Logical Factorisation Machines

Probabilistic Boolean Factor Models for Binary Data

Tammo Nikolas Fenner Rukat

Department of Statistics

University of Oxford

This dissertation is submitted for the degree of

Doctor of Philosophy

Kellogg College

Trinity 2018

Abstract

Logical Factorisation Machines (LFMs) are a class of latent feature models, that aim to decompose binary matrices, tensors or higher-arity relations into an approximate logical product of low rank, binary matrices. These products are defined through the use of logical operators instead of arithmetic operations. The resulting factor matrices contain interpretable patterns that lend themselves to the discovery of hidden causal structure. We frame LFMs as probabilistic generative models and derive sampling based posterior inference. Despite full uncertainty quantification, the inference procedure scales to Billions of data points which is possible through exploitation of the logical structure in the factor conditionals.

OrMachines are a particularly interesting subset of LFMs, where a single latent cause is sufficient to explain an outcome. They represent a probabilistic approach to the well-studied problems of Boolean Matrix Factorisation and Boolean Tensor Factorisation. The proposed model and inference procedure yield decompositions of higher accuracy than the existing techniques throughout a wide range of conditions. Real-world examples include single-cell genomics, cancer genomics, relational and spatio-temporal data. We propose several extensions, including hierarchies of OrMachines for data integration and a Bayesian nonparametric approach to infer the latent dimensionality.

LFMs with less established logics, their relationships and interpretation are considered. While the latent structure in many of these models is inherently difficult to recover, we demonstrate that certain LFMs can provide complementary representations and reveal new findings. A fast and flexible implementation is introduced and publicly available on GitHub.

Parts of this thesis have been published as follows:

- **Rukat, Tammo**, Holmes, Chris C., Titsias, Michalis K., and Yau, Christopher (2017). Bayesian Boolean Matrix Factorisation. *Proceedings of the 34th Annual International Conference on Machine Learning*, pp. 2969–2978, Jul 2017, Sydney, Autralia
- **Rukat, Tammo**, Lange, Dustin, and Archambeau, Cédric (2017). An Interpretable Latent Variable Model for Attribute Applicability in the Amazon Catalogue. Proceedings of the NIPS 2017 Symposium on Interpretable Machine Learning.
- **Rukat, Tammo**, Holmes, C., and Yau, C. (2018). Probabilistic Boolean Tensor Decomposition. *Proceedings of the 35th International Conference on Machine Learning*, pp. 4413–4422, Jul 2018, Stockholm, Sweden
- **Rukat, Tammo** and Yau, Christopher (2018). Nonparametric Boolean Factor Models Accepted for publication at the NIPS 2018 workshop All of Bayesian Nonparametrics

Additional oral presentations have been given at the following occasions:

- Advances in Data Science, 15-16 May 2017, Museum of Science & Industry, Manchester
- The Sixteenth International Symposium on Intelligent Data Analysis (IDA) - 26-28 October 2017, Birkbeck, University of London
- CogX, The Festival of All Things AI, Blockchain and Emerging Technologies, 11-12 June 2018, Tobacco Dock, London

Additional posters have been presented at:

- Statistical Methods for Postgenomic Data (SMPGD), 12-13 January 2017, Imperial College London, London
- Artificial Intelligence at Oxford (AI Expo), 27 March 2018, Oxford
- International Society for Bayesian Analysis (ISBA) World Meeting, 24-29 June 2018, Edinburgh

Acknowledgements

First and foremost, I would like to thank Chris Yau for guiding me through the peaks and troughs of this DPhil and for being an incredibly knowledgeable teacher and a great companion. I am further grateful to Chris Holmes for his supervision, enthusiasm and great mentorship. Thanks to Neil Lawrence and to Dino Sejdinovic for their insightful comments and for examining this thesis in great depths. Thanks also to Michalis Titsias for his insights and advice and to Miguel Lázaro-Gredilla for sharing helpful perspectives on this work. I would further like to thank Satu Nahkuri and Fabian Schmich at Hoffmann-La Roche for their support and collaboration.

I am grateful to Charlotte Deane for setting up the SABS CDT programme that enabled me to come to Oxford and to pursue this research. Thanks to Stefan Kowarik for sparking my enthusiasm about research during my undergraduate years and for sending me to Cornell. A special gratitude goes to Stefan Reinsberg for being a wonderful supervisor during my master's degree, for introducing me to Bayesian Statistics and for taking me kitesurfing.

I am very thankful to Florian Klimm for being a great friend and *partner in crime* during my time in Oxford and beyond, without him I would not have made this journey. Thanks to Luke Kelly, especially for the best games of squash. Their importance for this work must not be underestimated. I would also like to thank my friends in Oxford who have made this time so much fun, Anna Niedballa, Bernadette Stolz, Christoph Prätzer, Hyun-Jik Kim, Julia Wiedmann, Martin Strohmeier, Max Schleich and Sarah Griffin.

I am deeply thankful to my parents and my family, Annika, Bodo and Julius for their love and support. Finally, to Marie. When we met, this work began to fall into place, and so did everything else.

Table of Contents

1 Introduction **1**

1.1 A Motivating Example 1

1.2 Latent Variable Models for Binary Data 3

1.3 Boolean Factorisation and Multicausal Interactions 4

1.4 Thesis Overview 8

2 Bayesian Boolean Matrix Factorisation **11**

2.1 Related Work 12

2.2 OrMachine: Bayesian Boolean Matrix Factorisation 14

2.2.1 Model Formulation 14

2.2.2 Fast Posterior Inference for Factor Matrices 16

2.2.3 MAP Inference for the Dispersion Parameter 19

2.2.4 Missing Data and Imputation 21

2.2.5 Generalisation to Multiple Layers 22

2.3 Experiments on Simulated Data 24

2.3.1 Random Matrix Factorisation 25

2.3.2 Random Matrix Completion 27

2.4 Real-World Benchmark: MovieLens 28

2.5 Explorative Analysis at Scale: Single Cell Genomics 30

2.6 Implementation of a Simple Factor Model 35

2.7 Hierarchies of OrMachines for Data Integration 40
2.8 Community Detection for Network Analysis 44
2.8.1 Bottom-up Emergence of Communities 45
2.8.2 Application of the OrMachine to Community Detection 46
2.9 Infinite OrMachine 48
2.9.1 Posterior Inference 50
2.9.2 Evaluation on Simulated Data 55
2.9.3 Practical Applicability of the IBP-OrMachine 56
2.10 Conclusion 56

3 Boolean Factorisation of Polyadic Data 59

3.1 Related Work 62
3.2 Inference and Prediction 62
3.3 Experiments on Simulated Data 66
3.3.1 Random Tensor Factorisation 66
3.3.2 Model Selection 69
3.4 Real-World Applications 71
3.4.1 Hospital Ward Interaction Networks 71
3.4.2 Student Seating Position 72
3.4.3 Networks of Relative Gene Expression in Cancer 74
3.4.4 Multiple-Tissue Gene Expression 77
3.5 Conclusion 80

4 Logical Factorisation Machines 81

4.1 Generalisation of Logical Operators 82
4.2 Conversion between Logical Factor Machines 84
4.3 A Taxonomy of Logical Factorisation Machines 86

4.3.1 The AND-AND Clan 88
4.3.2 The AND-NAND Clan 89
4.4 Expected Factor Densities for LFMs 93
4.5 Factor Conditionals 96
4.6 Data Reconstruction 98
4.7 Machines Containing XOR-Operations 98
4.7.1 The XOR-AND Class 98
4.7.2 The AND-XOR-Class 101
4.7.3 The XOR-XOR-Class 103
4.8 Reconstruction Accuracies for LFM Model Selection 106
4.9 Recovering the Data-Generating Logic 108
4.10 Advanced Usage of our Implementation 110
4.11 Conclusion 112

5 Discussion 113
5.1 Concluding Remarks 113
5.2 Future Work 115

References 117

Appendix A Formal Derivation of Factor Conditionals 125

Appendix B IBP Experiment with Noise 127

Appendix C Derivations for the Dispersion Parameter 129

Appendix D LFMs: Derivations and Examples 131
D.1 Conditional Distributions 131
D.2 Mean-Field Reconstruction 133

Chapter 1

Introduction

1.1 A Motivating Example

Figure 1.1 attempts to visualise a small subset of data from a typical genomics experiment: Around 1.5 billion black or white dots indicate the presence or absence of gene expression for 12 thousand genes across hundreds of thousands of single cells. No customary printer or screen has a sufficient pixel density to fully display these dots, let alone the human ability to visually process this information. This depiction may not seem very informative, but considering that the human visual system has the greatest bandwidth of all our sensory inputs, it underlines the question:

> How can we use such data to extend our understanding of the underlying processes, generate testable hypotheses and get to meaningful insights?

This thesis contributes to answering this question, by developing novel, probabilistic and scalable latent variable models that summarise binary datasets through latent factors, endowed with a clear, insightful interpretation. It is important to note, that this motivation generalises far beyond applications in biology. Recent years have not only witnessed an exponential growth in available data, but also breakthroughs in machine learning techniques

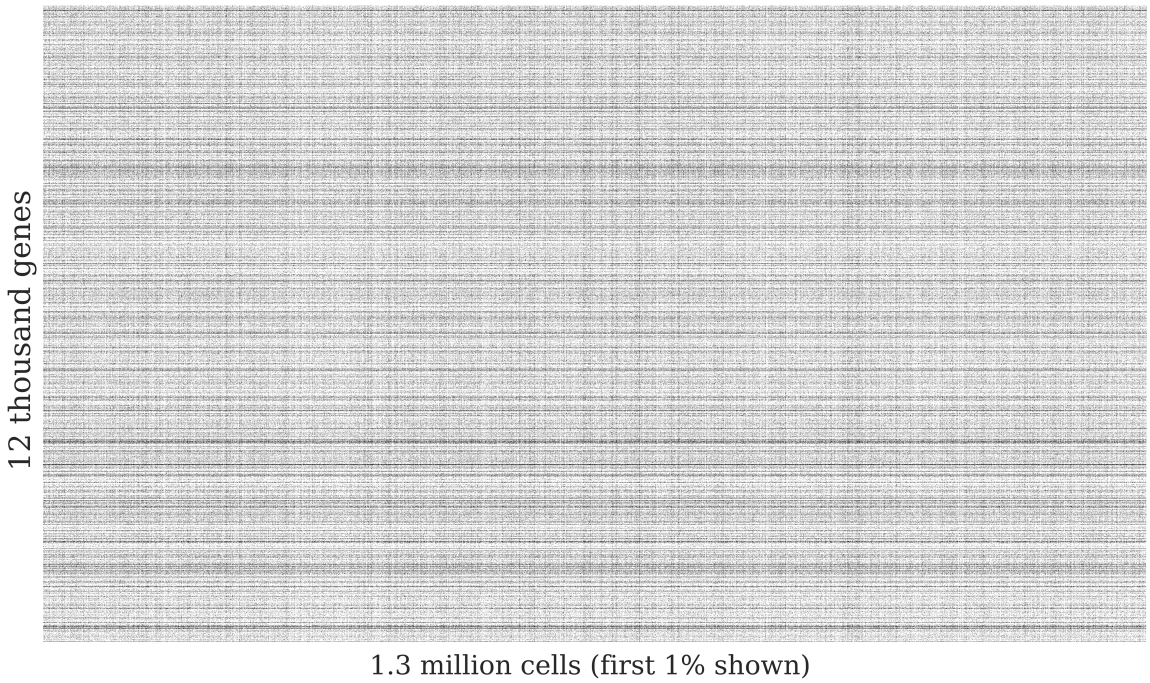


Fig. 1.1 Single cell gene-expression in mouse cells from different brain regions. The original data indicates counts of gene transcripts. The number of sequencing reads per nucleotide is low, such that binarising the data retains the essential information of whether a gene is expressed. Black dots indicate expressed genes. The data is publicly available from `https://support.10xgenomics.com` (retrieved in December 2016).

that capture complex correlations and feature unprecedented predictive abilities. Many of these techniques operate as black-boxes in utter separation from a human understanding of the world and, thus, they are incapable of contributing to an answer of the question above. Conversely, the development of methods that do comply with human representations of knowledge or allow for causal interpretation have not kept pace with the rise of big data.

The goal for the remainder of this chapter is to motivate the particular models that we develop to address this gap and to put them, as well as their applications, into a wider context. For an in-depth analysis of the dataset in Fig. 1.1, we ask the reader to exercise patience until Section 2.5.

1.2 Latent Variable Models for Binary Data

An important approach to making complex, high-dimensional datasets amenable to human insight are latent variable models. Many of these models can be understood as a type of matrix factorisation that summarises a data-matrix by a set of abstract properties and a set of latent variables. These variables indicate how the properties manifest in the data and provide a compressed representation. Observations are modelled as conditionally independent given the latent variables.

Here, we focus on binary outcomes such as *True* and *False* that are abundant in many domains of application. In contrast to standard numerical data, the domain of binary data are exclusive categories. The representation as numerical values $\{0, 1\}$ or $\{-1, 1\}$ is arbitrary and chosen for convenience rather than correspondence to the numerical values that these symbols reference. Compared to real-valued data, these categories may lack a certain equivalence in their meaning. For instance, the presence of disease is usually of greater interest than its absence. At the same time, absence of evidence is not equivalent to evidence of absence, whereas presence of evidence is usually evidence for presence. In the application of latent variable models to binary data, many practitioners resort to ill-suited models such as Principal Component Analysis (PCA) that have been developed for real-valued observations. An example of such an abuse is given in Fig. 1.2 and will be discussed in the context of Boolean matrix factorisation in the next section.

Existing latent variable models that are tailored for binary data include generalisation of PCA to the exponential family and thus to Bernoulli random variables (Collins et al., 2002; Mohamed et al., 2009; Tipping, 1999). Further candidates are Bernoulli mixture models (Section 9.3.3 in Bishop, 2013; Juan and Vidal, 2004) and models based on the noisy-OR (Eck et al., 2009; Šingliar and Hauskrecht, 2006; Wood et al., 2012) that we will discuss in more detail in the next section.

1.3 Boolean Factorisation and Multicausal Interactions

Latent variable models specify a generative process for the data that often lends itself to causal interpretation. The language of causality will occasionally be used throughout this thesis and will be instructive in our discussion. It should, however, be emphasised that the relationships we deal with are defined by joined probability distributions that describe associations rather than causation (Pearl et al., 2009). Therefore, the reader should keep in mind that causal interpretations may provide an instructive narrative but are not formally justified.

The contributions of this thesis evolve around a model that can be interpreted as describing multicausal interactions, where the latent variables are thought of as hidden causes whose interaction follows a particular logic. Conventional models often assume a logic whereby outcomes of interest are related to all different underlying causes. For instance in PCA, an observation is modelled as a weighted combination of latent states. When modelling risk factors for heart-disease, a schematic representation of such a model can be written as

$$\text{heart disease} = \text{obesity } \texttt{AND} \text{ smoking } \texttt{AND} \text{ exercise } \texttt{AND} \text{ genetics.}$$

Whilst these contributory factors exist in the general population, in any individual, only a single factor may be required to trigger the outcome,

$$\text{heart disease} = \text{obesity } \texttt{OR} \text{ smoking } \texttt{OR} \text{ exercise } \texttt{OR} \text{ genetics.}$$

Similarly, for the introductory example, a gene will be expressed if one or more of the relevant biological processes are active.

Generating data according to this logic can be formalised as a special matrix product. The data is a binary matrix, X, that indicates presence or absence of D attributes (diagnoses, genes) in N objects (individuals, cells). It is generated by the product of two binary factor

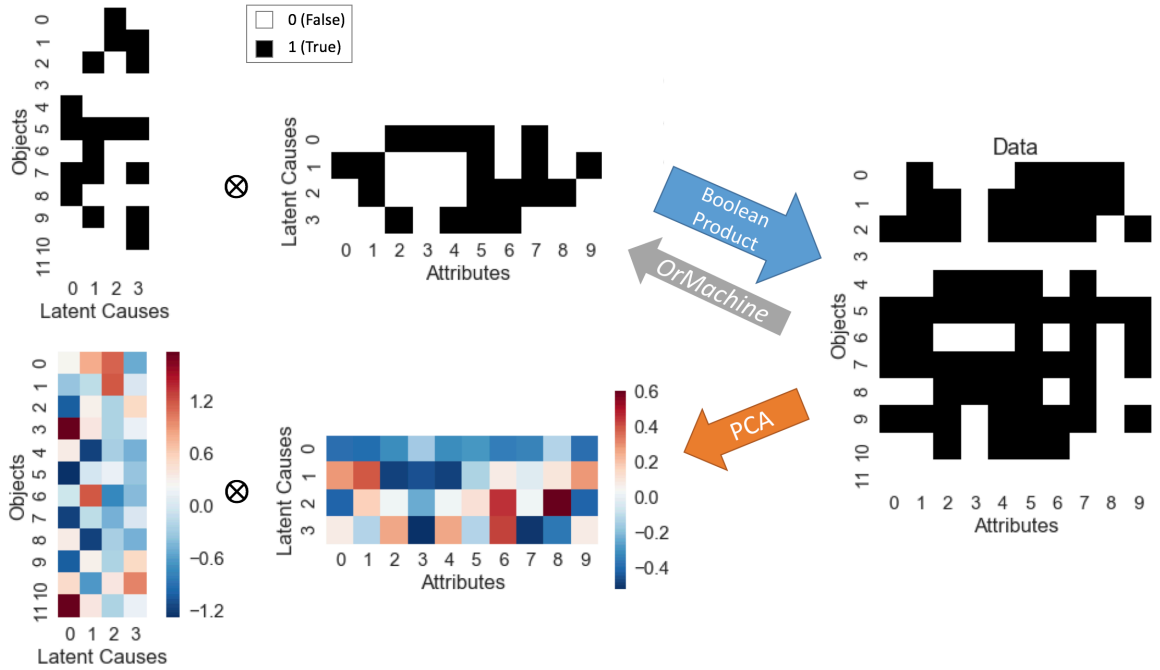


Fig. 1.2 Binary data generated according to a Boolean product of two random matrices. Based on its representation in $\{0, 1\}$ coding, the data is analysed using principal component analysis. The first four principal components are shown as heatmaps and ordered in the attempt to resemble the underlying Boolean factor matrices.

matrices U and Z: The rows of U specify a number, L, of latent causes, each defined through a set of associated attributes. Conversely, the columns of Z indicates presence or absence of latent causes in each object. The Boolean product formalises this logic:

$$x_{nd} = \bigvee_{l=1\ldots L} \left[\mathtt{AND}(z_{nl}, u_{ld})\right] \quad . \tag{1.1}$$

Here, `OR` and `AND` denote the Boolean disjunction and conjunction, respectively. In the language of our genomics example, the equation states that a gene is expressed in a cell, n, if and only if there exist one or more latent biological processes, l, that are associated to the gene and to the cell ($z_{nl} = u_{ld} = \mathtt{True}$).

We generate data according to this procedure from two random matrices and apply principal component analysis to the resulting data matrix as shown in Fig. 1.2. Even though

the 4-dimensional continuous embedding explains around 95% of the variation in the data, it does not reveal the simple binary structure but instead provides us with continuous latent variables whose interactions are difficult to interpret.

Inference in Boolean Matrix Factorisation amounts to determining the optimal factors, U and Z, for a given data matrix. The first approaches to this problem aim at an exact factorisation, in which case it is equivalent to the NP-complete biclique-cover problem (Gimpel, 1974; Vazirani, 2013): *Given a set of finite sets, find a basis with minimal cardinality such that each set can be represented as a union of a subset of the basis*. Here, however, we consider the problem of approximate factorisation that has been formalised in the context of biological data analysis (Nau et al., 1978) and rule-mining (Agrawal et al., 1994) and has been shown to be NP-hard by Miettinen et al. (2006). We provide a more thorough review of the related literature in Section 2.1.

Developing a state-of-the-art algorithm for Boolean Matrix Factorisation, named *OrMachine*, is one of the contributions of this thesis. To this end, we adopt a probabilistic approach which provides a principled strategy to dealing with uncertainties, missing data, prediction and model selection. For this introductory discussion, we merely state the model likelihood, whereas specification of the full joint probability distribution is postponed to Chapter 2. The likelihood factorises over all data points and is given by the following distribution, parametrised by $q \in [½, 1]$:

$$p(x_{nd} = \texttt{True} | \mathbf{u}, \mathbf{z}) = \begin{cases} q; & \text{if } \exists\ l : z_{nl} = u_{ld} = \texttt{True} \\ 1-q; & \text{otherwise .} \end{cases} \tag{1.2}$$

As $q \to 1$, the model describes the deterministic Boolean product from eq. (1.1), whereas x_{nd} is uniformly random for $q \to ½$. This simple factorial likelihood and the resulting logical dependences in the full conditional distribution over the factors will enable us to develop highly scalable, sampling-based posterior inference.

