihood computations intractable (DARN, RWS-NADE). In Figure 6.1 we show samples of the learned model for MNIST digits, together with nearest neighbors from the training set. Table 6.1 also shows the test log-likelihoods after refitting the parameters using the pooled training and validation data. This always led to a small improvement. When comparing with previous results for fairness we only considered the model without refitting. Compared with mixtures of sparse autoregressive networks from Chapter 5 the generative performances are overall rather similar (cf. Tables 5.1 and 5.2). For larger datasets LogitBoost networks tend to perform a bit better and for smaller datasets the mixture networks give slightly better results.

As an additional comparison we trained autoregressive networks using a single probability

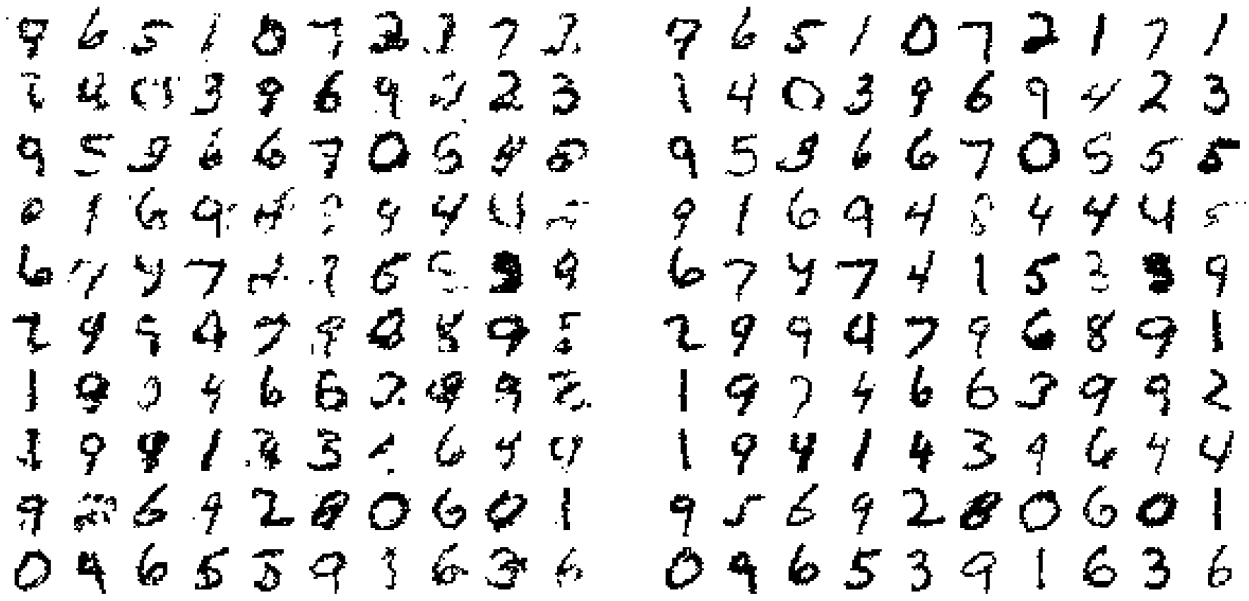

Figure 6.1: Left: Samples from LBARN trained on MNIST digits. Right: Closest training examples.

estimation tree for each dimension. The number of leaves for each tree was determined based on the performance on the validation set. In addition, small pseudocounts were used to regularize the probabilities. For each dataset the same regularization strength was chosen for all trees using the validation set. The obtained test log-likelihoods of these single-tree autoregressive networks can also be found in Table 6.1. With once exception boosted trees performed much better than single trees.

A variant of LogitBoost [40] is Gradient Boosting [42] which approximates the gradient of the Bernoulli log-likelihood to learn boosted trees rather than using a second-order expansion. We ran a few experiments with this alternative boosting method. The performance was often a bit worse than with LogitBoost. This confirms the observation made by Li [68] that second-order information helps the learning algorithm.