The likelihood in eq. (1.2) has an interesting relation to the noisy-OR model (Pearl, 1988). In $\{0, 1\}$ representation and with $r \in [0, 1]$ its likelihood can be written as

$$p(x_{nd} = \texttt{False}|\mathbf{u}, \mathbf{z}) = \prod_l (1 - r)^{z_{nl} u_{ld}} \ . \tag{1.3}$$

The model makes the assumption of *exception independence*, that is, independence between the inhibition of active latent causes. More formally, any active latent dimensions (l, where $z_{nl}u_{ld} = 1$) has a probability of failure, r. Thus, the likelihood of an event ($x_{nd} = \texttt{True}$) increases with the number of active latent causes. Noisy-OR latent variable models have been studied in the context of image analysis and relational data (Šingliar and Hauskrecht, 2006), medical data (Wood et al., 2012) and music classification (Eck et al., 2009). Techniques for approximate inference have been developed by Jaakkola and Jordan (1999). Our probabilistic model for Boolean factorisation violates exception independence with inhibition occurring on the event-level rather than on the cause-level. As a consequence more than one latent cause will not make the outcome more likely. In order to maximise the likelihood, it is sufficient to associate a single cause with any effect.

To give an example, consider the event that you invite your colleagues to the pub which may have two possible causal reasons (i) your paper has been accepted for publication or (ii) you have received a promotion. An inhibition on the level of causes might occur because you did not actually want the promotion. In that case, the accepted paper increases the likelihood of celebration. This intuition is captured by a noisy-OR. Opposed to this, inhibition on the event-level may occur when the pub is closed. This case is naturally described by a Boolean factor model, where additional reasons to go to the pub will not change the inhibiting fact of closure. Compared to the noisy-OR model, Boolean factorisation aims to associate exactly one hidden cause with every observation, favouring sparse representations. Moreover, as suggested by the heart disease example above, even if we were to believe in exception independence, it is usually only one latent cause that triggers the event. Thus, for many practical

purposes Boolean factorisation models are an interesting alternative. Further discussion of these opposing assumptions is given in the context of community detection in Section 2.8.

1.4 Thesis Overview

We continue with a detailed treatment of the OrMachine in Chapter 2. The eponymous family of models, Logical Factorisation Machines (LFMs), is derived through generealisation of the OrMachine in two ways that constitute chapters 3 and 4, respectively. Firstly, the logical product of binary matrices is extended to multi-way interactions of higher arity. In particular, a K-dimensional tensor can be factorised into K binary matrices, such that

$$x_{[\mathbf{n}]} \approx \underset{l=1\dots L}{\texttt{OR}} \left[\underset{n_k \in [\mathbf{n}]}{\texttt{AND}} (f_{n_k l}) \right] \quad . \tag{1.4}$$

Here, $x_{[\mathbf{n}]}$ is a single tensor entry and $[\mathbf{n}]$ denotes a tuple of indices $[n_1, \dots, n_K]$. We focus on the case of Boolean Tensor Factorisation, where $K = 3$. Both, Boolean Matrix and Boolean Tensor Factorisation have previously been tackled by various researchers, but are solved by our approach with unrivalled speed and accuracy.

For the second generalisation, we replace the `OR` and `AND` operations with other combinations of logical operators (`LOP`s), including `XOR` and negated operators,

$$x_{[\mathbf{n}]} \approx \underset{l}{\texttt{LOP}_o} \left[\underset{n_k \in [\mathbf{n}]}{\texttt{LOP}_i} \left(f_{n_k l} \right) \right] \quad . \tag{1.5}$$

The entirety of models that can be described through equation 1.5 constitute Logical Factorisation Machines and will be discussed in Chapter 4.

Our treatment is accompanied by a variety of real-world examples that demonstrate the wide range of applicability and the scalability of the proposed algorithms. As such, the examples in Sections 2.5 and 3.4.4 feature many billions of data points and can easily run on

today's commodity hardware. An application to collaborative filtering is given in Section 2.4. Further applications to different types of genomics data can be found for somatic mutations in cancer in Section 2.7, for gene expression in multiple tissues types in Section 3.4.4, and for gene-expression patterns in cancer in Section 3.4.3. A spatio-temporal dataset is analysed in Section 3.4.2. Wider areas of application are introduced, such as community detection in Section 2.8, extended to temporal networks in Section 3.4.1. An application to continuous data is discussed in Section 3.4.4. All these examples highlight the clear meaning and interpretability of inferred factors, which also enables the integration of expert domain knowledge and other sources of information into factor hierarchies, as demonstrated for the mutations and pathways in various cancer types in Section 2.7. Chapter 5 provides concluding remarks and directions for future work.

An important objective of this work is to enable practitioners to make use of the models that we developed. To this end, we have devised an object-oriented framework with a simple Python API that implements all models discussed throughout this thesis. It emphasises scalability and usability and is described alongside the relevant applications in sections 2.6, 2.7, and 4.10. Scalability is achieved through implementing all sampling routines such that they can be compiled to native machine instructions and exploit parallelism wherever possible. This is done using the Cython and Numba packages[1]. The code is freely available at `https://github.com/TammoR/LogicalFactorisationMachines` .

[1] `http://cython.org/`, `https://numba.pydata.org/`

Chapter 2

Bayesian Boolean Matrix Factorisation

In this chapter, we introduce the *OrMachine* (OrM), a probabilistic approach to Boolean matrix factorisation (BooMF), and describe how to fit the model using a fast and scalable Metropolised Gibbs sampling algorithm. Following a review of the related literature in Section 2.1, we describe the model and inference procedure in Section 2.2. On simulated and real-world data, our method is shown to significantly outperform the current state-of-the-art message passing approaches for learning BooMF models, as shown in Section 2.3 and 2.4. Section 2.6 demonstrates how these methods can be applied using our implementation. Next, we consider a challenging application in the analysis of high-throughput single cell genomics data in Section 2.5. OrM identifies latent gene signatures that correspond to key cellular pathways or biological processes from large gene expression datasets consisting of 1.3 million cells across 11 thousand genes. Genes are expressed if one or more relevant biological processes are active, a property which is naturally modelled by the Boolean OR operation. Next, Section 2.7 is concerned with the application of hierarchical layers of OrMachines for data integration, while Section 2.8 discusses its application to community detection in relational data. Eventually, Section 2.9 introduces the Indian Buffet Process as a prior distribution over factor matrices, performing posterior inference not only for the factor

matrices but also for the number of latent dimensions. We begin by briefly recapitulating the problem of Boolean Matrix Factorisation.

BooMF aims to decompose a binary matrix $X \in \{0,1\}^{N \times D}$ with N observations and D features into an approximate Boolean product of two low rank, binary factor matrices $Z \in \{0,1\}^{N \times L}$ and $U \in \{0,1\}^{D \times L}$. One of the factors contains meaningful patterns, the other quantifies how the observations can be expressed as a combination of these patterns. The data generating process is based on the Boolean product, a special case of matrix product between binary matrices where all values larger than zero are set to one, as stated in eq. (1.1).

We can think of BooMF as binary factor analysis or as clustering with joint assignments, where each observation is assigned to a subset of L cluster centroids or codes. The L-dimensional indicators provide a compact representation of which codes are allocated to each observation. A feature x_{nd} takes a value of one if it equals one in any of the assigned codes. This representation is illustrated for an example of images of calculator digits in Fig. 2.1. The observed data, $\mathbf{x}_n$, are 10 Arabic numerals `0,1,…,9` typed in their traditional representation in calculators. The data vectors are arranged to 17×10 pixelated images for interpretation. Using the probabilistic model that is explicated in the following sections, the data is factorised into two matrices of rank 6. This rank is not sufficient for full error-free reconstruction. All digits, except `7`, can be constructed by Boolean combination of the inferred codes, u, as indicated by the columns of the z. For constructing a `7`, however, we infer a posterior mean probability of ½ of using code l=5. Note, that there exist other equally valid solutions to this problem with 6 latent dimensions.

2.1 Related Work

BooMF has many recent real-world applications ranging from knowledge discovery (Erdos and Miettinen, 2013a) to collaborative filtering (Su and Khoshgoftaar, 2009) and computer vision (Lázaro-Gredilla et al., 2016). Its first applications dates back to biological data anal-

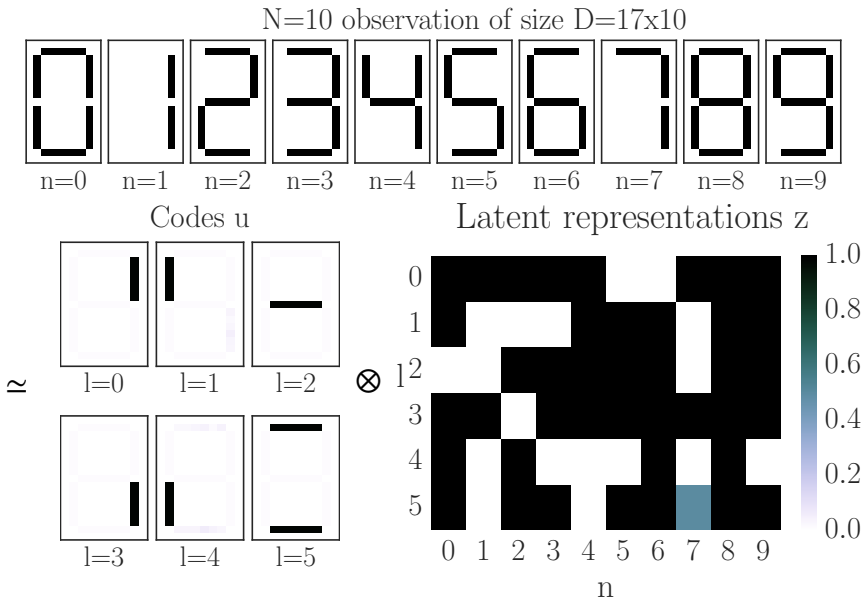


Fig. 2.1 **Calculator digits** - Toy example for Boolean Matrix Factorisation using images of digits as input data. The pixels in the factor matrices represent marginal posterior means. Data and codes are rearranged to 2D for interpretation (black: 1 (True), white: 0 (False)).

ysis (Nau et al., 1978). There has been a sustained interest in algorithms for BooMF. Miettinen et al. (2006) provide a greedy heuristic algorithm to solve BooMF without recourse to an underlying probabilistic model. It is based on association rule mining (Agrawal et al., 1994) and has more recently been extended to automatically select the optimal dimensionality of the latent space based on the minimum description length principle (Miettinen and Vreeken, 2014). There exist heuristic approaches rooted in formal concept analysis (Ganter and Wille, 2012), as for instance the work by Belohlavek and Trnecka (2015). In contrast, multi assignment clustering for Boolean data (Frank et al., 2012; Streich et al., 2009) builds on a probabilistic model that combines the noisy-OR and a global noise source. Point estimates are inferred by deterministic annealing. Similarly, Wood et al. (2012) develop a probabilistic model to infer hidden causes based on the noisy-OR. The authors use an Indian Buffet process prior over the latent space and a Gibbs sampler to infer the distribution over the unbounded number of hidden causes. A more expressive model for matrix factorisation with binary latent variables is introduced by Meeds et al. (2007), who combine binary inter-

actions with continuous weights. Frank et al. (2012) provide a detailed overview of BooMF and related methods.

An approach more similar to ours is the work by Ravanbakhsh et al. (2016). The authors tackle BooMF using a probabilistic generative model and derive a message passing algorithm to perform MAP inference. Their method is shown to have state-of-the-art performance for matrix factorisation and completion. It therefore serves us as baseline benchmark in these tasks. The message passing approach has recently been employed by Lázaro-Gredilla et al. (2016) in a hierarchical network combined with pooling layers to infer the building blocks of binary images.

2.2 OrMachine: Bayesian Boolean Matrix Factorisation

2.2.1 Model Formulation

The OrMachine is a probabilistic generative model for Boolean matrix factorisation. A matrix of N binary observations $\mathbf{x}_n \in \{0,1\}^D$ is generated from a discrete mixture of L binary codes $\mathbf{u}_l \in \{0,1\}^D$. Binary latent variables z_{nl} denote whether or not code l is used in generating a particular observation $\mathbf{x}_n$. The probability for a data point x_{nd} to be one is greater than ½ if the corresponding codes and latent variables in at least one latent dimension both equal one; conversely, if there exists no dimension where codes and latent variables both equal one, the probability for the data point to be one is less than ½. The exact magnitude of this probability is inferred from the data and, for later notational convenience, is parametrised as the logistic sigmoid of a global dispersion parameter, $\sigma(\lambda) = (1 + e^{-\lambda})^{-1}$, with $\lambda \in \mathbb{R}^+$. Note, that $\sigma(\lambda)$ corresponds to q in eq. (1.2) in Section 1.3. Alternatively, we could have a different noise magnitude for positive and negative predictions, similar to Ravanbakhsh et al. (2016). The same principles as in the following exposition would apply, but we leave

this generalisation for future work. The full joint distribution is given by

$$p(X, Z, U, \lambda) = p(X|Z, U, \lambda)\, p(Z)\, p(U)\, p(\lambda)\,, \tag{2.1}$$

where the likelihood factorises over features and observations with each factor given by

$$p(x_{nd}|\mathbf{u}, \mathbf{z}, \lambda) = \sigma\left[\lambda \tilde{x}_{nd}\left(1 - 2\prod_l (1 - z_{nl}u_{dl})\right)\right] = \begin{cases} \sigma(\lambda); & \text{if } x = \min(1, \mathbf{u}_d^T \mathbf{z}_n) \\ 1-\sigma(\lambda); & \text{if } x \neq \min(1, \mathbf{u}_d^T \mathbf{z}_n) \end{cases} \tag{2.2}$$

Here, tilde denotes the $\{0, 1\} \to \{-1, 1\}$ mapping, such that for any binary variable $x \in \{0, 1\}$, we have $\tilde{x} = 2x - 1$. This is particularly convenient thanks to the property of the logistic sigmoid, $\sigma(-x) = 1 - \sigma(x)$.

The expression inside the parentheses of eq. (2.2) encodes the OR operation and evaluates to 1 if $z_{nl}{=}u_{ld}{=}1$ for at least one l, and to -1 otherwise. The dispersion parameter controls the noise in the generative process, i.e. as $\lambda \to \infty$, all probabilities tend to 0 or 1 and the model describes a deterministic Boolean matrix product. Note that the likelihood can be computed efficiently from Eq. (2.2) as we describe in detail in the next section. We merely need to find a single latent dimension l where $z_{nl} = u_{ld} = 1$, and the product over the latent space collapses to 0, regardless of the remaining latent dimensions.

For now, we assume independent Bernoulli prior distributions for all variables u_{ld} and z_{nl}. Such prior distributions allow us to promote denseness or sparsity in codes and latent variables. We will discuss an infinite-dimensional extension in Section 2.9. Notice that the designation of U as codes and Z as latent variables is arbitrary since these matrices appear in a symmetric manner. If we transpose the matrix of observations X, then codes and the latent variables merely swap roles. More specifically, our model assumes no difference in sampling from a population of objects across a range of features as compared to sampling

from a population of features across a range of objects. Often this interpretation follows from the data collection, e.g. for sampling from a population of cells across a range of genes.

Finally, we place a (truncated) Beta-distribution as prior over the dispersion, $\sigma(\lambda)$, and maximise it using an Expectation Maximisation (EM) type algorithm described in the following section.

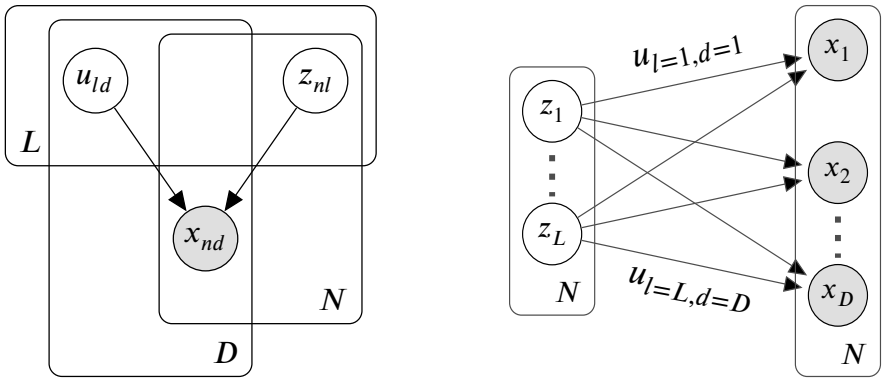
2.2.2 Fast Posterior Inference for Factor Matrices

The full conditional for an entry of the factor matrix, z_{nl}, (and analogous for u_{ld}) takes a remarkably simple form that lends itself to efficient computation. This form and its algorithmic implementation constitute a main result of this chapter and states that the distribution of an entry in the factor matrix, z_{nl}, conditioned on the state of all other variables and the data is given by

$$\boxed{p(z_{nl}|\cdot) = \sigma\left[\lambda\tilde{z}_{nl}\sum_d \tilde{x}_{nd}u_{ld}\prod_{l'\neq l}(1-z_{nl'}u_{l'd}) + \mathrm{logit}(p(z_{nl}))\right].} \tag{2.3}$$

A formal derivation is given in Appendix A in context of the more general problem of polyadic decompositions, treated in Chapter 3. Notice, that the independent Bernoulli prior enters the expression as an additive term inside the sigmoid function that vanishes for the uninformative Bernoulli prior $p(z) = ½$.

The form of eq. (2.3) allows for computationally efficient evaluation of the conditionals. To understand this in more detail it is useful to think of the representation as directed graphical model, as shown in Fig. 2.2. Computing the full conditional distribution for a hidden variable in a directed model requires evaluation of the variable's Markov blanket, that is its children and its children's parents. The latter dependency is occasionally referred to as *explaining away*, a term that was coined by (Pearl, 1988, Chapter 2) who discusses it in the context of causality: Two otherwise independent causes become correlated, conditional



(a) Graphical model with all dimensions collapsed into plates.

(b) Graphical model unrolled across the latent and the feature dimensions, L and D.

Fig. 2.2 Dependencies in the Boolean Matrix Factorisation model, represented as directed graphical model in plate notation. Observed nodes are filled in grey.

on their common effect. In the graphical model this occurs as V-structure, where the bottom node is observed as in Fig. 2.2a. The corresponding intuition is that observation of one cause makes the occurrence of another cause less likely. To give an example, adopted from Pearl (1988), imagine you observe an *alarm* at your house. Let's assume it may have two possible causes, a *burglary* and *earthquake*. Once you observe that there is indeed an earthquake, this observation *explains away* the alarm and reduces the probability of a burglary.

BooMF exhibits two V-structures. The previously discussed dependency between u_{ld} and z_{nl}, originates from the Boolean conjunction, that requires both variables to be one in order to to explain their observed child-node. This becomes apparent in Fig. 2.2b, where we unroll the model in dimensions L and D, treating the u_{ld} as parameters, indicated by arrows. For any $u_{ld} = 0$, the arrow disappears and the corresponding x_{nd} and z_{nl} become independent. Fig. 2.2b also shows the second V-structure, the dependency between any pair $(z_{n,l}, z_{n,l'})$, where $l' \neq l$. This dependency originates from the Boolean disjunction, where any latent dimension with $z_{nl} = u_{ld} = 1$ fully explains the child-node x_{nd}, and makes the child conditionally independent of all other parents. Thus, once one of these two conditions for independence is met, the contribution to the full conditional from a variable x_{nd} is known

without considering the remainder of it Markov blanket. In practice, when computing updates for z_{nl}, terms in the sum over d in eq. (2.3) necessarily evaluate to zero if one of the following conditions is met.

1. $u_{ld} = 0$.

2. $z_{nl'} u_{l'd} = 1$ for some $l' \neq l$.

We see that eq. (2.3) simply counts the correctly explained observations and subtracts the incorrectly explained observations for a choice of z_{nl}, after accounting for explaining-away dependencies. This leads to Algorithm 1 for fast evaluation of the conditionals. This algorithms scales linearly in the number of latent dimensions and the number of observations. In contrast, a naive implementation would scale quadratically in the number of latent dimensions.

Algorithm 1 Computation of the full conditional of z_{nl}

```
accumulator = 0
for d in 1, ..., D do
  if u_ld = 0 then
    continue (next iteration over d)
  end if
  for l' in 1, ..., L do
    if l' ≠ l and z_nl' = 1 and u_l'd = 1 then
      continue (next iteration over d)
    end if
  end for
  accumulator = accumulator + x̃_nd
end for
p(z_nl | ·) = σ(λ · z̃_nl · accumulator)
```

To infer the posterior distribution over all variables u_{ld} and z_{nl} we could iteratively sample from the above conditionals using standard Gibbs sampling. In practice we use a modification of this procedure which is referred to as Metropolised Gibbs sampler and is described by Liu (1996). We always propose to flip the current state, such that $z' \neq z$. This leads to a

Hastings acceptance probability of

$$q_{\text{Hastings}} = \frac{p(z'|\cdot)\, q(z|z')}{p(z|\cdot)\, q(z'|z)} = \frac{p(z'|\cdot)}{1 - p(z'|\cdot)} \geq p(z'|.) \tag{2.4}$$

This yields lower variance Monte Carlo estimates (Peskun, 1973). The metropolised Gibbs sampler can be illustrated by the example of draws from a fair coin with a Bernoulli probability 0.5. Proposing to change the current state at every step of the Markov chain and accepting with probability 1 amounts to alternation between heads and tails at every flip.