Figure 6.2 shows a map of the selected number of trees at each pixel for the MNIST dataset. For many dimensions around 400 trees were enough. However, a large fraction of the conditionals uses close to 1,000 trees which is the total number of boosting rounds. In those cases the holdout performance saturated, i.e., more trees do not lead to overfitting but

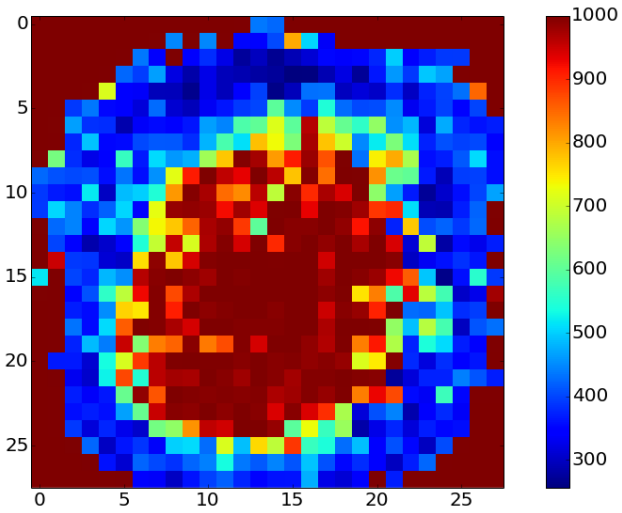

Figure 6.2: Map of the selected numbers of trees for the MNIST dataset.

the additional gain is very small. Training for up to 2,000 rounds of boosting increased the test log-likelihood from the original -86.69 only to -86.62.

We also explored the alternative model selection procedures mentioned in Section 6.2.1. For the procedure that creates a linear ordering of all trees we used the forward-selection variant. The obtained results for all three methods are also reported in Table 6.1. Separately validating each dimension worked well in most cases. Overfitting of the validation set only occurred for the NIPS-0-12 dataset which has 500 dimensions but contains merely 100 validation samples. Selecting a common number of trees for all dimensions worked poorly in most cases. The linearized model selection worked almost as well as separate selection and performed better for small validation sets. In Table 6.1 we also report the standard errors for the test log-likelihoods of LBARN on the different datasets when validating separately for each dimension.

An important hyperparameter which has to be chosen for LBARN is the number J of leaves for each tree. In Figure 6.3 we show the validation log-likelihood as a function of boosting rounds for three different choices of J on the OCR-letters dataset. The shrinkage factor was always $\nu = 0.02$. For $J = 32$ LogitBoost has not fully converged after 1,000 rounds of boosting. For $J = 64$ the best validation performance is obtained with around

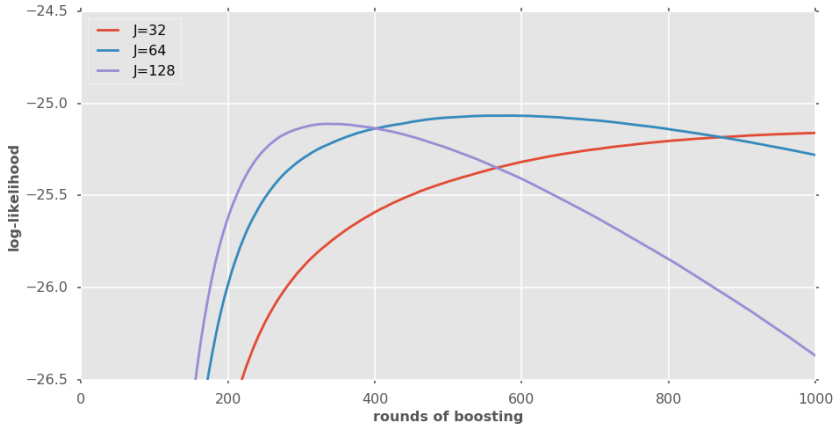

Figure 6.3: Validation log-likelihood on the OCR-letters dataset for different numbers of leaves.

600 rounds of boosting, after that slight overfitting begins. For $J = 128$ strong overfitting occurs after 400 rounds of boosting. What should be noticed is that the maximum of all three curves is almost the same. This means that the exact value of J is not that important as long as it is in the right range.

6.3.1 Ordering of the Variables

For any permutation π of the dimensions $\{1, \ldots, D\}$ the product of conditional distributions

$$\prod_{d=1}^{D} \mathbb{P}(\boldsymbol{x}(\pi(d)) \,|\, \boldsymbol{x}(\pi(1)), \ldots, \boldsymbol{x}(\pi(d-1)))$$

is equal to the joint distribution $\mathbb{P}(\boldsymbol{x})$. However, if the conditionals are learned from data then different decompositions yield different results. So far we always used the ordering of the variables in which they are stored in the computer. For specific applications alternative orderings can be more appropriate. For example, by using an ordering in which the first few variables provide as much information as possible about the complete data we can create a simple (noisy) compression mechanism. Such an ordering is characterized by the fact that the later conditional distributions have a low entropy. A greedy strategy to find an ordering

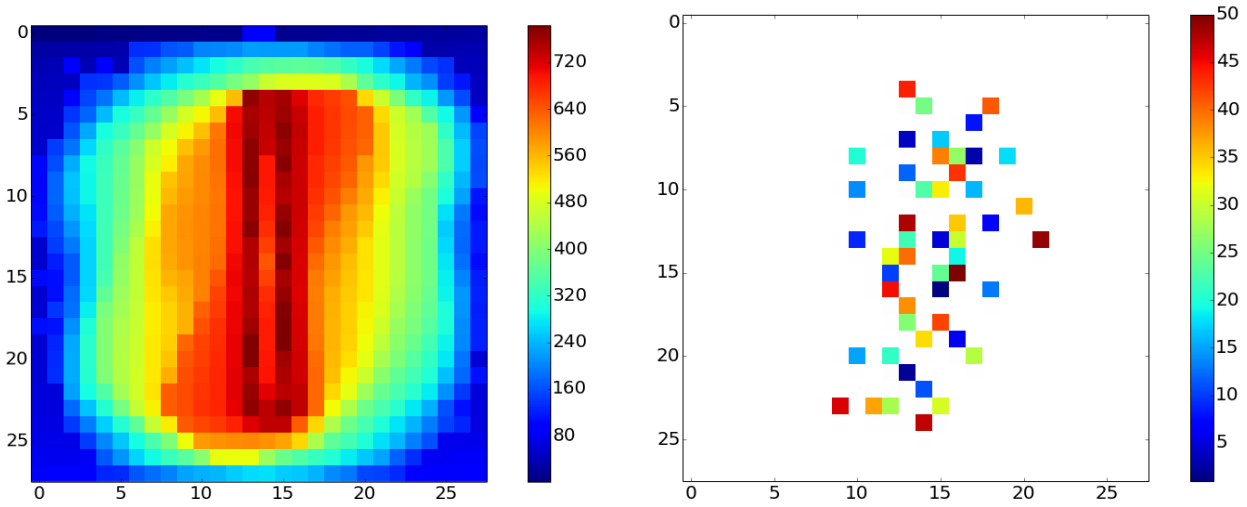

Figure 6.4: Orderings for MNIST digits. Left: Pixels ordered by increasing conditional entropy. Right: First 50 pixels ordered by decreasing conditional entropy.

with that property is to sequentially add the variable with the highest conditional entropy given all previously added variables. For more details about this approach see the work by Geman and Jedynak [48].

We used the training examples from the MNIST digits dataset to sort the $28 \times 28 = 784$ variables by decreasing conditional entropy. The right panel of Figure 6.4 visualizes the obtained ordering. For comparison we also show the learned ordering for increasing conditional entropy (left panel). The increasing-entropy order starts from background pixels at the boundary of the image grid and circulates towards the inside. The decreasing-entropy order on the other hand starts from central pixels and moves outwards. We illustrate the captured amount of information in Figure 6.5 by plotting the cumulative log-likelihoods $\sum_{d=1}^{D} \log \mathbb{P}(\boldsymbol{x}(\pi(d)) \,|\, \boldsymbol{x}(\pi(1)), \ldots, \boldsymbol{x}(\pi(d-1)))$ as a function of D. When sorted by increasing conditional entropy the variables at the end of the ordering contribute most. When sorted by decreasing conditional entropy most of the contributions to the log-likelihood come from variables at the beginning of the ordering. Nevertheless, the average test log-likelihoods for the full data are rather similar. The small gain from using informative variables early in the ordering is statistically significant though. The primary advantage of the decreasing-entropy