We investigated a random scan Gibbs sampler as proposed by Geman and Geman (1984), where, for each iteration, one scans through all variables in uniformly random order. We found that the scan order does not influence the quality of the samples measured by posterior predictive accuracy and therefore resort to sampling all variables in consecutive order. This approach is computationally more efficient and easier to implement.

2.2.3 MAP Inference for the Dispersion Parameter

After every sweep through the factor entries, the dispersion parameter λ is set to its maximum a posterior estimate, akin to the M-step of a Monte Carlo EM algorithm. This is available in closed form and is the subject of this section.

Given the current values of the factor matrices, U and Z, we can compute how many observations x_{nd} are correctly predicted by the deterministic model, as

$$T = \sum_{n,d} \mathrm{I}\left[x_{nd} = 1 - \prod_{l} \left(1 - z_{nl} u_{ld}\right)\right] \ . \tag{2.5}$$

Conversely, the number of incorrect prediction is $F = N \cdot D - T$. This allows us to rewrite the likelihood as

$$\mathscr{L} = \sigma(\lambda)^T \sigma(-\lambda)^F \ . \tag{2.6}$$

It follows that maximum likelihood estimate of λ is the logit of the fraction of correctly predicted entries. Instead of specifying a prior over λ directly, we consider a change of variables to $\sigma(\lambda)$, which directly corresponds to the Bernoulli parameter in the likelihood. As a prior for $\sigma(\lambda)$, it is natural to employ a truncated Beta-distribution,

$$p(\sigma(\lambda)) \propto \text{Beta}(\sigma(\lambda)|\alpha, \beta) \; ; \quad \sigma(\lambda) \in [0.5, 1]. \tag{2.7}$$

We are interested in MAP estimation, and thus do not need to deal with the inconvenient normalisation constant of the truncated Beta distribution. We explicate in Appendix C, that this prior affects the maximum likelihood estimate of λ equivalent to additional data: α correctly predicted data points and β additional incorrectly predicted data points.

$$\lambda_{\text{MAP}} = \text{logit}\left[\frac{\alpha + T}{\alpha + \beta + N \cdot D}\right] \tag{2.8}$$

Interestingly, a uniform prior over the interval $[0.5, 1]$, equivalent to a truncated beta-prior with $\alpha = \beta = 1$, corresponds to applying Laplace's rule of succession to the maximum likelihood estimate. If not specified otherwise, this is our choice of hyperparameters in the following experiments. We further see that the prior contribution becomes negligible in situations where the data is large. Appendix C shows that the variance of this estimate decreases approximately quadratically in the number of datapoints. Thus, it is well justified to neglect posterior uncertainty of λ in practice.

The alternation between sampling (U, Z) and updating the dispersion parameter is carried out until convergence; see Algorithm 2 for all steps of this procedure. The frequency of updates for λ, proposed after every sweep through all factor entries, is somewhat arbitrary. It can be treated as a hyperparameter but our experiments have shown no systematic benefits of diverging from the proposed default. In rare cases model performance, as measured by posterior predictive accuracy, can be improved by injecting additional noise in the sampler

during the burn-in phase. This can practically be achieved by keeping λ fixed to a small value. This stragegy, however, is not used in any of the examples presented here.

Algorithm 2 Sampling from the OrMachine

for i in $1, \ldots,$ max-iters **do**
  for n in $1, \ldots,$ N *(in parallel)* **do**
    for l in $1, \ldots,$ L **do**
      Compute $p(z_{nl}|\cdot)$ following Algorithm 1
      Flip z_{nl} with probability $[p(z_{nl}|\cdot)^{-1}-1]^{-1}$
    end for
  end for
  for d in $1, \ldots,$ d *(in parallel)* **do**
    for l in $1, \ldots,$ L **do**
      Compute $p(u_{ld}|\cdot)$ following Algorithm 1
      Flip u_{ld} with probability $[p(u_{ld}|\cdot)^{-1}-1]^{-1}$
    end for
  end for
  Set λ to its MAP estimate according to eq. (2.8).
end for

2.2.4 Missing Data and Imputation

We can handle unobserved data by marginalising the likelihood over the missing observations. More precisely, if $X = (X_{\text{obs}}, X_{\text{mis}})$ is the decomposition of the full matrix into the observed part X_{obs} and the missing part X_{mis}, after marginalisation, the initial likelihood $p(X|U, Z, \lambda)$ simplifies to $p(X_{obs}|U, Z, \lambda)$. Then, a naïve implementation could be based on indexing the observed components inside matrix X and modifying the inference procedure so that the posterior conditionals of z_{nl} and u_{ld} involve only sums over observed elements. A simpler, equivalent implementation, which we follow in our experiments, is to represent the data as $\tilde{x}_{nd} \in \{-1, 0, 1\}$ where missing observations are encoded as zeros, each contributing the constant factor $\sigma(0) = \frac{1}{2}$ to the full likelihood, so that $p(X|U, Z, \lambda) = C\, p(X_{obs}|U, Z, \lambda)$, where C is a constant. Thus, the missing values do not contribute to the posterior over U and Z which is also clear from the form of the full conditionals in eq. (2.3) that depend on

a sum weighted by x_{nd}s. For the update of the dispersion parameter in eq. (2.8), we need to subtract the number of all missing observations in the denominator, such that the maximum likelihood dispersion indicates the fraction of correct predictions in the observed data.

Following this inference procedure, we can impute missing data based on a Monte Carlo estimate of the predictive distribution of some unobserved x_{nd} as

$$\frac{1}{S}\sum_{s=1}^{S} p(x_{nd}|U^{(s)}, Z^{(s)}, \hat{\lambda}) \,, \tag{2.9}$$

where each $(U^{(s)}, Z^{(s)})$ is a posterior sample. A much faster approximation of the predictive distribution is obtained by $p(x_{nd}|\hat{U}, \hat{Z}, \hat{\lambda})$, where we simply plug the marginal posterior mean estimates for (U, Z) into the predictive distribution. For the simulated data in Section 2.3.2, we find both methods to perform equally well and therefore follow the second, faster approach for all remaining experiments in this Chapter. A more in-depth comparison of different reconstruction methods is given in the context of Tensor factorisation in Section 3.2.

2.2.5 Generalisation to Multiple Layers

BooMF learns patterns of correlation in the data. In analogy to generative multi-layer neural networks, we can build a hierarchy of correlations by applying another layer of factorisation to the factor matrix Z. This is reminiscent of the ideas of multilayer networks of stochastic units as for instance proposed by Hinton et al. (1998) and Hinton et al. (2006) and of the more recent work on deep exponential families (Ranganath et al., 2015). The ability to learn features at different levels of abstraction is commonly cited as an explanation for the success that deep neural networks have across many domains of application (Bengio et al., 2013; Lin and Tegmark, 2016). In the present setting, with stochasticity at every step of the generative

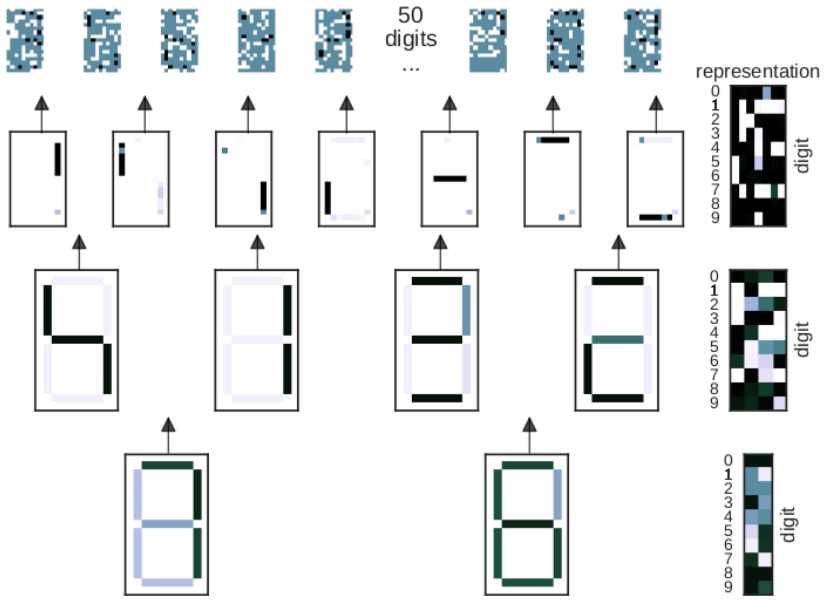


Fig. 2.3 An OrMachine with 3 hidden layers is trained to reconstruct 50 calculator digits with 70% of observations missing. The rows depict increasingly abstract layers of the model. Shown are the latent prototypes fed forward to the data layer. Variables are arranged to 17×10 images for interpretation. The right sides show the corresponding posterior means for representations of the partially observed input digits.

process and posterior inference, we are able to infer meaningful and interpretable hierarchies of abstraction.

To give an example, we determine the optimal multi-layer architecture for representing the calculator digit toy dataset as introduced in Fig. 2.1. We observe 50 digits and consider 70% of the data points randomly as unobserved. We then train multi-layer OrMachines with various depths and layer widths, cycling through the individual layers. More specifically we perform one iteration of Algorithm 3 for every layer and repeat the procedure for 200 iterations, discarding the first 100 as burn-in. We then draw 200 samples from each consecutive layer with the remaining layers held fixed to their marginal MAP estimate. This estimate is based on the 100 retained samples of the initial training phase. In order to enforce distributed representations, we choose independent Bernoulli sparsity priors for the codes: $p(u_{ld}) = [0.01, 0.05, 0.2]$ for each layer, respectively. Superior performance in reconstructing the unobserved data is achieved by a 3-hidden layer architecture with hidden

layers of size $L_1 = 7$, $L_2 = 4$, $L_3 = 2$. This 3-layer model reduces the reconstruction error from 1.4% to 0.4% compared to the single-layer model with width $L = 7$. MAP estimates of the dispersion for the three layers are $\hat{\lambda} = [0.99, 0.93, 0.8]$. The first layer infers the seven bars that compose all digits, the following layer infers dominant groupings of these bars, and so on. In Fig. 2.3, we plot the probabilities that each prototype induces in the observation layer. They are given by the means of the posterior predictive distribution as described in the previous section, conditioned on the one-hot activations of $\mathbf{z}_n$ with $z_{nl} \in \{0, 1\}$ and $\sum_l z_{nl} = 1$. Alongside, we depict the average posterior mean of the corresponding representations for each digit in the training data. This example illustrates that the multi-layer OrM infers interpretable higher-order correlations and is able to exploit them to achieve significant improvements in missing data imputation.

2.3 Experiments on Simulated Data

In this section, we probe the performance of OrM at random matrix factorisation and completion tasks. Message passing (MP) has been shown to compare favourably with other state-of-the-art methods for BooMF that we introduced in Section 2.1 (Ravanbakhsh et al., 2016). It therefore is the focus of our comparison. The following settings for MP and the OrM are used throughout our experiments, unless mentioned otherwise. For MP, we use the Python implementation provided by the authors[1]. We also proceed with the authors' choice of hyperparameters, as experimentation with different learning rates and maximum number of iterations did not lead to any improvements. For both methods, we set the priors $p(u)$ and $p(z)$ to the factor matrices' expected value based on the average density of the product matrix in an Empirical-Bayes fashion. The only exception is MP in the matrix completion task, where uniform priors, as used by Ravanbakhsh et al. (2016), lead to slightly better performance. For the OrM, we initialise the parameters uniformly at random and draw 100

[1] https://github.com/mravanba/BooleanFactorization, retrieved on May 23, 2018

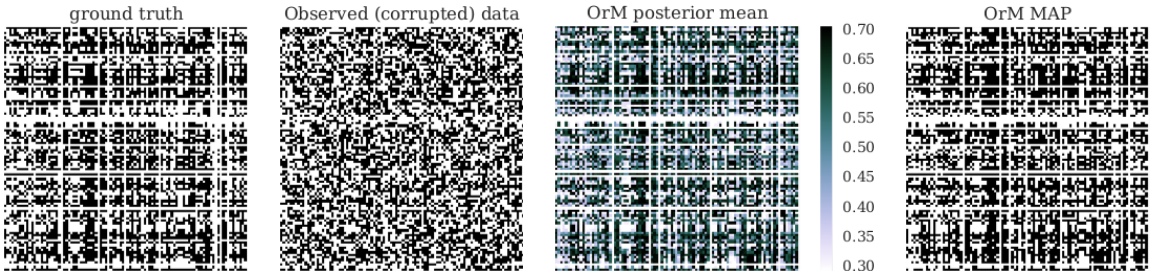


Fig. 2.4 Illustration of the matrix factorisation task for a 100×100 matrix of rank 7. The posterior means estimate the probability of each data point to take a value of one. MAP estimates are computed by rounding to the closest integer.

iterations after 100 samples of burn-in. Note that around 10–50 sampling steps are usually sufficient for convergence.

2.3.1 Random Matrix Factorisation

We generate a quadratic matrix $X \in \{0,1\}^{N \times N}$ of rank L by taking the Boolean product of two random $N \times L$ factor matrices. The Boolean product X of two rank L binary matrices that are sampled i.i.d. from a Bernoulli distribution with parameter p has an average density of $\rho(X) = 1 - (1 - p^2)^L$. Since we generally prefer X to be neither sparse nor dense, we fix its expected average density to ½, unless stated otherwise. This ensures that a simple bias toward zeroes or ones in either method is not met with reward.

Bits in the data are flipped at random with probabilities ranging from 5% to 50%. Factor matrices of the correct underlying dimension are inferred and the data is reconstructed from the inferred factorisation. An example of the task is shown in Fig. 2.4. Results for the reconstruction error, defined as the fraction of misclassified data points, are depicted in Fig. 2.5. All experiments were repeated 10 times with error bars denoting standard deviations.

The OrM outperforms MP under all conditions, except when both methods infer equally error-free reconstructions. Fig. 2.5 (top) reproduces the experimental settings of Fig. 2 in Ravanbakhsh et al. (2016). We find that the OrMachine enables virtually perfect reconstruction of a 1000×1000 matrix of rank $L = 5$ for up to 35% bit flip probability. Notably, MP

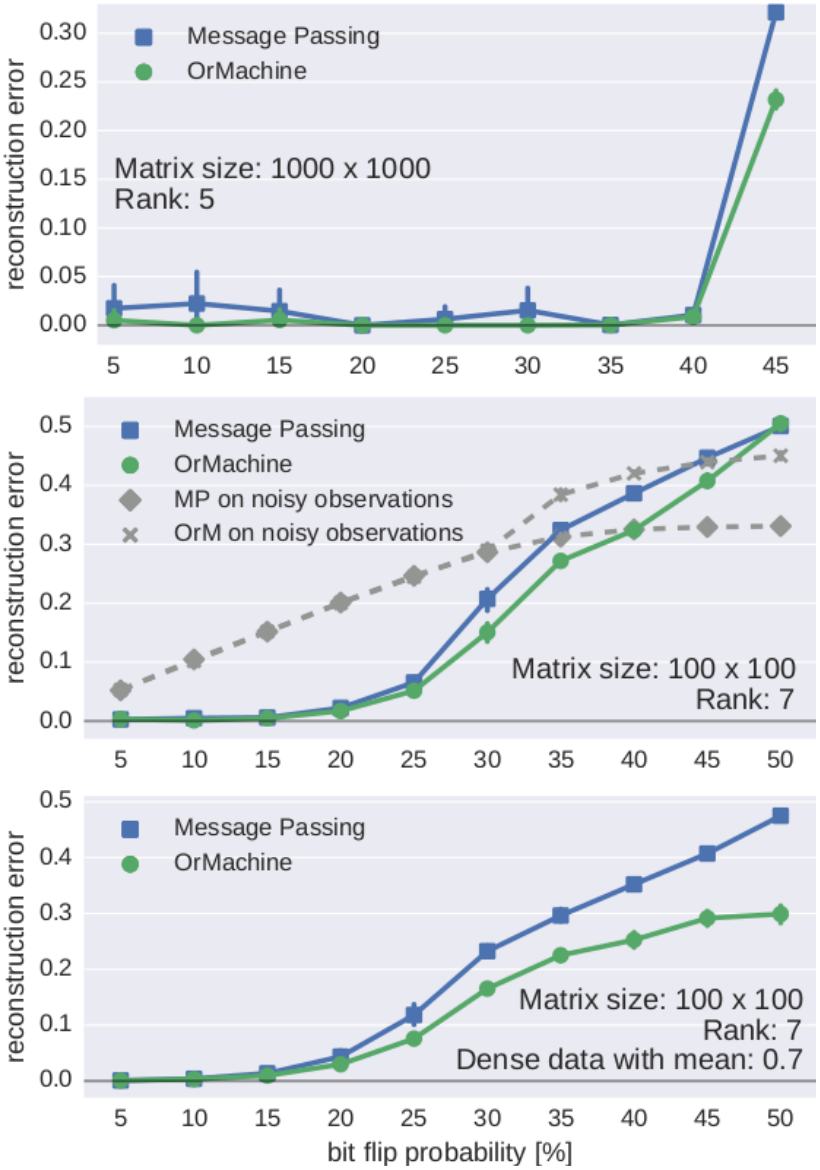


Fig. 2.5 Comparison of OrMachine and message passing for BooMF for random matrices of different size, rank and density. Compare to Fig. 2 in Ravanbakhsh et al. (2016). Grey lines in the middle figure indicate the reconstruction error with respect to the noisy training data and indicate that MP overfits to the training data in the regime of large noise. See text for further discussion.

performs worse for smaller noise levels. It was hypothesised by Ravanbakhsh et al. (2016) that symmetry breaking at higher noise levels helps message passage to converge to a better solution. Fig. 2.5 (middle) demonstrates the consistently improved performance of the OrMachine for a more challenging example of 100×100 matrices of rank 7. The reconstruction performance of both methods is similar for lower noise levels, while the OrMachine consistently outperforms MP for larger noise levels. For biased data with $\rho(x) = 0.7$ in Fig. 2.5 (bottom), we observe a similar pattern with a larger performance gap for higher noise levels. Even for a bit flip-probability of 50% the OrMachine retains a reconstruction error of approximately 30%, which is achieved by leveraging the bias in the data. Fig. 2.5 (middle) also shows the reconstruction error on the observed data, indicating that MP overfits the data more than the OrM for larger noise levels. This may contribute to the improved performance of the OrMachine. A more detailed investigation of this hypothesis is discussed in the context of tensor factorisation in Section 3.3.

2.3.2 Random Matrix Completion

We further investigate the problem of matrix completion or collaborative filtering, where bits of the data matrix are unobserved and reconstructed from the inferred factor matrices. Following the procedure outlined in Section 2.3.1, we generate random matrices of rank 5 and size 250×250. We only observe a random subset of the data, ranging from 0.5% and 3.5%. The missing data is reconstructed from the inferred factor matrices. As shown in Fig. 2.6, the OrMachine outperforms message passing throughout. The plot indicates means and standard deviations from 10 repetitions of each experiment.

Notably, the OrMachine does not only provide a MAP estimate, but also an estimate of the posterior probability for each unobserved data point x_{nd}. Fig. 2.6 (bottom) shows an estimate of the density of the posterior means for the correctly and incorrectly completed data points. The distribution of incorrect predictions peaks around a probability of ½, indicating

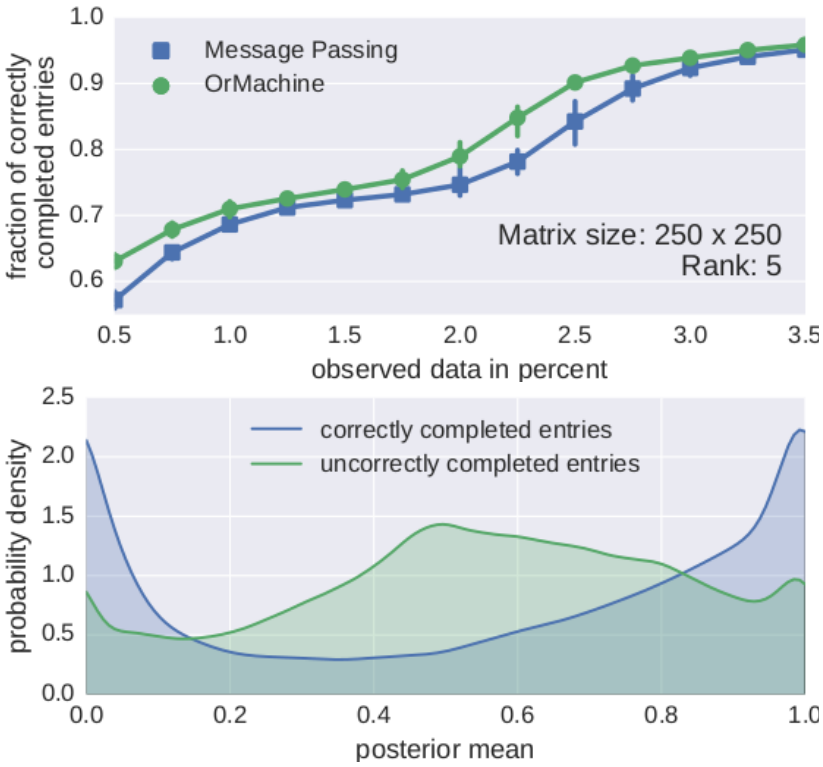


Fig. 2.6 Matrix completion performance for simulated low rank matrices (top) and kernel density estimate of the distribution of posterior means for inferred matrix entries (bottom).

that the OrMachine's uncertainty about its reconstruction provides further useful information about the missing data. For instance, this information can be used to control for false positives or false negatives, simply by setting a threshold for the posterior mean.