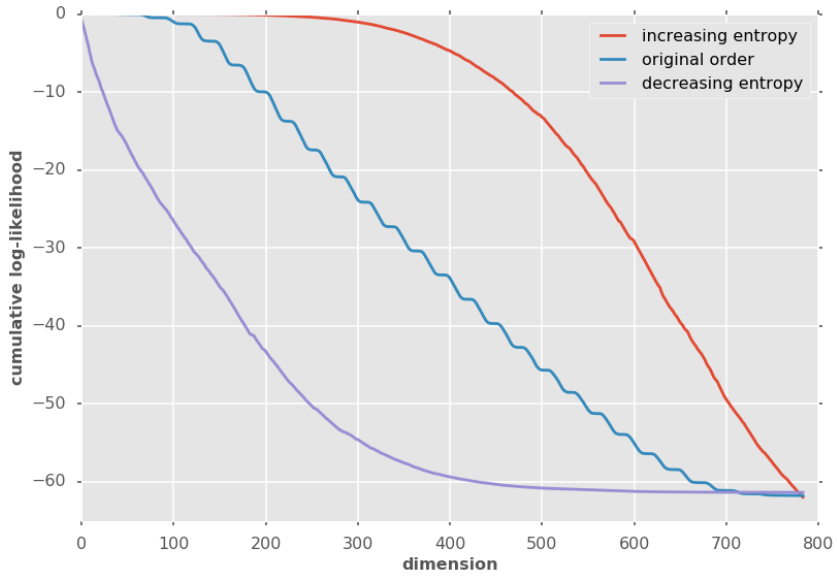

Figure 6.5: Cumulative log-likelihoods of the MNIST test digits for different orderings.

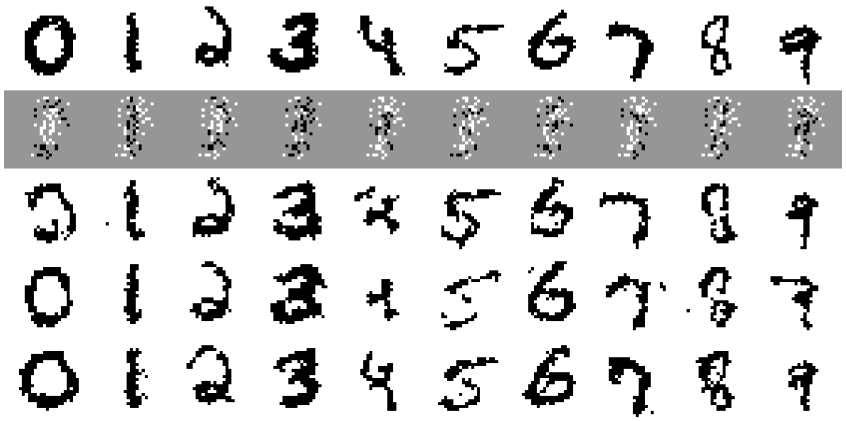

Figure 6.6: Compression of MNIST digits. 1st row: Test examples. 2nd row: Reduced data consisting of 78 pixels (gray indicates missing values). 3rd-5th row: Generated samples conditioned on the reduced data.

ordering is that we can reduce the data to a small fraction of the variables and can still reconstruct the full data well. This is illustrated in Figure 6.6 where we use examples from the MNIST test set and generate samples conditioned on the first 10% of the variables.

CHAPTER 7

CONCLUSION

We have explored several approaches for high-dimensional distribution estimation. The models in Chapter 2-4 assumed independence of the observable variables conditioned on the latent variables while the models in Chapter 5 and 6 used learned dependencies which were based on an autoregressive decomposition. Specifically, we started with regularized mixture models for which the number of components was bounded by the number of samples. We then studied compositional models in which mixture components were formed according to some composition rule. Different rules were discussed and new rules were introduced which led to very compact representations and which did a better job in disentangling factors of variation. The autoregressive network which we introduced after that achieved likelihood performance comparable to or even better than the best existing models. Broadly speaking, the models in this thesis were presented in increasing complexity. The models from the earlier chapters can be learned from relatively small datasets and generalize well. But they might underfit more complicated data distributions. The last two models on the other hand can fit rather complex distributions but at the same time require more data and are more vulnerable to overfitting.