2.4 Real-World Benchmark: MovieLens

We investigate the OrMachine's performance for collaborative filtering on a real-world dataset. The MovieLens-1M dataset[2] contains 10^6 integer film ratings from 1 to 5 from 6000 users for 4000 films, such that 1/24 of the possible ratings are available. Similarly, the MovieLens 100k dataset contains 943 users and 1682 films. Following Ravanbakhsh et al. (2016), we binarise the data taking the global mean as threshold. We observe only a fraction of the

[2]The MovieLens dataset is available at: https://grouplens.org/datasets/movielens/. Retrieved January 2017.

Table 2.1 Collaborative filtering performance for MovieLens 1M and 100k dataset. Given are the percentages of correctly reconstructed unobserved data as means from 10 random repetitions. Compare to Table 1 in Ravanbakhsh et al. (2016), who also provide comparison to other state-of the art methods. Their results for message passing were independently reproduced. The multi-layer OrMachine has two hidden layers of size 4 and 2, respectively.

	observed percent. of available ratings					
	1%	5%	10%	20%	50%	95%
100K						
OrM	**58.5**	**63.5**	64.9	66.4	**68.9**	70.0
MP	52.8	60.7	63.0	65.2	67.5	69.5
Multi layer OrM	**58.5**	**63.5**	**65.2**	**66.5**	68.8	**70.1**
1M						
OrM	63.4	67.0	68.5	69.8	70.9	71.2
MP	56.7	64.9	67.2	68.8	70.7	71.5
Multi layer OrM	**63.8**	**67.2**	**68.6**	**70.0**	**71.4**	**72.1**

available data, varying from 1% to 95%, and reconstruct the remaining available data following the procedure in Section 2.3.2 with L=2 latent dimensions. Reconstruction accuracies are given as fractions of correctly reconstructed unobserved ratings in Table 2.1. The given values are means from 10 randomly initialised runs of each algorithm. The corresponding standard deviations are always smaller than 0.2%. The OrMachine is more accurate than message passing in all cases, except for the 1M dataset with 95% available ratings. The OrMachine's advantage is particularly significant if only little data is observed. Increasing the latent dimension L to values of 3 or 4 yields no consistent improvement, while a further increase is met with diminishing returns. We achieve the best within-sample performance for a two-layer OrMachine with different architectures performing best for different amounts of observed data. An OrM with two hidden layers of sizes 4 and 2 respectively yields the best average performance. As indicated in Table 2.1, it provides better results throughout but exceeds the performance of the shallow OrMachine rarely by more than 1%. This indi-

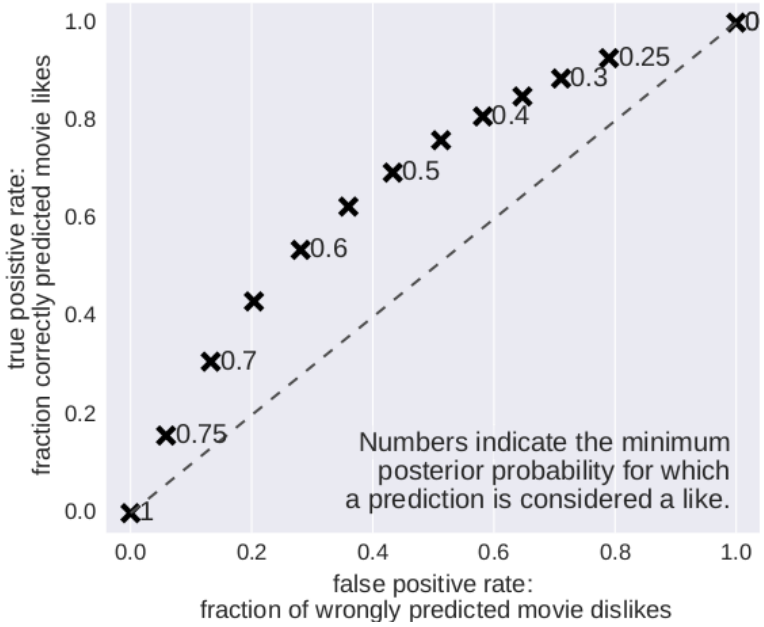


Fig. 2.7 ROC curve for MovieLens 100k data, adjusting the threshold for when a prediction is considered a like. 10% of the available data were observed and used for inference with an OrM of size $L = 2$. Predictions were tested on the remaining 90%.

cates that there is not much higher order structure in the data, which is unsurprising given the sparsity of the observations and the low dimensionality of the first hidden layer.

Inferring posterior distributions, we can choose a threshold for how likely we want a certain prediction to take a certain value and trade off false with true positives. A corresponding ROC curve for the MovieLens 100k dataset, where 10% of the available data was observed, is shown in Fig. 2.7.

2.5 Explorative Analysis at Scale: Single Cell Genomics

Single-cell RNA expression analysis is a revolutionary experimental technique that facilitates the measurement of gene expression on the level of a single cell (Blainey and Quake, 2014). In recent years this has led to the discovery of new cell types and to a better understanding of tissue heterogeneity (Trapnell, 2015). The latter is particularly relevant in cancer research where it helps to understand the cellular composition of a tumour and its

relationship to disease progression and treatment (Patel et al., 2014). Here we apply the OrMachine to binarised gene expression profiles of approximately 1.3 million cells for about 28 thousand genes per cell. The data was presented as an introductory example in Chapter 1. Cell specimens were obtained from cortex, hippocampus and subventricular zone of E18 (embryonic day 18) mice; the data is publicly available[3]. Only 7% of the data points are non-zero. We set all non-zero expression levels to one, retaining the essential information of whether or not a particular gene is expressed. We remove genes that are expressed in fewer than 1% of cells with roughly 11 thousand genes remaining. This leaves us with approximately 1.4×10^{10} data points. We apply the OrMachine for latent dimensions $L = 2, \dots, 10$. The algorithm converges to a posterior mode after 10–20 iterations, taking roughly an hour on a 4-core desktop computer and 10–30 minutes on a cluster with 24 cores. We draw 125 samples and discard the first 25 as burn-in.

Factorisations with different latent dimensionalities form hierarchies of representations. Features that appear together in representations for lower dimensions are progressively split apart when moving to a higher dimensional latent space. This bears similarity to hierarchical PCA model for data visualisation proposed by Bishop and Tipping (1998). We illustrate this approach to analysing the inferred factorisations on calculator digits in Fig. 2.8. Each row corresponds to an independently trained OrMachine with the dimensionality L increasing from 3 to 7. We observe denser patterns dividing up consecutively until only the seven constituent bars remain. This is a form of hierarchical clustering that, in contrast to traditional methods, does not impose any hierarchical structure on the model. Note, in the bottom row of Fig. 2.8, that the model exhibits an auto-regulating sparse behaviour. Seven bars are sufficient to model all digits and an additional, eighth dimension does not converge to any non-zero patterns. The latent allocations that correspond to the eighth dimension have a posterior mean of ½.

[3] https://support.10xgenomics.com. Retrieved December 2016.

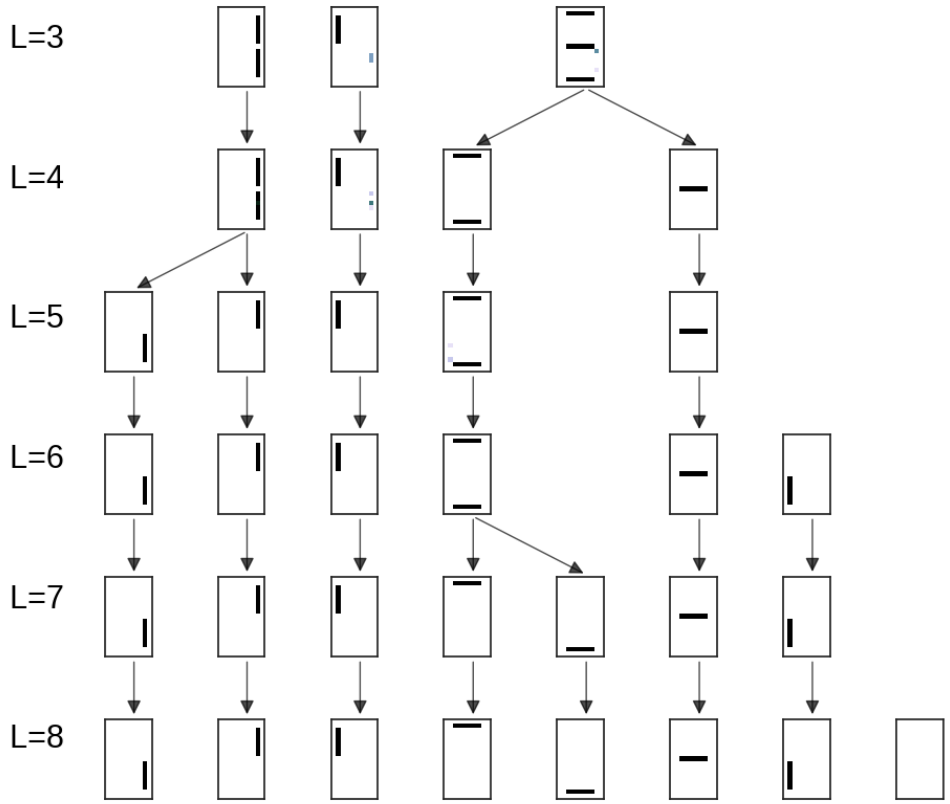


Fig. 2.8 Hierarchy in the latent features of calculator digits. The arrows indicate the split of codes into more distributed codes of lower density. The arrows are algorithmically inferred as latent variables in an OrMachine, with codes from the model of size L as data and codes from model of size $L+1$ as fixed codes.

We perform the same analysis on the single cell gene expression data with the results for both, gene patterns and specimen patterns shown in Fig. 2.9. This Figure should be interpreted in analogy to Fig. 2.8. Furthermore, we run a gene set enrichment analysis for the genes that are unique to each inferred code, looking for associated biological states. This is done using the Enrichr analysis tool (Chen et al., 2013) and a mouse gene atlas (Su et al., 2004). Biological states are denoted above each code, together with the logarithm to base 10 of their adjusted p-value. Increasing the latent dimensionality leads to a more distributed representation with subtler, biologically plausible patterns. The columns in Fig. 2.9 are ordered to emphasise the hierarchical structure. For instance, in the first column for $L = 5$ and second column for $L = 6$, a gene set with significant overlap to two biological processes

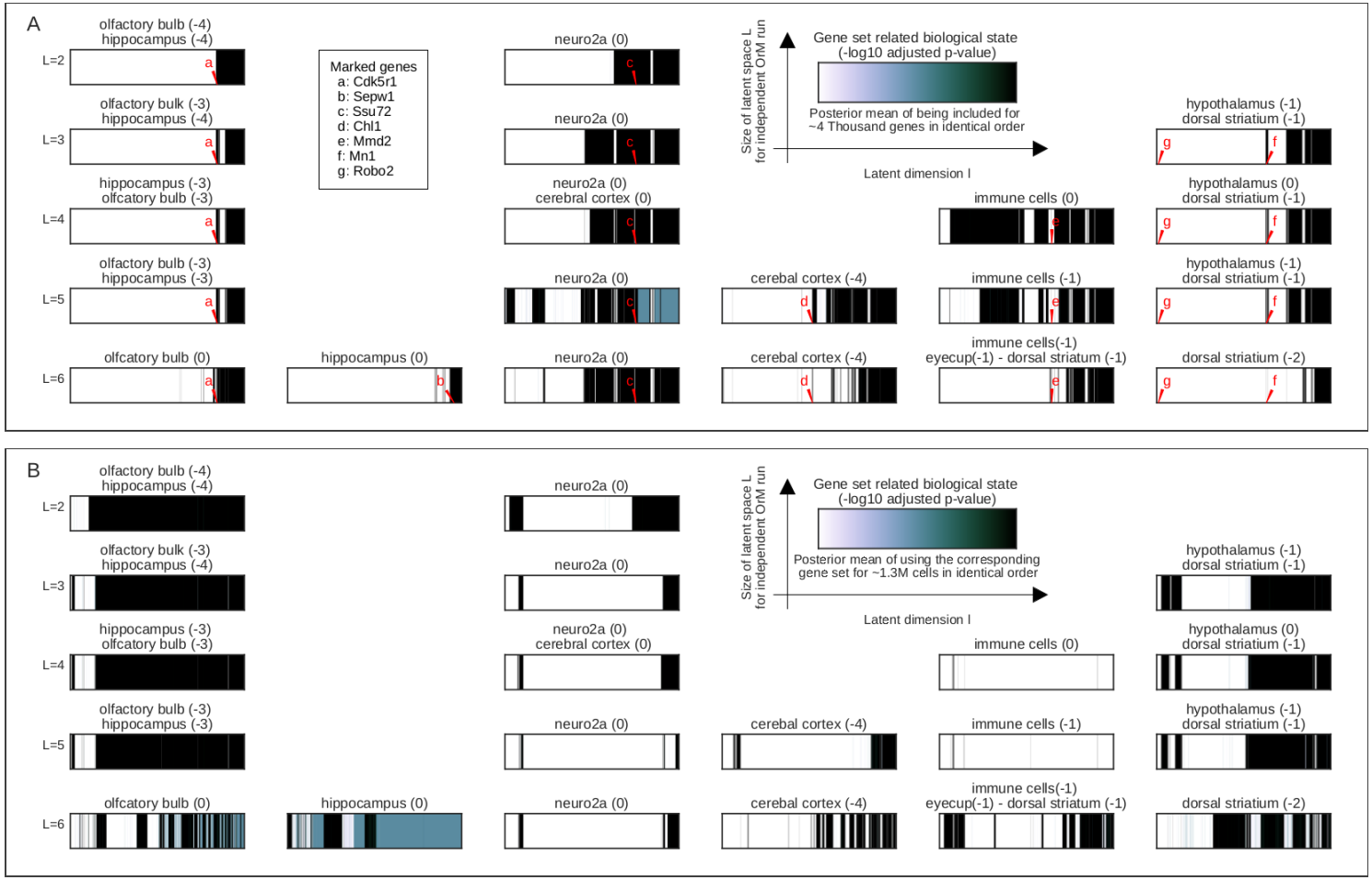


Fig. 2.9 Hierarchy in the latent representations of genes (A) and specimens (B) under variation of the latent dimensionality. Rows with the same dimensionality ($L = 2 \dots 6$) correspond to the same factorisation. Rows with different dimensionality are trained independently. The ordering of genes/specimens is identical throughout. Fig. (A) describes sets of expressed genes, Fig. (B) describe representations of cell specimens in terms of which of the gene sets they express. See legend in each box for more details.

(olfactory bulb and hippocampus) splits into two gene sets each corresponding to one of the two processes. In the assignment of cells to these sets (Fig. 2.9B), this is associated to an increase in posterior uncertainty as to which cell expresses this property. The significance levels of the associated biological processes drop with p-values on the order of 10^{-3} increasing to p-values on the order of 1. In addition, typical genes for each of the biological states are annotated (Demyanenko et al., 2010; Lopez-Bendito et al., 2007; Raman et al., 2013; Upadhya et al., 2011; Zheng et al., 2008). This example illustrates the OrMachine's ability to scale posterior inference to massive datasets. It enables the discovery of readily interpretable patterns, representations and hierarchies, all of which are biologically plausible.

2.6 Implementation of a Simple Factor Model

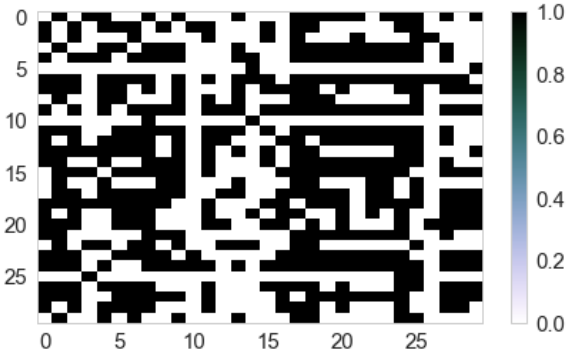
In this section we give a simple example of how our implementation can be used to do inference in the models that were discussed in the previous sections. More advanced usage will be demonstrated in Section 4.10.

We begin by generating synthetic data from Boolean product of two random matrices. Noise is added by randomly flipping every entry with 10% probability and we visually inspect the resulting matrix:

```
import numpy as np
import lfm  # import relevant modules
import lfm.auxiliarty_functions as aux

N = 200; D = 100; L = 4  # choose dimension of matrix and latent factors
Z = np.array(np.random.rand(N, L) > .7, dtype=np.int8)
U = np.array(np.random.rand(D, L) > .7, dtype=np.int8)
X = 2*np.array(np.dot(Z,U.transpose())>0, dtype=np.int8)-1
X = aux.add_bernoulli_noise_2d(X, p=0.1)

aux.plot_matrix(X)
```



Here `X` is a `numpy.ndarray` of 8bit integers and is coded as $\{-1, 1\}$ such that missing data would be represented as 0. We specify the model by instantiating the class lfm.Machine:

```
simple_model = lfm.Machine()
```

The `simple_model` object is a container for the entire model, including the inference method, data, factors and parameters. We attach a matrix, here the data, to our model using the `add_matrix` method:

```
data = simple_model.add_matrix(X, fixed=True)
```

In the current implementation X can have two or three dimensions for matrix decomposition or tensor decomposition, respectively. Like the factor matrices that we define next, the `data` object is an instance of `lfm.MachineMatrix`. Practically, the only difference between data and factors is that data is held fixed, whereas the factors are inferred. This is indicated through the `fixed` argument above.

In the next step, we define the model by adding a layer of factor matrices whose product will approximate the data.

```
layer = simple_model.add_layer(latent_size=3, child=data, model='OR-AND')
```

The `layer` object is an instance of `MachineLayer` and encapsulates factor matrices and parameters that are associated to the same data array. In this call, we specify the dimensionality of the latent space and pass the data object that is to be factorised as argument. Matrices and layers have mutual relationships akin to nodes in a graphical model. In our example, `data` is the child of `layer` and conversely, `layer` is the parent of `data`. These relationship are exposed as attributes. For example, the following identities are true:

```
layer.child == data
data.parents == layer
```

Importantly, any pair of logical operators, introduced in Chapter 4, can be passed as argument to `add_layer`, for example `model='XOR-AND'`, `model='NOR-XOR'`, etc. The model is now fully defined and we can run inference by calling:

```
simple_model.infer()
```

The `infer` method draws samples from the full conditional distribution of all matrix entries that are not explicitly fixed. After convergence is detected, a pre-specified number of samples, by default 50, is drawn and the sampling traces are saved. The output of the previous function call looks as follows:

```
Assigning sampling functions: OR_AND_2D
Assigning sampling functions: OR_AND_2D
Assigning update function: OR_AND_2D
burning in markov chain...
    iteration: 60
    converged at reconstr. accuracy: 0.9
allocating memory to save samples...
drawing samples...
    iteration 50; recon acc.: 0.9
finished.
```

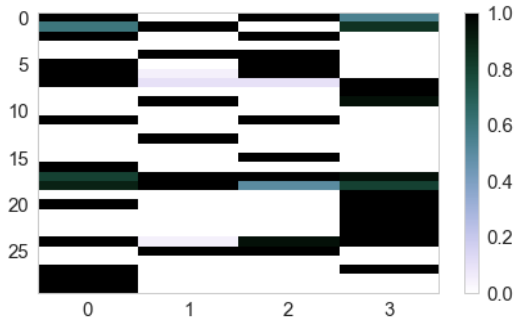
We explain this output in turn. Each factor and parameters exposes and inference method, that executes the sampling or update steps. The first three lines, indicate that the appropriate update functions are assigned, first to the two factor matrices, then to the noise parameter. The inference procedure consists of repeatedly calling these methods. This enables the simple extension of our framework to incorporate different models through manually assigning sampling functions to each matrix. For example, `my_sampling_fct` can be a python function that takes an instance of MachineMatrix as sole argument and updates its entries. To incorporate the modification, we overwrite the automatic assignment and simply execute

```
layer.z.sampling_fct = my_sampling_fct,
```

before calling the inference method. This modularity allows for code re-use. For instance, for updating the two factor matrices in Boolean matrix factorisation we can use the same function and merely require a small wrapper that transposes the data.

Convergence is detected when the average reconstruction accuracy does not improve over a given window of samples, here after 60 iterations. Subsequent samples are saved as traces, which are properties of the corresponding factor matrix and can, for example, be accessed as `layer.u.trace` or `layer.u.mean`. In particular, we can visually inspect the posterior sample mean.

```
aux.plot_matrix(layer.u.mean())
```



Moreover, the noise parameters, `lbda`, is attached to each layer and inherits from the same abstract `trace` methods as the factor matrices. Thus its posterior sample mean can be accessed by calling `layer.lbda.mean()`.

Finally, each layer object has an `output` method, that computes an estimate of the child's value, based on the full posterior sample or on the sample means of its parents.

```
aux.plot_matrix(layer.output())
```

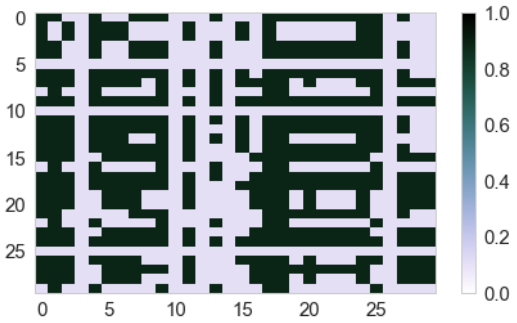


Fig. 2.10 gives a summary of the full example.

```
# Assume data array, X, is in scope
import lfm
simple_model = lfm.Machine()  # initialise model
data = simple_model.add_matrix(X, fixed=True)  # attach data

# attach parent layer with 'OR-AND' logic and 3 latent dimensions
layer = simple_model.add_layer(latent_size=3, child=data, model='OR-AND')

simple_model.infer()  # run inference

# visually inspect factors; .show() method is equivalent to
 ↪  aux.plot_matrix()
layer.z.show()  # z is alias for layer.factors[0]
layer.u.show()  # u is alias for layer.factors[1]
```

Fig. 2.10 Basic procedure for Boolean matrix factorisation

2.7 Hierarchies of OrMachines for Data Integration

In this section we demonstrate the composition of hierarchies of OrMachines to integrate multiple data sources. We have three sets of binary observations:

1. Mutation profiles for 8100 cancer patients across 2200 genes, indicating the presence of non-silent mutations. The data is from the Cancer Genome Atlas (TCGA) (Weinstein et al., 2013).

2. For each patient we have a one-hot vector indicating one out of 24 cancer types that they are suffering from, resulting in a 24 by 8100 binary matrix.

3. For each of the 2200 genes we indicate its membership in 66 cellular pathways that are known to be important in cancer[4].

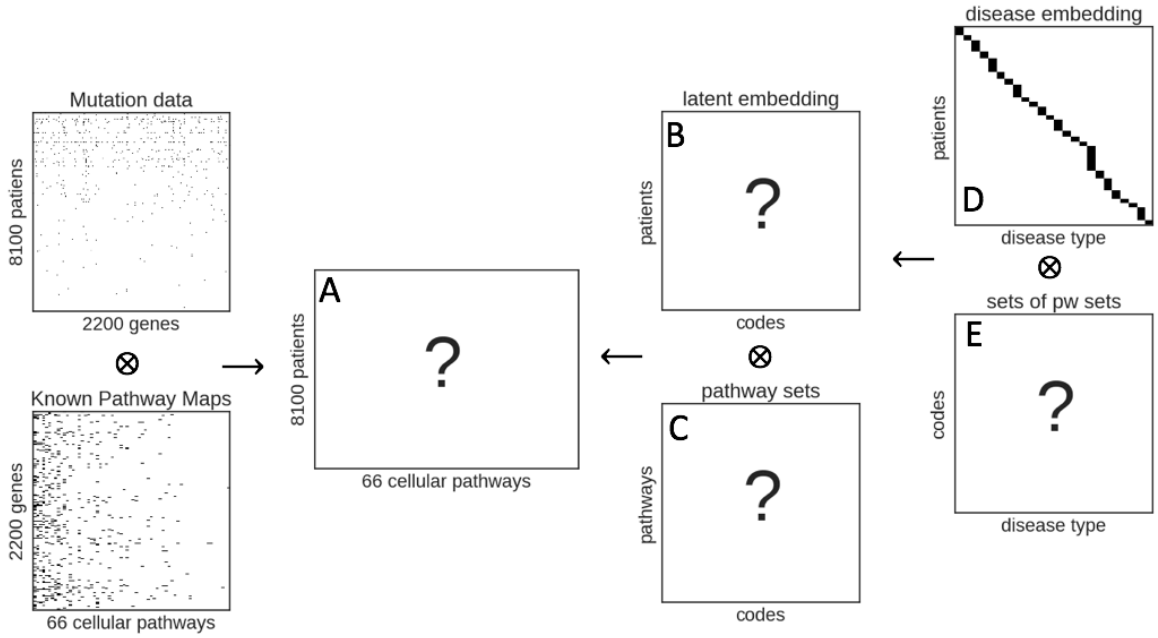


Fig. 2.11 Hierarchical encoding for integration of cancer mutation data. All *?*-labelled matrices are simultaneously inferred via Gibbs sampling. Alphabetical labels denote correspondence to the inferred posteriors means in Fig. 2.12. See Fig. 2.13 for how to implement this model hierarchy. Each arrow denotes a single OrMachine Factorisation.

[4]http://www.genome.jp/kegg/atlas.html. Retrieved in November 2016

The factor hierarchy is shown in Fig. 2.11, with all latent matrices labelled by question marks. The probabilistic Boolean product between mutation and pathway data is taken along the gene dimension and maps to a patient-pathway-matrix that indicates, for each patient, which pathways are likely to be affected. At the same time, this matrix is factorised into a low rank representation of 25 latent codes that indicate sets of co-occurring pathways. In a third step the patient embedding, as disjunction of pathway-sets, is factorised. The new higher level patient embedding is held fixed in order to represents each patient's known cancer type. We cycle through all latent matrices at each sampling step. Otherwise the inference proceeds as described in the previous examples. Results are shown in Fig. 2.12. The patient rows in (A) and (B) are sorted and labelled by primary disease type. We observe a strong disease type specificity, as encouraged by the hierarchical prior (D). This can be seen in the block patterns in (B). Many pathway sets are shared consistently among certain types of diseases, for example in latent dimension 2, WNT, PI3K, cell-cycle and E2F4-targets pathways are shared between lung (LUSC), esophageal (ESCA) and ovarian (OV) cancers. At the same time, however, there is distinct heterogeneity across the disease types and within each type of disease. This experiment exemplifies not only how hierarchical OrMachines can integrate data types, but how this can lead to interpretable representations and testable scientific hypotheses.

Our implementation that was introduced in Section 2.6 can handle arbitrary hierarchical relationships. We provide code to specify the hierarchical factor model for data integration in Fig. 2.13.

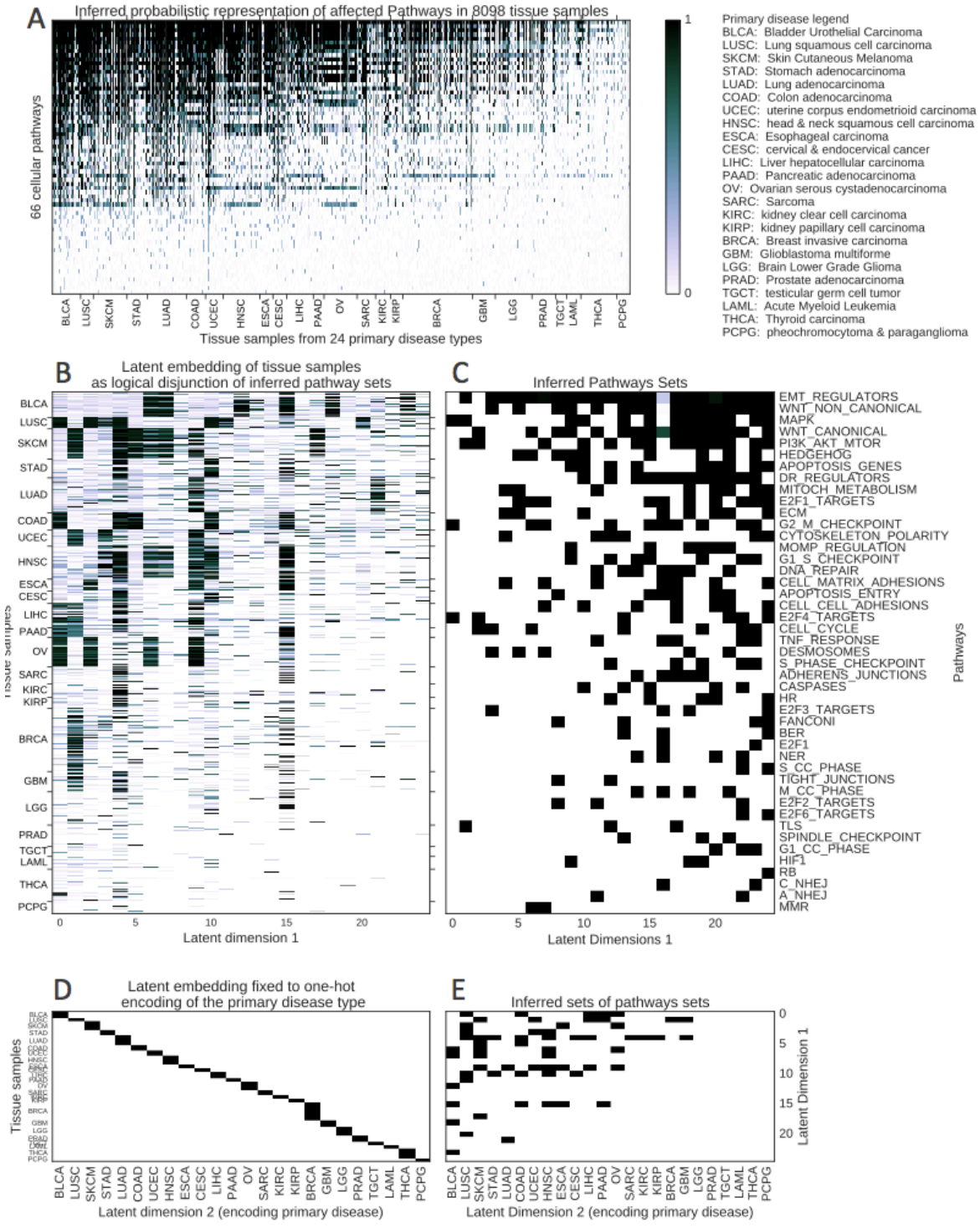


Fig. 2.12 Distributed representation of tumour samples in 4-layer architecture leveraging additional knowledge about pathways and the primary disease type.

```
import lfm
# Available data
# pw: gene-pathway matrix
# mt: patient-gene matrix indicating mutation status
# ds: patient-disease matrix

model = lfm.Machine()

# initialise patient-pathway matrix
A = model.add_matrix(shape=(mt.shape[0], pw.shape[1]))

# initialise first layer and fix its matrices
layer1 = model.add_layer(child = A)

layer1.z.val = mt
layer1.facotrs[0].fixed = True

layer1.u.val = ds
layer1.facotrs[1].fixed = True

# initialise first latent layer
layer2 = model.add_layer(child = A, latent_size = 25)

layer3 = model.add_layer(child = layer2.z)
layer3.z.val = ds

# inference
model.infer()
```

Fig. 2.13 Example of how to define complex hierarchical relationships. See Section 2.6 for more details on the implementation.

2.8 Community Detection for Network Analysis

Binary matrices are a natural way to represent relations between discrete objects. Such relations include networks and the OrMachine immediately lends itself to the analysis of directed and undirected network data. In this section, we discuss the OrM posterior in the context of community detection and point out its advantages over existing methods. We conclude with the analysis of a popular example, a social network of monks in a monastery (Sampson, 1968). Note, that a further examples for community detection is presented in the context of tensor factorisation in Section 3.4

Communities can be defined as groups of nodes that share certain properties. Community detection is the task of identifying these groups from unlabelled networks. Furthered by the enormous breadth of applications and, usually, the lack of ground truths, this problem receives continuing attention from various perspectives (for an overview see Fortunato and Hric, 2016). A variety of powerful approaches for detecting *overlapping communities* in networks is based around statistical inference in generative models, most prominently represented by the mixed-membership stochastic block model (MMB) introduced by Airoldi et al. (2008). Inference in such models is notoriously difficult and limited to small datasets or approximate inference schemes. Moreover, Yang and Leskovec (2012) have pointed out that typical modelling assumptions for overlapping communities are widely inappropriate. In particular, most existing methods make the implicit assumption that regions with community overlap are less densely connected than non-overlapping regions. To overcome this problem, the authors propose a model akin to the noisy-OR. Here, we propose probabilistic Boolean Matrix Factorisation with the OrMachine as a novel, simple and scalable approach that addresses both challenges.

The simple framework provides great flexibility: BooMF can deal with directed and undirected networks, with unobserved edges, with assortative and disassortative communities, and, by virtue of multi-layer BooMF, with community hierarchies. In contrast to

existing methods, each node can belong to an arbitrary number of zero or more communities.

2.8.1 Bottom-up Emergence of Communities

The generative process in most probabilistic models for community detection can be viewed as directed graphical model suggesting a causal process of data generation. In particular, these models describe the presence or absence of edges in a network as being generated from a latent community structure. Claims about its rationality and interpretability rest on the nature of this generative process. Here, we consider the inverse generative procedure and describe communities as emergent properties of the underlying network structure. For instance, in a protein interaction network, biological processes (communities) emerge from the interactions among proteins, rather than the reverse. Similarly, social communities emerge from the relationships of the people within them.

We now describe a simple generative process in this spirit. Communities are defined by latent properties, given by a set of nodes that the community's members are likely connected to (or receive connections from). Consequently, a node is more likely to belong to a community, if it shares many of the community's latent properties. Importantly, this only holds for properties that are not yet explained by affiliation with another community. Consider an example of a friendship network, visualised in Fig. 2.14. Suppose we are interested in finding evidence for Alice and Bob to be part of the same community. It turns out that they have a common friend, Charlie. This friendship is a shared property and hence provides evidence that Alice and Bob belong to the same community. Now assume that Alice and Charlie went to school together, whereas Bob grew up in a different city. The latter fact perfectly explains the connection between Alice and Charlie. Thus, Bob's friendship with Charlie does not provide evidence for Alice and Bob being members of the same community. This corresponds exactly to the *explaining-away* property across latent dimensions that we

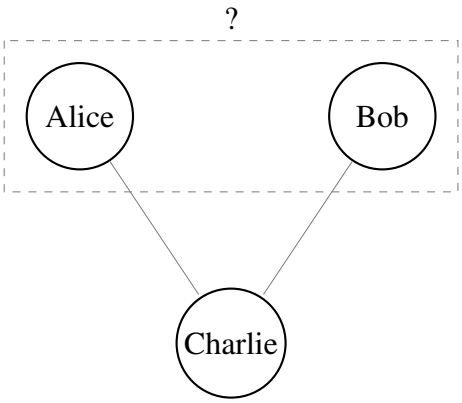


Fig. 2.14 Example friendship network

discussed in Section 2.2.2. In fact, the process that we just described corresponds exactly to the posterior distribution of the OrMachine as stated in eq. (2.3).

2.8.2 Application of the OrMachine to Community Detection

With this observation, we can readily apply the OrMachine to detect communities. In an undirected network the data matrix $X \in \{0,1\}^{N\times D}$ is simply the network adjacency matrix A. We can model directed networks by concatenating $X = [A, A^T]$, such that $\mathbf{x}_n$ is a vector that indicates incoming and outgoing edges. Then, the rows of U encode the defining properties of communities in terms of connections to other nodes in the network. Similarly, rows of Z encode the community association of node n.

Assortativity is the tendency of nodes to be connected to nodes with similar properties. We can model assortative networks by treating each node as connected to itself, that is we set the diagonal elements of the adjacency matrix to one. Similarly we can set them to zero for disassortative networks or remain agnostic by treating self-connections as unobserved.

Importantly, each node can belong to any number from 0 to L communities, which is not the case in most existing methods. For instance in MMB, community affiliation probabilities are drawn from a simplex such that strong affiliation with one community makes affiliations with another less likely.

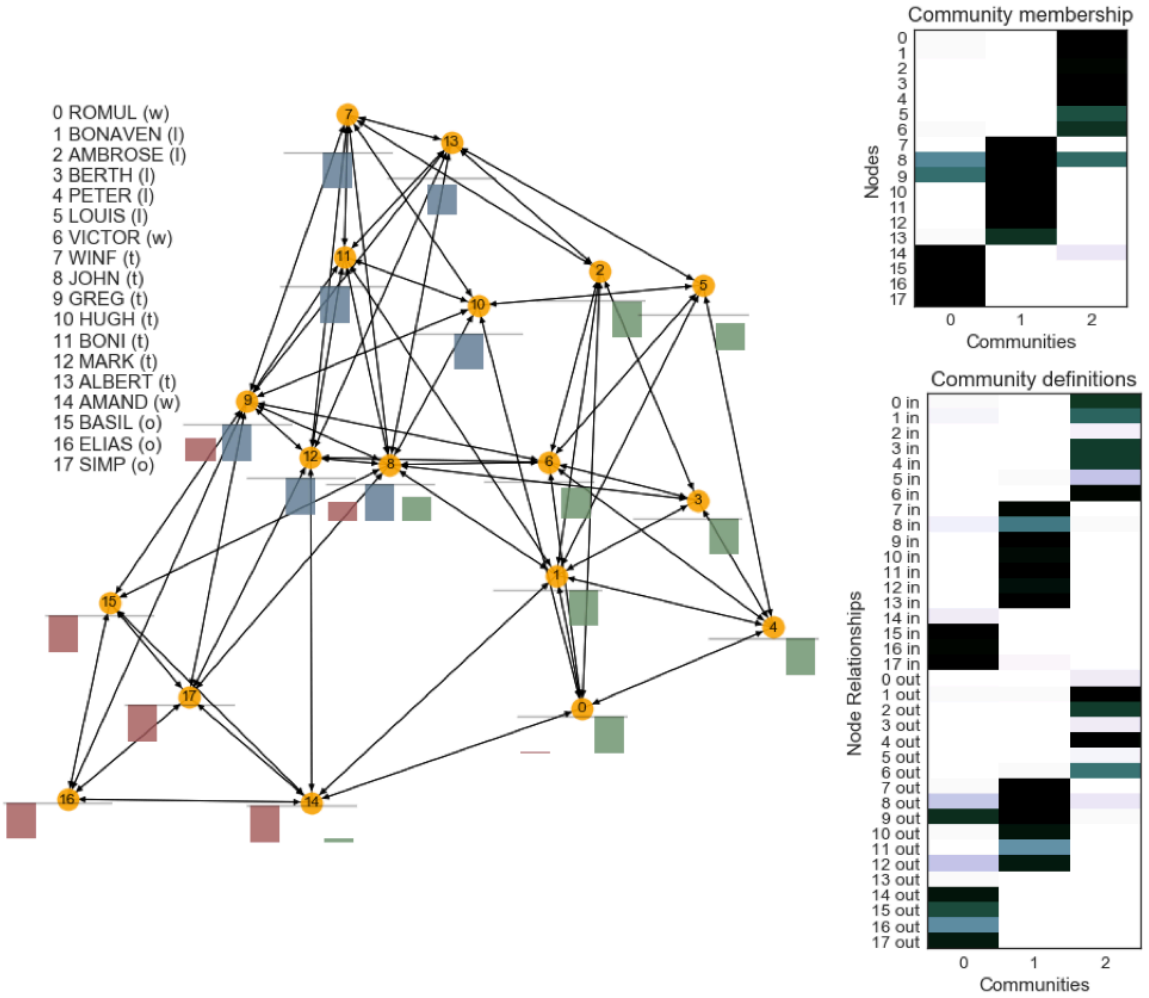


Fig. 2.15 Community detection in a network of monk relations. The small bar plots in the network indicate the posterior probability of association to each of the communities for every node. They correspond to the community affiliation matrix (middle). Each community is defined by a set of incoming and outgoing connections, the marginal posterior mean of which is shown in the community definitions matrix.

To illustrate community detection with the OrMachine, we analyse Sampson's popular monk dataset (Breiger et al., 1975; Sampson, 1968) which consists of directed relationships between 18 monks in a monastery and has served as an example for the MMB (Airoldi et al. (2008), Fig. 3). The relationships are based on a survey, that asked each monk about like/dislike of the other monks. In the original analysis Sampson suggests the existence of four groups, based on his months-long observations: the loyal opposition (l), the young

Turks (t), the outcasts (o) and the waverers (w). Since friendship networks are assortative, we encode each node as being connected to itself.