There are many opportunities to extend or combine the presented approaches. For example, the hierarchical shrinkage procedure can be combined with existing methods which adapt the kernel bandwidth. Another possible extension of that method is an iterative refinement process for the sample paths. One could relax the tree structure and allow the node contributions to be shared between different branches. Each sample could then use any of the nodes from each layer of the hierarchy. It is also conceivable to apply hierarchical shrinkage to compositional models or autoregressive networks. Moreover, the maximum-expertise composition rule could be applied to autoregressive networks as well. Indeed, the conditional

distributions could be modeled as

$$\mathbb{P}(\boldsymbol{x}(d) \,|\, \boldsymbol{x}(1{:}d{-}1), \boldsymbol{h}) = \mathbb{P}_{k^\star(d)}(\boldsymbol{x}(d) \,|\, \boldsymbol{x}(1{:}d{-}1))$$

where $k^\star(d)$ is the active expert with the highest level of expertise for dimension d. Furthermore, we believe that our LogitBoost autoregressive network can still be improved through recent advances like Bayesian additive regression trees [21] or dropouts for additive regression trees [90]. Note however that there is nothing special about the choice of LogitBoost for the conditionals. Other state-of-the-art probability estimators might lead to similar results. The natural extension of the nonparametric autoregressive approach to real-valued data is to train separate regression estimators for each dimension using L2-boosting [42, 20], Gaussian process regression [91] or sparse additive models [92].

REFERENCES

[1] K. Abend, T. J. Harley, and L. N. Kanal. Classification of binary random patterns. *Information Theory*, 11(4):538–544, 1965.

[2] I. S. Abramson. On bandwidth variation in kernel estimates–a square root law. *The Annals of Statistics*, pages 1217–1223, 1982.

[3] J. Aghajanian and S. J. Prince. Mosaicfaces: A discrete representation for face recognition. In *IEEE Workshop on Applications of Computer Vision*, pages 1–8, 2008.

[4] M. Aharon, M. Elad, and A. Bruckstein. K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing*, 54(11):4311–4322, 2006.

[5] J. Aitchison and C. G. Aitken. Multivariate binary discrimination by the kernel method. *Biometrika*, 63(3):413–420, 1976.

[6] Y. Amit and A. Trouvé. Pop: Patchwork of parts models for object recognition. *International Journal of Computer Vision*, 75(2):267–282, 2007.

[7] L. R. Bahl, P. F. Brown, P. V. De Souza, and R. L. Mercer. A tree-based statistical language model for natural language speech recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37(7):1001–1008, 1989.

[8] D. J. Bartholomew, M. Knott, and I. Moustaki. *Latent Variable Models and Factor Analysis*. Wiley, 2011.

[9] S. Bengio and Y. Bengio. Taking on the curse of dimensionality in joint distributions using neural networks. *IEEE Transactions on Neural Networks*, 11(3):550–557, 2000.

[10] Y. Bengio. Discussion of the neural autoregressive distribution estimator. In *International Conference on Artificial Intelligence and Statistics*, pages 38–39, 2011.

[11] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[12] C. M. Bishop. Mixture density networks. Technical Report NCRG/94/004, Aston University, Birmingham, 1994.

[13] D. M. Blei and M. I. Jordan. Variational inference for Dirichlet process mixtures. *Bayesian Analysis*, 1(1):121–144, 2006.

[14] D. M. Blei, T. L. Griffiths, M. I. Jordan, and J. B. Tenenbaum. Hierarchical topic models and the nested chinese restaurant process. *Advances in Neural Information Processing Systems*, 16:17, 2004.

[15] E. Borenstein and S. Ullman. Combined top-down/bottom-up segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(12):2109–2125, 2008.

[16] J. Bornschein and Y. Bengio. Reweighted wake-sleep. In *International Conference on Learning Representations*, 2015.

[17] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in bayesian networks. In *International Conference on Uncertainty in Artificial Intelligence*, pages 115–123, 1996.

[18] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and Regression Trees*. CRC press, 1984.

[19] J. Bruna and S. Mallat. Invariant scattering convolution networks. *Pattern Analysis and Machine Intelligence*, 35(8):1872–1886, 2013.

[20] P. Bühlmann and B. Yu. Boosting with the L-2 loss: Regression and classification. *Journal of the American Statistical Association*, 98(462):324–339, 2003.

[21] H. A. Chipman, E. I. George, and R. E. McCulloch. BART: Bayesian additive regression trees. *The Annals of Applied Statistics*, pages 266–298, 2010.

[22] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

[23] A. L. Comrey and H. B. Lee. *A First Course in Factor Analysis*. Psychology Press, 2013.

[24] G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine learning*, 9(4):309–347, 1992.

[25] N. Cressie and J. L. Davidson. Image analysis with partially ordered markov models. *Computational Statistics & Data Analysis*, 29(1):1–26, 1998.

[26] S. Davies and A. Moore. Mix-nets: Factored mixtures of Gaussians in Bayesian networks with mixed continuous and discrete variables. In *Conference on Uncertainty in Artificial Intelligence*, pages 168–175, 2000.

[27] P. Dayan and R. S. Zemel. Competition and multiple cause models. *Neural Computation*, 7(3):565–579, 1995.

[28] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (methodological)*, pages 1–38, 1977.

[29] J. Domke, A. Karapurkar, and Y. Aloimonos. Who killed the directed model? In *Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[30] D. L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52 (4):1289–1306, 2006.

[31] M. Elad. *Sparse and Redundant Representations*. Springer, 2010.

[32] S. A. Eslami, N. Heess, C. K. Williams, and J. Winn. The shape Boltzmann machine: A strong model of object shape. *International Journal of Computer Vision*, 107(2): 155–176, 2014.

[33] T. Evgeniou and M. Pontil. Regularized multi–task learning. In *International Conference on Knowledge Discovery and Data Mining*, pages 109–117, 2004.

[34] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding*, 106(1):59–70, 2007.

[35] J. Felsenstein. Maximum-likelihood estimation of evolutionary trees from continuous characters. *American Journal of Human Genetics*, 25(5):471, 1973.

[36] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *Conference on Computer Vision and Pattern Recognition*, volume 2, pages 264–271, 2003.

[37] C. Ferri, P. A. Flach, and J. Hernández-Orallo. Improving the AUC of probabilistic estimation trees. In *European Conference on Machine Learning*, pages 121–132, 2003.

[38] S. Fidler and A. Leonardis. Towards scalable representations of object categories: Learning a hierarchy of parts. In *Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[39] B. J. Frey. *Graphical Models for Machine Learning and Digital Communication*. MIT press, 1998.

[40] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28(2):337–407, 2000.

[41] J. Friedman, T. Hastie, and R. Tibshirani. *The Elements of Statistical Learning*. Springer, Berlin, 2009.

[42] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, pages 1189–1232, 2001.

[43] N. Friedman and M. Goldszmidt. Learning Bayesian networks with local structure. In *Learning in Graphical Models*, pages 421–459. Springer, 1998.

[44] Z. Gan, R. Henao, D. Carlson, and L. Carin. Learning deep sigmoid belief networks with data augmentation. In *International Conference on Artificial Intelligence and Statistics*, pages 268–276, 2015.

[45] J. Gao and D. B. Hitchcock. James–Stein shrinkage to improve k-means cluster analysis. *Computational Statistics & Data Analysis*, 54(9):2113–2127, 2010.

[46] M. Gavish, B. Nadler, and R. R. Coifman. Multiscale wavelets on trees, graphs and high dimensional data: Theory and applications to semi supervised learning. In *International Conference on Machine Learning*, pages 367–374, 2010.

[47] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian Data Analysis*. 2013.

[48] D. Geman and B. Jedynak. An active testing model for tracking roads in satellite images. *Pattern Analysis and Machine Intelligence*, 18(1):1–14, 1996.

[49] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *Pattern Analysis and Machine Intelligence*, (6):721–741, 1984.