The OrMachine converges to solutions with the best cross-validation accuracy for three communities, measured by posterior predictive accuracy on 20% held out test-data. Results are shown in Fig. 2.15. The network is shown alongside the posterior means of the factor matrices which indicate community affiliation and the community definitions. The three communities correspond to Turks, outcasts, and the loyal opposition. Two waverers, Romul (0) and Victor (6) are explained part of the loyal opposition. The remaining waverer, Amand (14), is explained as outcast. We find that John (8) and Greg (9) play a role in multiple communities. All observations are in line with results for MMB shown by Airoldi et al. (2008) (Fig. 3), where the representations of John and Greg furthest away from the corners of the simplex. More subtle peculiarities, particularly about the directed edges, can be read from the community definitions matrix. Another analysis of the same dataset will be discussed in Section 4.7.2.

2.9 Infinite OrMachine

In the previous sections, we have assumed a finite number of latent dimensions, L. This number has been treated as hyperparameter and can be chosen heuristically or by cross-validation. Additionally, we have demonstrated empirically that the OrMachine naturally infers hierarchies of representations under variation of L. A more detailed discussion of hyperparameter optimisation for L is given in the context of tensor decomposition in Chapter 3.

Here, we lift this restriction and define a prior on the latent factors using the Indian Buffet Process (IBP). The IBP is a prior over binary matrices, where the entries in each column follow a Bernoulli distribution with parameter μ_l and where each μ_l is drawn independently

from a Beta-distribution:

$$\begin{aligned} \mu_l &\sim \text{Beta}\left(\frac{\alpha}{L}, 1\right) \\ z_{nl}|\mu_l &\sim \text{Bernoulli}(\mu_l) \end{aligned} \tag{2.10}$$

The IBP results from integrating out the μ_l and taking the limit as $L \to \infty$, such that the distribution has support over an infinite number of latent dimensions. More specifically, integrating out the μ_l in the finite model, we have

$$p(Z) = \prod_{l=1}^{L} \frac{\frac{\alpha}{L}\Gamma\left(m_l + \frac{\alpha}{L}\right)\Gamma\left(N - m_k + 1\right)}{\Gamma\left(N + 1 + \frac{\alpha}{K}\right)} \tag{2.11}$$

We only consider the columns with non-zero entries, $m_l > 0$, whose number is denoted L^+. Imposing a particular ordering on the columns and taking the infinite limit, $L \to \infty$ yields the distribution

$$p(Z) = \frac{\alpha^{L^+}}{\prod_{h=1}^{2^N-1} K_h!} \exp\left[-\alpha H_n\right] \prod_{l=1}^{L^+} \frac{(N - m_l)!(m_l - 1)!}{N!} \,. \tag{2.12}$$

Here $H_n = \sum_{i=1}^{N} 1/i$ and K_h counts the number of distinct histories of previous assignments across features. The details of this derivation are explicated by Ghahramani and Griffiths (2006).

The name, *Indian Buffet Process*, originates from a stochastic process that generates samples from the prior and that can be associated to the following narrative (Griffiths and Ghahramani, 2011). A sequence of customers, $n = 1, 2, \ldots, N$, enters a restaurant and chooses from an infinite number of dishes (L). The first customer picks a Poisson(α) number of dishes. Each remaining customer chooses any previously ordered dish with a probability proportional to the number of times it has been ordered. Additionally, she orders a Poisson(α/N) random number of new dishes. Importantly, the resulting distribution is ex-

changeable, that is, independent of the order in which the customers enter the restaurant. The general approach of turning the problem of choice of dimensionality into a problem of Bayesian inference over an unbounded number of dimensions is known as Bayesian Nonparametrics. A more detailed treatment is beyond the scope of this thesis and the interested reader is referred to Gershman and Blei (2012); Ghahramani (2012); Hjort et al. (2010) for a general treatment of Bayesian Nonparametrics and to Doshi-Velez et al. (2009); Griffiths and Ghahramani (2011) for overviews of the Indian Buffet Process.

In the following Section we derive an efficient Gibbs sampler for the OrMachine with the infinite limit of the Beta-Binomial prior in IBP representation. This approach has previously been studied in similar models by Wood et al. (2012) and Meeds et al. (2007). Despite this similarity to previous work, our approach is methodologically interesting because the conditional distribution over the number of new latent dimensions takes a very simple form. Due to the Boolean disjunction, any new latent dimension only affects the likelihood of observations that are not explained as ones and summing over new parameter configurations amounts to counting true negative and false negative predictions. Thus we can sample in the collapsed representation with μ_l integrated out[5].

2.9.1 Posterior Inference

The full joint distribution and the likelihood are given in eqs. (2.1) and (2.2). The model is completed by IBP prior one one factor matrix,

$$Z \sim \text{IBP}(\alpha),$$

[5] A simple and popular alternative is the stick-breaking construction by Teh et al. (2007) where the μ_l are represented explicitly. Using this representation, it would be possible to construct a model with IBP-prior on both factors.

while we continue using an independent Bernoulli prior on the other factor,

$$p(U|q) = \prod_{d,l} q^{u_{ld}} (1-q)^{1-u_{ld}} \ . \tag{2.13}$$

As previously, the hyperparameter, q, is set to 0.5 if not mentioned otherwise. In order to retain a greater degree of symmetry between U and Z, we could alternatively choose a finite Beta-Bernoulli prior over the independent columns of U. However, we refrain from doing so, because it would prohibit parallel inference for the rows of U and the practical difference is negligible. Thus inference for U proceeds as described in the previous sections.

Next we discuss inference for Z. The number of columns, L, is notionally infinite and, in practice, denotes the number of columns with at least a *one*. The infinitely many other columns do not affect the likelihood and therefore do not need to be represented explicitly. The same reasoning applies to the corresponding columns of U. We define the number of ones per column as

$$m_l = \sum_{n=1}^{N} z_{nl} \tag{2.14}$$

Similarly, $m_{-n,l}$ omits the row n in the summation, denoting the number of times feature l has been chosen by observations $n' \neq n$. Thanks to the exchangeability of the IBP, we can assume that customer n is the last one to enter the restaurant. Thus, she chooses each previously chosen feature with a probability $\frac{m_{-n,l}}{N}$ and a Poisson(α/N) number of previously unused features.

Updates for existing codes

If $m_{-n,l} > 0$, we sample from the conditional as before, but with the infinite Beta-Bernoulli prior, $p(z_{nl}{=}1|\mathbf{z}_{n,-l}) = \frac{m_{-n,l}}{N}$. In analogy to eq. (2.3), we find

$$p(z_{nl} = 1|.) = \sigma\left[\text{logit}\left(\frac{m_{-n,l}}{N}\right) + \lambda \sum_d \tilde{x}_{nd} u_{dl} \prod_{l' \neq l} (1 - z_{nl'} u_{dl'})\right] . \tag{2.15}$$

It becomes clear that the prior contribution couples the rows of Z, such that updates can not be computed in parallel.

Sampling new codes

In practice, we only need to represent columns with non-zero entries. However, we still need to sample from the remaining columns. Let L'_n denote the number of columns of Z that contain a 1 only in row n and change the notation such that L denotes the number of remaining columns with non-zero entries. We can compute the probability of L'_n in order to sample the number of such columns. This corresponds to the number of new dishes ordered by customer n and is independent of the other rows of Z such that the conditional distribution is given by

$$\begin{aligned} p(L'_n|.) &= p(L'_n|\mathbf{x}_n, \mathbf{z}_{n,l=1:L+L'_n}, U_{d=1:D,\, l=1:L}) \\ &\propto p(\mathbf{x}_n|\mathbf{z}_{n,l=1:L+L'_n}, U_{d=1:D,\, l=1:L}, L'_n)\; p(L'_n) \; . \end{aligned} \tag{2.16}$$

The prior is Poisson(α/N) and the likelihood factorises over d,

$$p(\mathbf{x}_n|\mathbf{z}_{n,l=1:L+L'_n}, U, L'_n) = \prod_d p(x_{nd}|\mathbf{z}_{n,l=1:L+L'}, \mathbf{u}_{d,\,l=1:L}, L'_n) \; . \tag{2.17}$$

This can be computed by marginalising over the new columns in U,

$$\begin{aligned} & p(x_{nd}|\mathbf{z}_{n,l=1:L+L'}, \mathbf{u}_{d,l=1:L}, L'_n) \\ &= \sum_{\mathbf{u}_{d,l=L+1:L'}} p(x_{nd}|\mathbf{z}_{n,l=1:L+L'_n}, \mathbf{u}_{d,l=1:L+L}, L'_n)\, p(\mathbf{u}_{d,l=L+1:L'_n}) \\ &= \sum_{\mathbf{u}_{d,l=L+1:L'_n}} \sigma\left[\lambda\tilde{x}_{nd}\left(1 - 2\prod_{l=1}^{L}(1 - z_{nl}u_{ld})\prod_{l=L+1}^{L'_n}(1 - u_{ld})\right)\right] p(\mathbf{u}_{d,l=L+1:L'_n})\,. \end{aligned} \tag{2.18}$$

Note, that for positive predictions, that is for x_{nd}, where $\exists\, l \leq L' : z_{nl}u_{dl}{=}1$, the term in parentheses in the last row of eq. (2.18) is independent of the entries in the new columns of U. The intuition is, that the logical disjunction explaining these data-points already emits a *one*, independent of any additional arguments.

Taking the logarithm of eq. (2.17), we have a sum over the two different types of data-points, the previously mentioned positive predictions, $\bar{P}$, and the negative predictions, $\bar{N}$, defined as x_{nd}, where $\nexists\, l : z_{nl}u_{dl} = 1$. For the positive predictions this sum evaluates to

$$\sum_{\bar{P}} \log p(\mathbf{x}_n|\mathbf{z}_{n,l=1:L+L'_n}, U, L'_n) = \bar{P}\log\left[\sigma(\lambda\tilde{x}_{nd}) \sum_{\mathbf{u}_{d,l=L+1:L'_n}} p(\mathbf{u}_{d,l=L+1:L'_n})\right] \tag{2.19}$$

$$= \bar{P}\log\sigma(\lambda\tilde{x}_{nd}) \tag{2.20}$$

This term is independent of L'_n and will cancel when normalising the probabilities for different values of L'_n. For the negative predictions we have two cases to consider for the L'_n, new, previously unused columns of U. Since customer n is the first to choose any of these columns, only the corresponding row, $\mathbf{u}_{d,l=L+1:L'_n}$, is relevant for the likelihood. Further, the Boolean disjunction emits a one, if any entry in this row is one and emits a zero otherwise.

There exists a single configuration for the latter case where all new entries are zero. We have

$$\begin{aligned}
&\sum_{\bar{N}} \log p(\mathbf{x}_n | \mathbf{z}_{n,l=1:L+L'_n}, U_{d=1:D,l=1:L}, L'_n) \\
&= \sum_{\bar{N}} \log \left[p(\mathbf{u}_{d,l=L+1:L'_n} = \mathbf{0})\, \sigma(-\lambda \tilde{x}_{nd}) + p(\mathbf{u}_{d,l=L+1:L'_n} \neq \mathbf{0})\, \sigma(\lambda \tilde{x}_{nd}) \right] \\
&= \sum_{\bar{N}} \log \left[q^{L'_n}\, \sigma(-\lambda \tilde{x}_{nd}) + (1 - q^{L'_n})\, \sigma(\lambda \tilde{x}_{nd}) \right] .
\end{aligned} \tag{2.21}$$

Next, we subdivide the negative predictions into true negatives (TN), where $x_{nd} = 0$ and false negatives (FN) where $x_{nd} = 1$. True positives and false positive are defined analogous, but can be ignored in the computation. Together with eq. (2.20) for the positive predictions we find

$$\begin{aligned}
\log p(\mathbf{x}_n | \mathbf{z}_{n,l=1:L+L'}, U, L') = \mathrm{TP} \log \sigma(\lambda) &+ \mathrm{FP} \log \sigma(-\lambda) \\
&+ \mathrm{FN} \log \left[q^{-L'_n} \sigma(-\lambda) + \left(1 - q^{-L'_n}\right) \sigma(\lambda) \right] \\
&+ \mathrm{TN} \log \left[q^{-L'_n} \sigma(\lambda) + \left(1 - q^{-L'_n}\right) \sigma(-\lambda) \right]
\end{aligned} \tag{2.22}$$

With the Poisson prior in eq. (2.16) we can now compute the probability for new values L'. We truncate the distribution over L', by sampling only for $L' < 10$. Note, that we can pre-compute the terms in the square brackets. These precomputed quantities need only be updated for a new values of λ. Thus, sampling L'_n mainly amounts to counting the number of true positive and true negative predictions in the current configuration of the factors. The sampling procedure is sketched in Algorithm 3.

2.9.2 Evaluation on Simulated Data

We generate synthetic data of size 200×500 with balanced density. 100 samples are drawn after burn-in and each experiment is repeated ten times. Posterior mean and modes indicate the ability to recover the true data-generating dimensionality as shown in Fig. 2.16. Standard

Algorithm 3 Sampling from Z with IBP prior

for $n = 1, \ldots, N$ **do**
  for $l = 1, \ldots, L$ **do**
    if $m_{-n,l} > 0$ **then**
      sample z_{nl} from eq. (2.15)
    else if $m_{-n,l} = 0$ **then**
      Remove column l from Z and U.
    end if
  end for
  Draw number of new L'_n following eqs. (2.16) and (2.22).
  Set $z_{n,l=L+1:L'_n} = 1$
  for $l = L + 1, \ldots, L'_n$ **do**
    for $d = 1, \ldots, D$ **do**
      sample u_{dl} as previously, analogous to eq. (2.3)
    end for
  end for
  $L = L + L'_n$
end for

deviations from the repetitions are shown as error-bars. We find that the model reliably recovers the ground-truth number of latent dimensions. Mean and mode estimates are in close agreement, indicating that the distribution is not skewed. We repeat these experiments under noisy conditions in Appendix B, where find close-to perfect recovery for a noise level of 10% and a systematic overestimation of roughly 1 latent dimension for a noise level of 20%. For a higher noise level of 30%, the overestimation increases to roughly two dimensions for smaller dimensionality and is off by 10 or more dimensions for more than 10 true underlying dimensions.

2.9.3 Practical Applicability of the IBP-OrMachine

The sampling process that was described in the previous sections, in particular in Algorithm 1 is trivially parallelisable. This enables scalability to very large datasets that has been demonstrated in Section 2.5 and will further be made use of in the Chapter 3. In contrast, sampling from the IBP (Algorithm 3) in parallel is challenging. A recent approach has

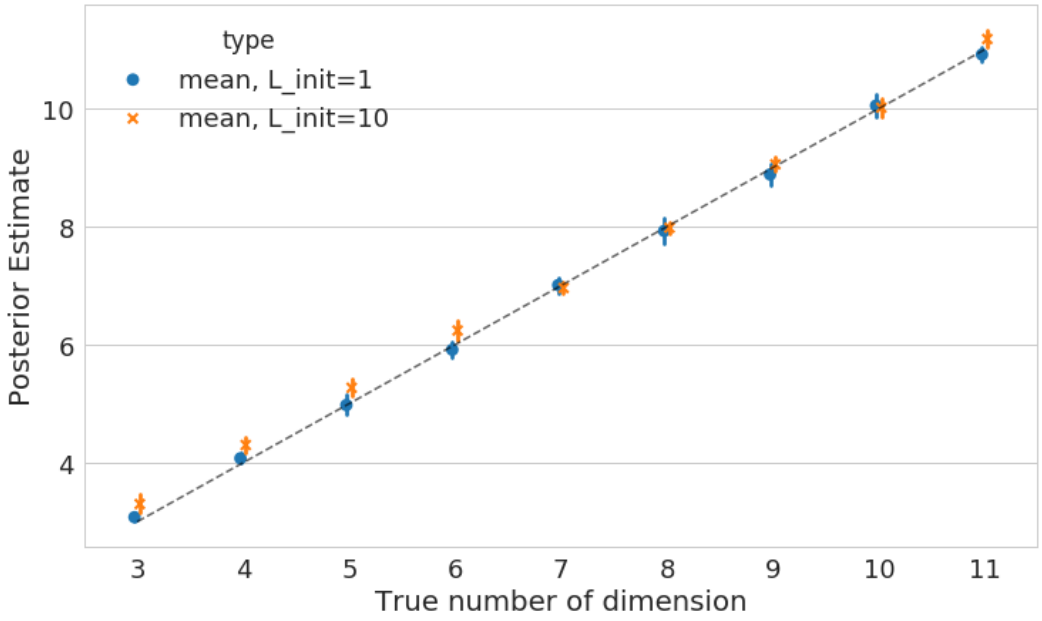


Fig. 2.16 Inference over the number of latent dimensions in the infinite OrMachine. Data of size 200×500 is generated from random factors with expected density of ½. The plot indicates the posterior modes and means of the distribution over the number of latent dimensions. Error bars denote standard deviations from 10 random repetitions. L was initialised to 1 and 10 to probe sensitivity. The IBP hyperparameter was chosen to $\alpha = 2$.

been described by Zhang et al. (2017) but requires a change between the IBP and the stick-breaking representation. Since scalability is a major benefit of the OrMachine and suitable tools for hyperparameter optimisation exist, we limit the discussion of Bayesian Nonparametric approaches to this section. In practice, the proposed IBP-OrM is well suitable for data of moderate size or in cases where only a single core is available. Around 10^7 data points are tractable on a laptop with a hundred samples being drawn in a few minutes. For larger datasets, using standard parallel OrM is advisable.

2.10 Conclusion

In this chapter, we have developed the OrMachine, a probabilistic model for Boolean matrix factorisation. The extremely efficient Metropolised Gibbs sampler outperforms state-of-the-art methods in matrix factorisation and completion. It is the first method that infers posterior

distributions for Boolean matrix factorisation, a property which is highly relevant in practical applications where full uncertainty quantification matters.

Despite full posterior inference, the proposed method scales to very large datasets. We have shown that tens of billions of data points can be handled on commodity hardware. Multi-layer models can be used to integrate different types of information and, akin to hierarchical PCA (Bishop and Tipping, 1998), infer representations at different levels of abstraction. This leads to improved reconstruction performance on simulated and real world data. Moreover, we have demonstrated the OrMachine as a flexible model for community detection in networks. Handling of missing links, assortativity and disassortativity, representation of uncertainty, scalability and a simple and intuitive community-generating process make it an appealing alternative to existing methods.

Finally, we have used the Indian Buffet Process as infinite-dimensional prior on one of the factor matrices. It enables inference of the posterior distribution of the number of latent dimensions and successfully recovers the true underlying dimensionality in synthetic data. The updates take a simple form that enables fast computation. However, the IBP representation does not allow for parallel computations which ultimately limits scalability. This leads us to explore other means of choosing the dimensionality in the data-intensive applications of the next chapter.

Chapter 3

Boolean Factorisation of Polyadic Data

This chapter describes a generalisation of the OrMachine to ternary data and more generally to data of arbitrary arity. Thus, in addition to binary relations between two objects, we can analyse binary relations between any number of objects. Section 3.1 reviews related work, before we specify the model and discuss the inference and prediction problem in Section 3.2. In Section 3.3 the performance of our approach is compared to the previous state-of-the-art method and approaches to choosing the correct model dimensionality are investigated. We demonstrate a variety of real-world applications in Chapter 3.4, including temporal interaction networks of hospital staff (Section 3.4.1), the relational binary encoding of continuous gene-expression data (Section 3.4.3) and single-cell gene expression across different tissue types (Section 3.4.4). We continue by outlining the intuition behind Boolean tensor decomposition.

Matrix decomposition methods, as discussed in the previous chapter, operate on two-way matrices with rows thought of as objects and columns thought of as properties. The decomposition amounts to computing two low-rank matrices, one that contains prototypes of properties and one that denotes a compressed representation of each object as a combination of these prototypes. However, these methods are ill-suited for data with higher arity, for example ternary interactions such as object $\times$ property $\times$ condition tuples. Such data requires

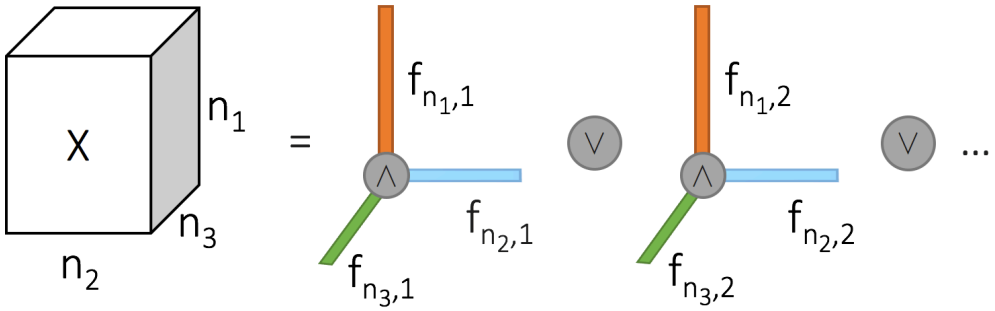


Fig. 3.1 Graphical intuition for 3-way Boolean tensor decomposition. Logical conjunction ($\wedge$) of Boolean vectors (columns of factor matrices) generates rank-1 tensors. The full tensor is the logical disjunction ($\vee$) of rank-1 tensors.

methods that specifically account for the higher-order relationship and that are commonly referred to as tensor decomposition (Kolda and Bader, 2009).

In analogy to Boolean Matrix Factorisation, Boolean tensor factorisation decomposes a K-way binary tensor $X \in \{0,1\}^{N_1 \times \ldots \times N_K}$, into K binary factor matrices $F_k \in \{0,1\}^{N_k \times L}$, using the Boolean algebra such that

$$x_{[\mathbf{n}]} \approx \bigvee_{l=1\ldots L} \left[\bigwedge_{n \in [\mathbf{n}]} f_{nl} \right] . \tag{3.1}$$

Here, $x_{[\mathbf{n}]}$ is a single tensor entry and $[\mathbf{n}]$ denotes a tuple of indices $[n_1, \ldots, n_K]$. We call $k = 1 \ldots K$ the *modes* of the tensor. In plain English, eq. (3.1) says that an observation is 1, if and only if there exist one or more latent dimensions in which all corresponding factors are 1. This leads to easily interpretable factor matrices, F_k, where each latent dimension denotes a subset of entries along mode k, for instance subsets of objects, properties and conditions that occur jointly in the data. To avoid clutter, we slightly abuse the notation such that n indexes both, the tensor modes and the entries along each mode. The meaning will be evident from the context.

Boolean Tensor Decomposition can be thought of as a particular type of canonical polyadic (CP) decomposition, as illustrated in Fig. 3.1. Here, a rank-1 tensor is composed by combining a latent column from every factor into a tensor with K modes using the AND-operation.

This is done for all latent dimensions, l. The resulting rank-1 tensors are then combined into a single tensor using the OR-operation. A different graphical intuition starts from the factor rows and considers the 3-way Boolean tensor product as a three-stage template assignment procedure. Rows of a factor matrix F_1 are one-dimensional binary templates of size of the first tensor mode. Rows of the second factor matrix F_2 indicate possible patterns of appearance of these templates along the second tensor dimensions. In the same manner, the third factor matrix, F_3, indicates which disjunction of these 2D patterns appears in each slice of the tensor, and so forth.

In Section 3.4.1 we will discuss an application of this model to a temporal interaction network. We briefly sketch the setup here to provide the reader with furhter intuition on how the decomposition works. The data consists of interactions of individuals represented as binary adjacency matrix. Measurements are taken at different points in time such that the adjacency matrices can be stacked to form a tensor. Boolean Tensor decomposition infers a factor matrix for each axis of the tensor. In the case at hand we have two, equivalent matrices that denote communities of individuals and one matrix that denotes points in time when these communities interact. A data point is modelled as 1, if the exists any latent community (column in the community specific factors) that contains the corresponding individual AND that is described as active at the corresponding point in time (column in the time specific factor).

In this chapter, we present the first probabilistic approach to Boolean tensor decomposition, the *TensOrMachine*, featuring distinctly improved accuracy compared to the previous state-of-the-art methods. Inference proceeds mostly analogous to the OrMachine in Chapter 2 and is therefore treated in a concise manner.

3.1 Related Work

Tensor decomposition methods are widely used in many domains of application, for instance for the analysis of electronic health records (Henderson et al., 2017) or in genomic applications (Ho et al., 2014). A comprehensive review is given by Kolda and Bader (2009).

Boolean Tensor Decomposition was first introduced in the psychometric literature by Leenen et al. (1999) and has received lasting attention, following the formal study by Miettinen (2011). The author shows that computing the optimal decomposition is NP-hard and proposes an alternating least squares heuristics as approximate strategy. Another greedy approach, rooted in formal concept analysis, is devised by Belohlavek et al. (2013). A further algorithm, based on a random-walk procedure, is described by Erdos and Miettinen (2013b) and trades accuracy in the factors for computational scalability. More recently a distributed Apache Spark implementation of the alternating least squares approach has been brought forward (Park et al., 2017). It demonstrates that alternating leasts squares is the state-of-the-art for computing accurate decompositions and therefore serves as baseline in our experiments.

Despite the sustained theoretical interest, there have been only few practical applications of Boolean tensor decomposition, as for instance for information extraction (Erdos and Miettinen, 2013a) and clustering (Metzler and Miettinen, 2015). One of the contributions of this chapter is the presentation of Boolean Tensor decomposition as an interpretable, versatile and scalable analysis method.

3.2 Inference and Prediction

We specify the probabilistic generative process for the model described in eq. (3.1). In analogy to Section 2.2, each tensor entry, $x_{[\mathbf{n}]} = x_{[n_1,\ldots,n_K]}$, is a Bernoulli random variable that

equals 1 with a probability greater than ½ if the following holds true

$$\exists\, l \,:\, f_{nl} = 1 \;\forall\; n \in [n_1, \ldots, n_K]. \tag{3.2}$$

Compared to the OrMachine, f generalises the factor matrices Z and U. The graphical model in plate notation, shown for the 2D case in Fig. 2.2a on page 17 is augmented by another latent variable that shares its latent dimension with all other latent factors and its feature dimension with the data. The logistic sigmoid, $\sigma(x) = (1 + e^{-x})^{-1}$, lets us readily parametrise the likelihood:

$$p(x_{[\mathbf{n}]}|.) = \sigma\left[\lambda\tilde{x}_{[\mathbf{n}]}\left(1 - 2\prod_l\left[1 - \prod_{n\in[\mathbf{n}]} f_{nl}\right]\right)\right]. \tag{3.3}$$

The term inside parenthesis evaluates to 1 if the condition in eq. (3.2) is met and to -1, otherwise. We can specify Bernoulli priors on the observations $x_{[\mathbf{n}]}$ or choose more structured prior beliefs. As before, a beta-prior on $\sigma(\lambda)$ is a computationally convenient choice but is easily outweighed by the observed data in the relevant regime of at least thousands of observations.

Conditioning any entry of a factor matrix, $f_{n_k l}$, on the state of all other parameters yields the full conditional probability

$$p(f_{n_k l}|.) = \sigma\left(\lambda\tilde{f}_{n_k l} \sum_{\substack{[\mathbf{n}] \\ n_k \text{fixed}}} \tilde{x}_{[\mathbf{n}]} M_{(n_k,l)\to[\mathbf{n}]}\right). \tag{3.4}$$

Here $M_{(n_k,l)}$ is defined as

$$M_{(n_k,l)\to[\mathbf{n}]} = \Big(\prod_{n\in[\mathbf{n}]/n_k} f_{nl}\Big)\prod_{l'\neq l}\Big(1 - \prod_{n\in[\mathbf{n}]} f_{nl'}\Big). \tag{3.5}$$

The formal derivation of this expression is given in Appendix A and intuition briefly summarised in the following: The sum in eq. (3.4) is taken over all observations $x_{[\mathbf{n}]}$ whose likelihood may depend on $f_{n_k l}$. These observations are given by the $(K-1)$–way sub-tensor of X with dimension n_k fixed and correspond to all children of $f_{n_k l}$ in a graphical model representation. For each of them, the first term inside the sigmoid contributes values in $\{-\lambda, 0, \lambda\}$. This depends on whether or not $\tilde{f}$ and $\tilde{x}$ are aligned and on the indicator variable $M_{(n_k,l)\to[\mathbf{n}]}$ that take values in $\{0, 1\}$ and denotes whether the state of $f_{n_k l}$ has any relevance for the likelihood of $x_{[\mathbf{n}]}$.

This indicator variable, defined in eq. (3.5) is itself composed of two indicators, corresponding to the `V`-structures in Section 2.2.2. In particular, the *first term* in eq. (3.5) is a product over the state of all co-parents of $f_{n_k l}$ to observation $x_{[\mathbf{n}]}$ in the same latent dimension l. Following the rules of the Boolean product, all these co-parents need to be one in order for $f_{n_k l}$ to be relevant to the likelihood of $x_{[\mathbf{n}]}$. This corresponds to the `AND`-operation in eq. (3.1) that evaluates to zero if any its arguments are zero. The *second term* is a product of the co-parents in all other latent dimensions and evaluates to zero if any of them *explains away* observation $x_{[\mathbf{n}]}$. Following the rules of the Boolean product, a single latent dimension is sufficient to explain an observation and hence the state of $f_{n_k l}$ becomes irrelevant. This corresponds to the `OR`-operation in eq. (3.1) that evaluates to one if any of its arguments is one. It is crucial for the speed of our implementation that eq. (3.5) can be rapidly evaluated. In particular, discovering a zero in any of the two terms suffices for the whole expression to be zero and avoid the necessity of considering the full Markov blanket. Pseudocode for the computational procedure is given in Algorithm 4. The algorithm scales, again, linear in the latent dimensions and the observed dimensions. A naive implementation would require us to all co-parents of a variable into account, leading to quadratic scaling behaviour in the number of latent dimensions.

Algorithm 4 Computation of the full conditional of $f_{n_k l}$

```
m = 0
for [n] in all tensor indices with slice n_k fixed do
  for n in [n] do
    // check relevance of f_{n_k l} for x_[n].
    if f_nl = 0 then
      // f_{n_k l} has no relevance for x_[n].
      continue with next [n]
    end if
  end for
  // check for explaining away.
  for l' in 1, ..., L except l do
    for n in [n] do
      if f_{nl'} = 0 then
        continue with next l'
      end if
      // x_[n] is explained away.
      continue (next [n])
    end for
  end for
  m = m + x̃_nd
end for
p(f_{n_k l}|.) = (1 + exp(-λ · f̃_{n_k l} · m))^{-1}
```

Treatment of the dispersion parameter, λ, of missing data and data imputation proceeds as described for the OrMachine in Sections 2.2.3 and 2.2.4. For a direct comparison of imputation performance to competing methods that only provide a point estimate of the factors, we can reconstruct the data based on the Boolean product of the factor MAP estimates, where $f_{n_k l} \in [0, 1]$. Alternatively, to take uncertainty in the factors into account, we can resort to the plug-in approximation

$$p(x_{[\mathbf{n}]} = 1|.) \approx 1 - \prod_l \left(1 - \prod_{n \in [\mathbf{n}]} \hat{f}_{nl}\right) , \tag{3.6}$$

where $\hat{f}$ denotes the posterior mean of a factor entry. The imputation performance of these two estimates, together with the computationally expensive full posterior predictive distribution is compared in Fig. 3.3. The results indicate no loss in performance of the plug-in estimate compared to the full posterior predictive but highlight the importance of characterising posterior uncertainty, especially in noisy settings. They are discussed in more detail in the next section.

3.3 Experiments on Simulated Data

3.3.1 Random Tensor Factorisation

We first demonstrate the capabilities of the TensOrMachine on simulated data and compare to the state-of-the-art *distributed boolean tensor factorisation* (dbtf) (Park et al., 2017). We simulate random 3-way tensors, X, of size $20 \times 20 \times 20$ and vary their rank L, their expected density $\rho(X)$, and the noise-level. To this end, we take the Boolean product between binary i.i.d. random matrices F_k of size $20 \times L$, such that the expected density is given by $\rho(X) = 1 - \left[1 - (\rho(F_k))^3\right]^L$. We introduce noise by flipping each entry in the tensor with a given probability. Posterior samples of the factors are drawn, following the procedure described in Section 3.2. The reconstruction of the tensor is determined from the posterior predictive distribution. The reconstruction accuracy is computed as the fraction of correctly reconstructed entries in the noise-free tensor and is shown across a variety of conditions in Fig. 3.2. Our method achieves distinctly higher accuracies throughout all conditions. The margin becomes bigger for very noisy data, as well as for higher tensor ranks and for particularly dense or particularly sparse data.

Can the superior reconstruction performance be explained by the implicit model averaging when computing the posterior predictive distribution? In order to address this conjecture, Fig. 3.3 compares the reconstruction accuracies of the posterior predictive distribution to the

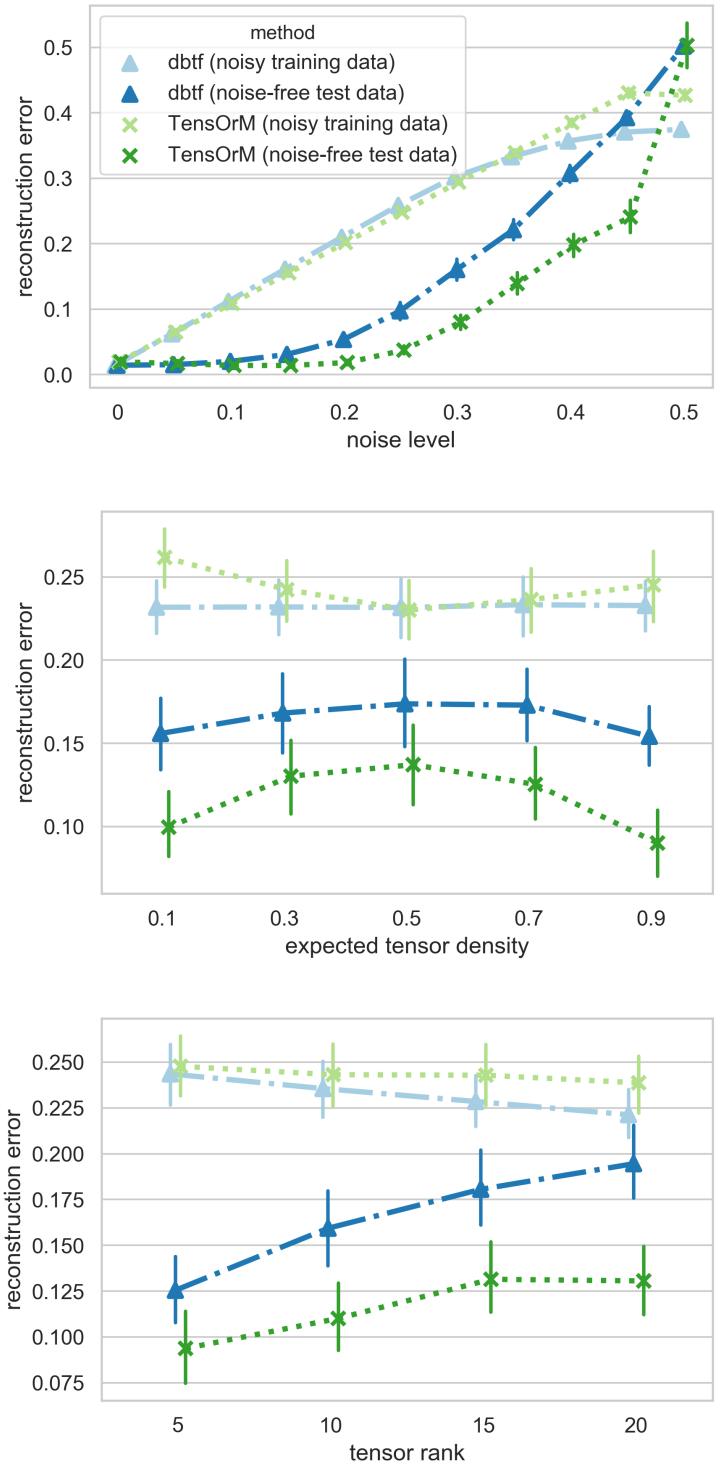


Fig. 3.2 Random tensor reconstruction error under variation of the noise level (top), the expected tensor density (centre) and the underlying tensor rank (bottom). Averages are taken across all shown combinations of the other two parameters and across ten random repetitions for each such configuration. Training accuracy on the noisy tensor is shown in faint colours. Compare to Fig. 8 in Park et al. (2017).

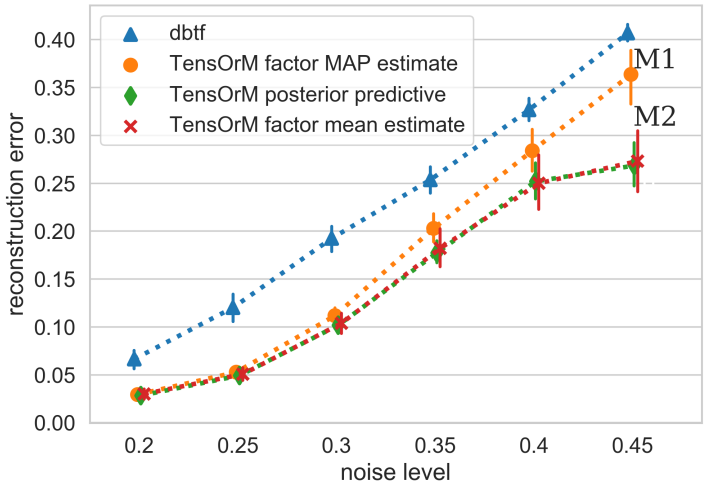


Fig. 3.3 Comparison of reconstruction methods – We compare predictions based on the TensOrMachine posterior predictive distribution, the factor matrix MAP, and mean to the dbtf baseline. M1 indicates the performance margin that is due to finding more accurate factor matrices, M2 denotes the margin that is due to posterior averaging. Results are shown for rank-10 tensors averaged across tensor densities and 10 random repetitions.

reconstruction based solely on the marginal MAP estimates of each factor. Reconstruction accuracy is measured with respect to noise-free simulated data while training proceeds with noisy data as indicated in the horizontal axis of the figure. We can see, that posterior averaging plays an important role only in scenarios with high noise levels. The main performance gain, however, is simply due to a more accurate decomposition. In the practically relevant regime of moderate noise levels below 30%, posterior averaging has virtually no impact. We further note in Fig. 3.2, that dbtf features a similar or higher reconstruction accuracy with respect to the noisy training data. This indicates over-fitting and becomes particularly apparent for large noise levels and large tensor ranks.

What distinguishes the decompositions of dbtf and the TensOrMachine in this regime? In Fig. 3.4 we show the training performance under random perturbations of the inferred factors for a noise level of 40% and an expected density of 10%. While the TensOrMachine has a lower training accuracy, it is much more stable towards random perturbations whereas the training accuracy of dbtf decreases rapidly. TensOrM recovers exactly the expected re-

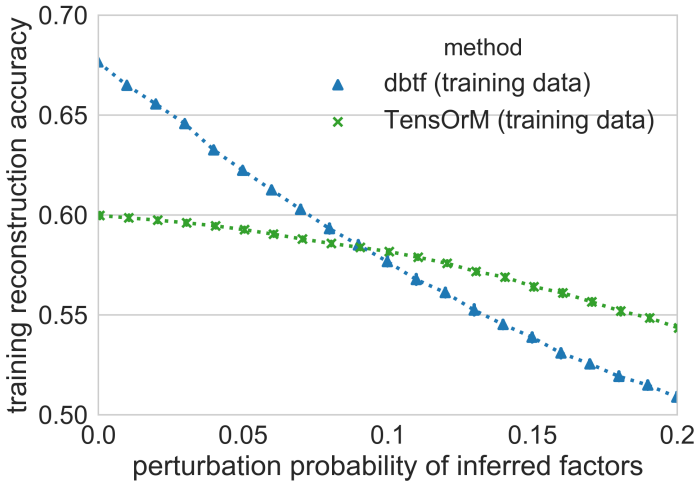


Fig. 3.4 **Robustness of training performance under random perturbations** – We show reconstruction accuracy on the noisy training data based on the factor matrix MAP estimate under random flips of the factor entries. The underlying tensor has a noise level of 40% and an expected density of 10%. We perform 10 random repetitions but standard deviations are too small to be visible.

construction accuracy of 60%, while dbtf overfits to the training data. Its reconstruction accuracy deteriorates rapidly under random perturbations, indicating that it has converged to a comparatively narrow mode of the posterior. This shows that TensOrM converges to solutions that occupy larger region of the parameter space. The probability of containing the true underlying solution is proportional to this volume and thus such solutions have better generalisation properties. Recently, this phenomenon has been studied in order to improve the generalisation of deep neural networks (Chaudhari et al., 2016). By estimating the posterior distribution, Bayesian inference naturally favours such solutions.