[50] S. Geman, D. F. Potter, and Z. Chi. Composition systems. *Quarterly of Applied Mathematics*, 60(4):707–736, 2002.

[51] M. Germain, K. Gregor, I. Murray, and H. Larochelle. MADE: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pages 881–889, 2015.

[52] K. Gregor, I. Danihelka, A. Mnih, C. Blundell, and D. Wierstra. Deep autoregressive networks. In *International Conference on Machine Learning*, pages 1242–1250, 2014.

[53] T. Hastie and R. Tibshirani. Discriminant analysis by gaussian mixtures. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 155–176, 1996.

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

[55] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[56] W. James and C. Stein. Estimation with quadratic loss. In *Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 361–379, 1961.

[57] I. Jolliffe. *Principal Component Analysis*. Wiley, 2014.

[58] J. Kim and C. D. Scott. Robust kernel density estimation. *The Journal of Machine Learning Research*, 13(1):2529–2565, 2012.

[59] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT press, 2009.

[60] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956.

[61] S. Kullback. *Information Theory and Statistics*. Courier Corporation, 1968.

[62] B. M. Lake, R. Salakhutdinov, and J. Tenenbaum. One-shot learning by inverting a compositional causal process. In *Advances in Neural Information Processing Systems*, pages 2526–2534, 2013.

[63] H. Larochelle and I. Murray. The neural autoregressive distribution estimator. In *International Conference on Artificial Intelligence and Statistics*, pages 29–37, 2011.

[64] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin. Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, 10(Jan):1–40, 2009.

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

[66] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *International Conference on Machine Learning*, pages 609–616, 2009.

[67] H. Lee, R. Raina, A. Teichman, and A. Y. Ng. Exponential family sparse coding with application to self-taught learning. In *International Joint Conference on Artificial Intelligence*, volume 9, pages 1113–1119, 2009.

[68] P. Li. Robust LogitBoost and adaptive base class (abc) LogitBoost. In *Conference on Uncertainty in Artificial Intelligence*, pages 302–311, 2010.

[69] S. Z. Li. *Markov Random Field Modeling in Image Analysis*. Springer Science & Business Media, 2009.

[70] M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`.

[71] H. Liu, J. D. Lafferty, and L. A. Wasserman. Sparse nonparametric density estimation in high dimensions using the rodeo. In *International Conference on Artificial Intelligence and Statistics*, pages 283–290, 2007.

[72] J. Lücke and M. Sahani. Maximal causes for non-linear component extraction. *The Journal of Machine Learning Research*, 9:1227–1267, 2008.

[73] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online dictionary learning for sparse coding. In *International Conference on Machine Learning*, pages 689–696, 2009.

[74] S. Mallat. *A Wavelet Tour of Signal Processing*. Academic press, 2008.

[75] S. G. Mallat and Z. Zhang. Matching pursuits with time-frequency dictionaries. *Signal Processing*, 41(12):3397–3415, 1993.

[76] B. M. Marlin, K. Swersky, B. Chen, and N. D. Freitas. Inductive principles for restricted Boltzmann machine learning. In *International Conference on Artificial Intelligence and Statistics*, pages 509–516, 2010.

[77] G. McLachlan and D. Peel. *Finite Mixture Models*. John Wiley & Sons, 2004.

[78] D. Mease, A. J. Wyner, and A. Buja. Boosted classification trees and class probability/quantile estimation. *The Journal of Machine Learning Research*, 8:409–439, 2007.

[79] E. Meeds, Z. Ghahramani, R. M. Neal, and S. T. Roweis. Modeling dyadic data with binary latent factors. In *Advances in Neural Information Processing Systems*, pages 977–984, 2006.

[80] M. Meila and M. I. Jordan. Learning with mixtures of trees. *Journal of Machine Learning Research*, 1:1–48, 2001.

[81] N. Meinshausen and P. Bühlmann. High-dimensional graphs and variable selection with the lasso. *The Annals of Statistics*, pages 1436–1462, 2006.

[82] I. Murray and R. R. Salakhutdinov. Evaluating probabilities under high-dimensional latent variable models. In *Advances in Neural Information Processing Systems*, pages 1137–1144, 2008.

[83] R. M. Neal. Connectionist learning of belief networks. *Artificial Intelligence*, 56(1): 71–113, 1992.

[84] B. A. Olshausen and D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:13, 1996.

[85] C. Pal, B. J. Frey, and N. Jojic. Learning montages of transformed latent images as representations of objects that change in appearance. In *Computer Vision–ECCV*, pages 715–731. 2002.

[86] H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *International Conference on Computer Vision (Workshops)*, pages 689–690, 2011.

[87] F. Provost and P. Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215, 2003.

[88] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[89] T. Raiko, Y. Li, K. Cho, and Y. Bengio. Iterative neural autoregressive distribution estimator nade-k. In *Advances in Neural Information Processing Systems*, pages 325–333, 2014.

[90] K. V. Rashmi and R. Gilad-Bachrach. DART: Dropouts meet multiple additive regression trees. In *International Conference on Artificial Intelligence and Statistics*, pages 489–497, 2015.

[91] C. E. Rasmussen. Gaussian processes for machine learning. 2006.

[92] P. Ravikumar, J. Lafferty, H. Liu, and L. Wasserman. Sparse additive models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 71(5):1009–1030, 2009.

[93] P. Ravikumar, M. J. Wainwright, and J. D. Lafferty. High-dimensional ising model selection using $\ell 1$-regularized logistic regression. *The Annals of Statistics*, 38(3):1287–1319, 2010.

[94] R. Salakhutdinov and I. Murray. On the quantitative analysis of deep belief networks. In *International Conference on Machine learning*, pages 872–879, 2008.

[95] E. Saund. A multiple cause mixture model for unsupervised learning. *Neural Computation*, 7(1):51–71, 1995.

[96] A. I. Schein, L. K. Saul, and L. H. Ungar. A generalized linear model for principal component analysis of binary data. In *International Workshop on Artificial Intelligence and Statistics*, volume 38, page 46, 2003.

[97] N. Shafik and G. Tutz. Boosting nonlinear additive autoregressive time series. *Computational Statistics & Data Analysis*, 53(7):2453–2464, 2009.

[98] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*, volume 26. 1986.

[99] J. S. Simonoff. *Smoothing Methods in Statistics*. Springer Science & Business Media, 2012.

[100] M. Steinbach, G. Karypis, V. Kumar, et al. A comparison of document clustering techniques. In *KDD Workshop on Text Mining*, volume 400, pages 525–526, 2000.

[101] Y. Tang, R. Salakhutdinov, and G. Hinton. Deep mixtures of factor analysers. In *International Conference on Machine Learning*, pages 505–512, 2012.

[102] G. R. Terrell and D. W. Scott. Variable kernel density estimation. *The Annals of Statistics*, pages 1236–1265, 1992.

[103] T. Tieleman. Training restricted Boltzmann machines using approximations to the likelihood gradient. In *International Conference on Machine learning*, pages 1064–1071, 2008.

[104] B. Uria, I. Murray, and H. Larochelle. Rnade: The real-valued neural autoregressive density-estimator. In *Advances in Neural Information Processing Systems*, pages 2175–2183, 2013.

[105] B. Uria, I. Murray, and H. Larochelle. A deep and tractable density estimator. In *International Conference on Machine Learning*, pages 467–475, 2014.

[106] L. van der Maaten. A new benchmark dataset for handwritten character recognition. Technical report, TiCC TR 2009-002, Tilburg University, 2009.

[107] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning*, pages 1096–1103, 2008.

[108] L. Wasserman. *All of Nonparametric Statistics*. Springer Science & Business Media, 2006.

[109] C. K. Williams and M. K. Titsias. Greedy learning of multiple objects in images using robust statistics and factorial learning. *Neural Computation*, 16(5):1039–1062, 2004.

[110] J. W. Woods and C. Radewan. Kalman filtering in two dimensions. *Information Theory*, 23(4):473–482, 1977.

[111] L. L. Zhu, C. Lin, H. Huang, Y. Chen, and A. Yuille. Unsupervised structure learning: Hierarchical recursive composition, suspicious coincidence and competitive exclusion. In *Computer Vision–ECCV*, pages 759–773. 2008.

[112] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.