3.3.2 Model Selection

A notorious challenge for latent variable models is the choice of dimensionality. Previously, Erdos and Miettinen (2013b) have used the Minimum description length (MDL) principle for this task in Boolean Tensor Decomposition. We follow their derivation and compare to the

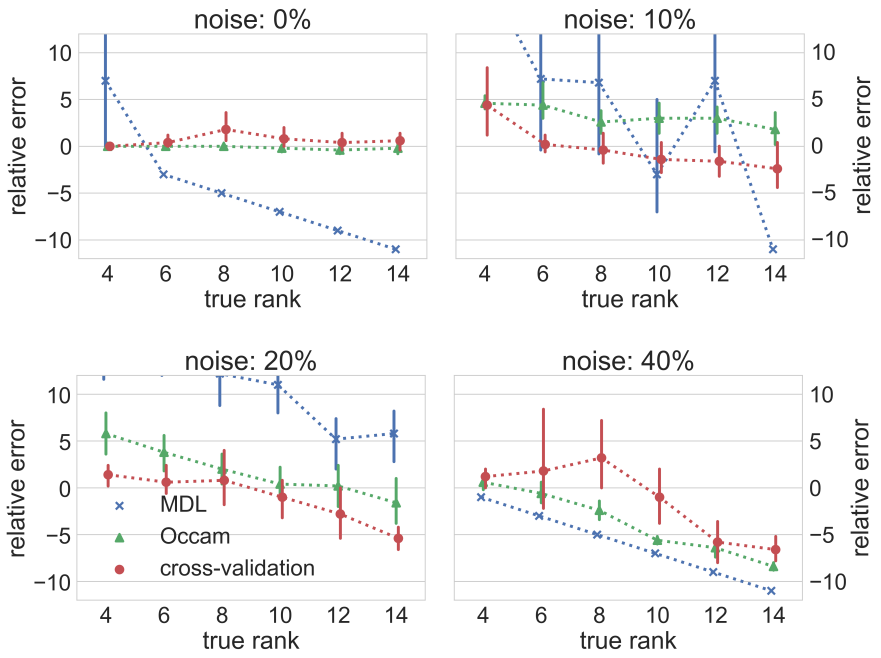


Fig. 3.5 **Model selection accuracy** in terms of the relative error in predicting the tensor rank. Model selection with an Bayesian Occam's razor or cross validation (Section 3.3.2) outperform the previously suggested MDL approach.

two approach for model selection that our Bayesian treatment readily offers. The **Bayesian Occam's razor** is a simple heuristic. Its naming is inspired by an observation by MacKay (1992), further discussed by Murray and Ghahramani (2005), where the author describes how Bayes rule automatically penalises unnecessarily complex models. For this approach, we start our inference procedure with a large latent dimensionality. After convergence of the Markov chain, we remove all latent dimensions that do not contribute to the likelihood. In these dimensions one of the factors is usually all zeros and the other factors are uniformly random, such as observed for the 8-dimensional model in the calculator digit hierarchy in Fig. 2.8 on page 31. After removal we restart the burn-in procedure and repeat until only contributing dimensions remain. In the second approach, **cross validation**, we treat 20% of the data as unobserved during training and choose the model dimensionality that achieves the highest posterior predictive on the held-out data. Results on random matrices for different noise levels are shown in Fig. 3.5, following the previously described simulation procedure.

We find that both approaches clearly outperform MDL. In the case of noise-free observations the Bayesian Occam's razor features virtually perfect accuracy. For the more realistic scenario of moderate noise levels cross validation is superior.

Importantly, these methods make use of parallel compute resources which, for bigger datasets, is an important advantage over the more elegant nonparametric approach described in Section 2.9.

3.4 Real-World Applications

Here we demonstrate the ability of the TensOrMachine to infer meaningful representations from 3-way relations in real-world datasets. Examples of temporal interaction networks and temporal object-property-data showcase interpretability of the latent space. Datasets in the first to examples are of moderate size with less than 100,000 data-points. Eventually, we turn to a large-scale biological example with around 10 billion data points, analysing networks of relative gene-expression in cancer patients. We conclude with an example of multi-tissue gene expression from single cells where TensOrM infers gene-sets in striking accordance to the originating tissue.

3.4.1 Hospital Ward Interaction Networks

Records of contact between all pairs of individuals have originally been acquired to investigate transmission routes of infectious diseases in a geriatric university hospital in Lyon (Vanhems et al., 2013). Proximity sensors measurements were taken for 75 individuals in 20 second time windows across 5 days. In order to examine daily patterns, we group the time-points into 13 time-of-day windows. This leaves us with a binary 13×75×75 time-adjacency tensor. We compute the decomposition, choosing 8 latent dimensions based on cross-validation as described in the previous section. The predictive accuracy on the held-out test data is ap-

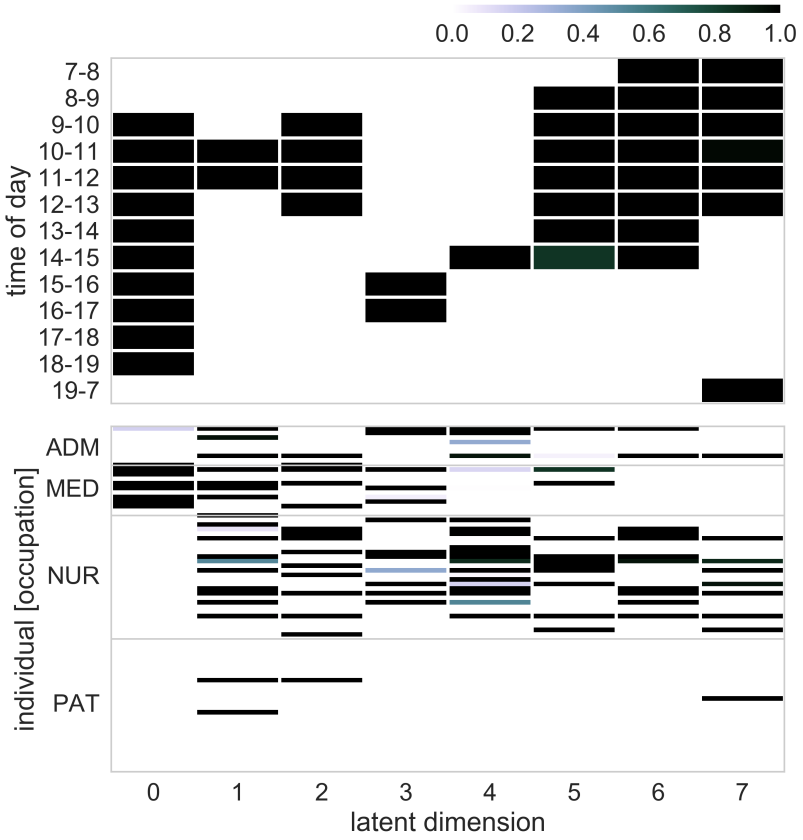


Fig. 3.6 **Dynamic hospital interaction networks** – Decomposition on the 13×75×75 time-adjacency-tensor of interactions in a hospital ward. We only show one of the two equivalent individual-specific factors.

proximately 91%. Fig. 3.6 shows posterior means of the time-specific and (one of the two equivalent) individual-specific factor matrices. Individuals are labelled as either administrative staff (ADM), medical doctors (MED), nurses (NUR) or patients (PAT). Using this proximity data alone, we were able to recapitulate information about the work patterns of staff groups. For example, 10:00am – 12:00pm and 3:00pm – 5:00pm represent the main morning and afternoon clinic hours when all administrative, medical and nursing staff came into interaction (dimensions 1 and 3). Whilst medical doctors are closely interacting throughout typical work hours, 9:00am – 7:00pm (dimension 0), their network does not include any nurses or patients which can be explained by frequent interaction in the doctor's room but also hints to a lack of interaction with the nursing staff, a phenomenon frequently discussed

in the literature (see e.g. Manias and Street, 2001). 2:00pm – 3:00pm indicates a period in which nurses and administrative staff were heavily interacting without participation of medical doctors, presumably indicating an afternoon break or daily joint meeting (dimension 4). Nurses, however, were active throughout the day including being the main staffing body at night (dimension 7).

3.4.2 Student Seating Position

Our second example is part of the student-life dataset introduced by Harari et al. (2017). The data contains records of the seating positions of students in an Android programming class, measured throughout the 9-week duration of the course. We partition the time-points into weeks of the course and the seating positions into six regions front/back – left/centre/right. An additional seating category is labelled with a question mark, where the seating coordinates were not provided but the students appeared in class. This yields a 9×7×42 week-seat-student tensor. We choose seven latent dimensions, again, by optimising for the test-set likelihood with a reconstruction accuracy of approximately 92% on the held out data. The decomposition is shown in Fig. 3.7 and, once more, open to straightforward interpretation. The majority of students shows up to class in the first week of the course and sits scattered throughout the lecture hall, as can be seen in dimensions 0 and 1. The most diligent group of students, dimension 4, sits in the front-centre of class and is most persistent in attending the lectures. The remaining groups describe subsets of students, each corresponding to exactly one of the four front/back left/right seating regions. They all eventually stop to appear in class after 4-6 weeks.

3.4.3 Networks of Relative Gene Expression in Cancer

Gene expression profiling has been used extensively in cancer research, for example for the characterisation of cancer sub-types, for the stratification of patients or for the identification

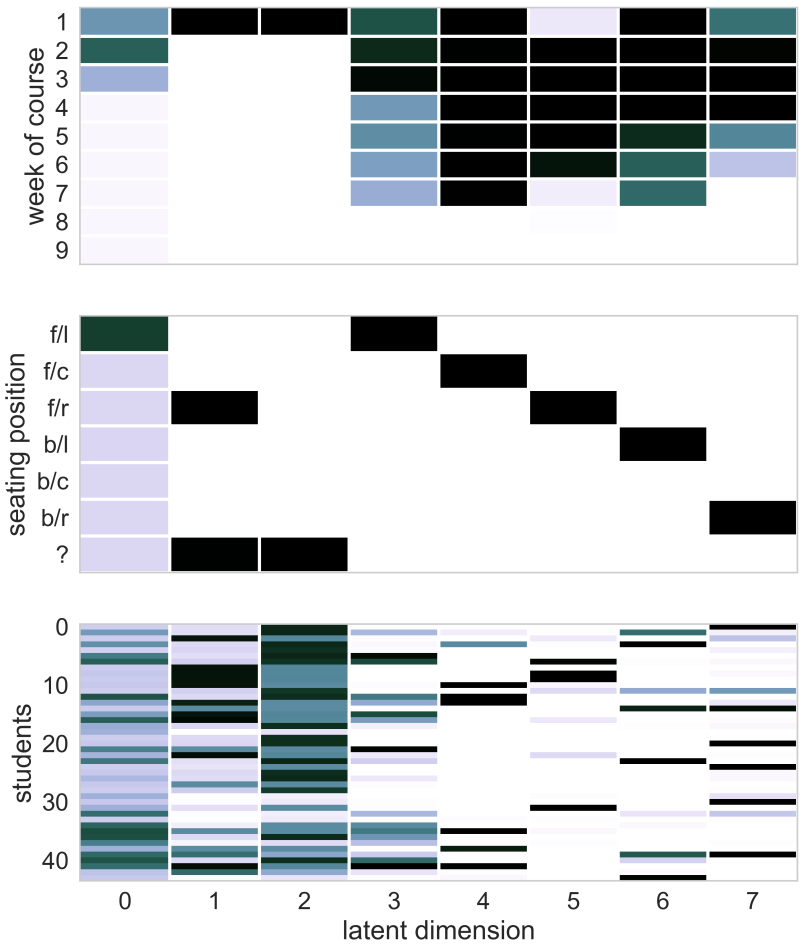


Fig. 3.7 **Student dynamic seating throughout course** – Decomposition of the 9×7×42 week-seat-student tensor indicating seating positions for a 9-week university course on Android programming.

of therapeutic targets. Correlations in gene expression are at the basis of protein interaction in cellular pathways and latent variable models are frequently applied to extract biologically meaningful information from such data. We propose a novel way of analysing gene expression data that can, in principle, be applied to any continuous measurement with object-by-attribute (here patient-by-gene) structure. The publicly available TCGA dataset[1] (Weinstein et al., 2013) contains gene expression measurements for more than 10,000 cancer patients. After preprocessing we are left with approximately 8,000 patients and 1,100 genes. First, we

[1]http://cancergenome.nih.gov/

normalise the expression values for each gene across all patients to allow for comparison between different genes. Then the data is transformed into a 3-way tensor using the relational encoding

$$(\text{patient}, \text{gene}_i, \text{gene}_j) = \begin{cases} 1; \text{ if } \text{expr}_i > \text{expr}_j \\ 0; \text{ if } \text{expr}_i < \text{expr}_j \\ \text{unobserved; else .} \end{cases} \tag{3.7}$$

This relational coding provides an entirely new view of the data in terms of sparse networks of relative expression. For every latent dimension, the two gene specific factor matrices indicate a subset of genes, such that genes in the first subset are likely to be more expressed than genes in the second subset. The patient factor indicates which of these properties of relative expression each patient exhibits. Importantly, the factors are amenable to distinct and intuitive interpretation with immediate biological relevance. We show the patient-specific factor in Fig. 3.8 with patients ordered by cancer-type. We observe a remarkable disease specificity with many latent expression networks being exclusively present in virtually all cancers of certain types. In addition, some latent properties are scattered throughout cancers of various types, highlighting the heterogeneity of the disease. The corresponding gene-specific factors characterise each latent network on the molecular level but are more difficult to analyse for a non-specialist. In particular, the richness of the relational latent encoding can only partially be captured by traditional pathway analyses. Nevertheless, we investigate the biological plausibility of the inferred gene sets by running a gene set enrichment analysis of the genes in each factor against the KEGG pathway database[2].

Application of a traditional method, principal component analysis (PCA) on the continuous patient by gene array, is shown in Fig. 3.9. It lacks both, interpretability and the degree of disease specificity. We use the latent representations of both methods as features for ran-

[2]http://www.genome.jp/kegg/pathway.html

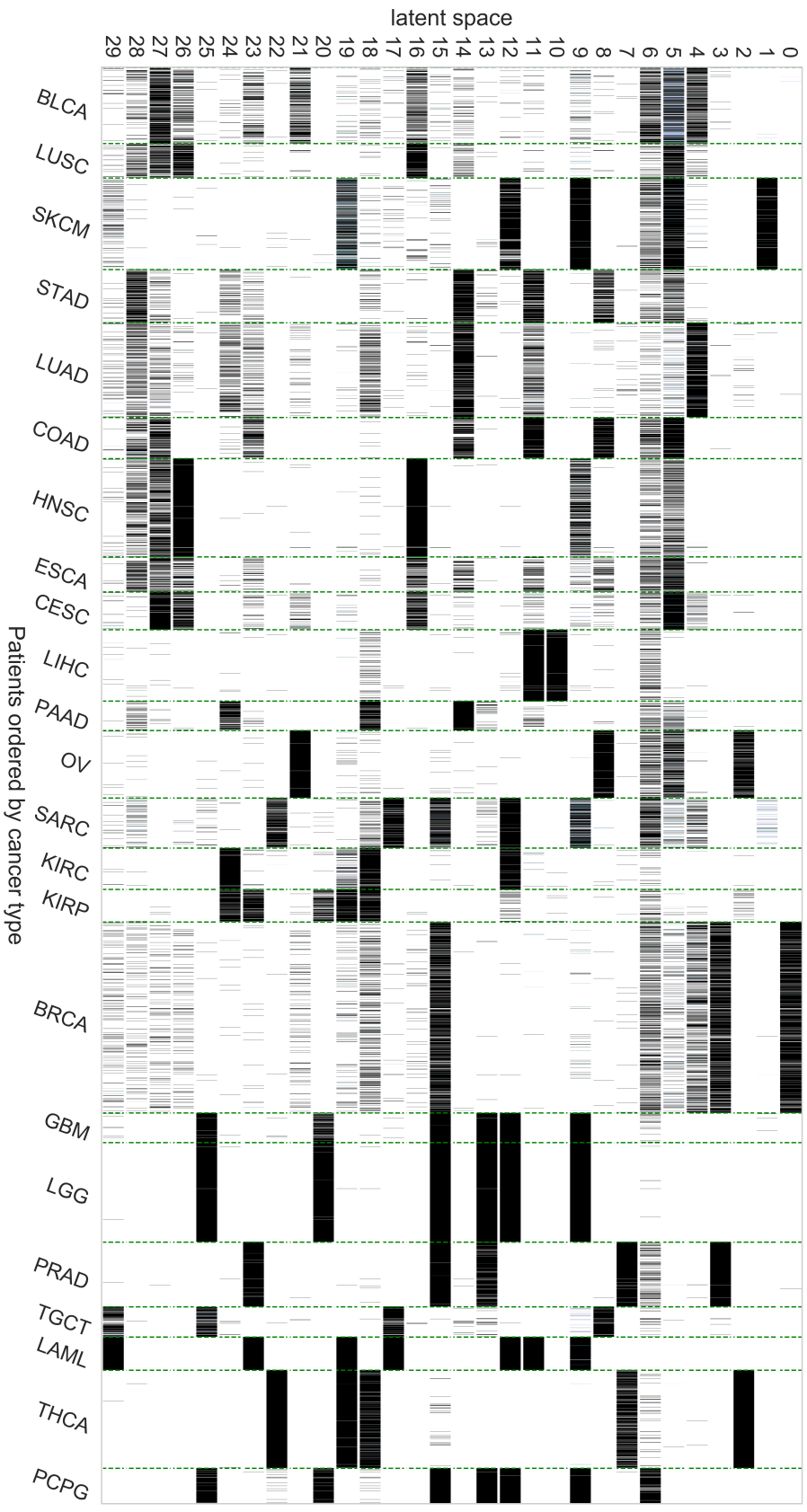


Fig. 3.8 **Representations of cancer patients** – Each column corresponds to one out of approximately 8,000 cancer patients and indicates which of the latent properties of relative expression among approximately 2,000 genes they exhibit. Patients are ordered by type of cancer. See Figure 1 in the supplementary material for the corresponding representation from PCA on the continuous expression data, as well as for a legend of the disease types.

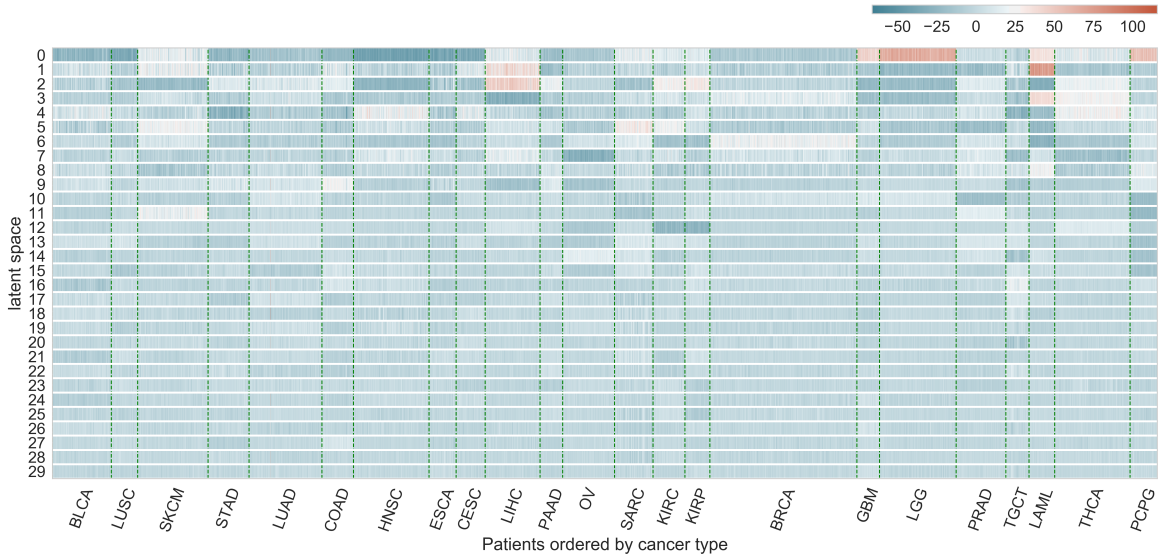


Fig. 3.9 Latent representation of the continuous patient – gene-expression array under principal component analysis and arranged in analogy to Fig. 3.8 on page 76.

dom forest classifiers. The predictive accuracy of the disease type is approximately 91% for the binary TensOrMachine features and only 86% for continuous features from PCA.

Results are indicated in Tab. 3.1 and show that we have, without supervision, recovered the appropriate tissue-specific phenotypes. We highlight a few examples on the following. Firstly, Hippo signalling has been shown to be active in LUAD, LUSC, COAD, OV and LIHC (Harvey et al., 2013), all recovered by dimensions 4 and 5. Secondly, aberrant Wnt signalling is observed in many cancers but most prominently in COAD (Polakis, 2012) which is found in dimension 5. Next, PRAD progression is known to be controlled by focal adhesion (Figel and Gelman, 2011) as found in dimension 15 as well as by the AGE-RAGE signalling pathway recovered in dimension 7 (Bao et al., 2015). Finally, Ras signalling is aberrant in most tumours, but most significantly in PAAD (Downward, 2003) as found in dimensions 18 and 24.

Table 3.1 Gene-set analysis of selected gene networks indicate their correspondence to various regulatory and signalling pathways. The (expr.) sign indicates whether genes in the set were relatively over/under expressed. The corresponding set of genes in relation to which the over/under expression occurs is not taken into account for this traditional analysis. Cancer types are included if the latent feature is active in more than 50% of cases.

Dim./ (Expr.)	Cancer types	Pathways
1 (-)	SKCM	Tight juntions
4 (+)	LUAD, BLCA	Hippo*
5 (+)	CESC, COAD, SKCM, LUSC, ESCA, OV	Hippo, Wnt*
7 (+)	PRAD, THCA	AGE-RAGE*
9 (+)	LGG, PCPG, LAML, GBM, SKCM, HNSC	Rap1 signalling
11 (+)	LAML, LIHC, COAD, STAD	Actin cytoskeleton
15 (+)	PCPG, LGG, PRAD, GBM, BRCA, SARC	Focal adhesion*
16 (-)	HNSC, LUSC, CESC, ESCA	PI3K-Akt
17 (+)	LAML, SARC, TGCT	Platelet activation
18 (+)	KIRC, KIRP, THCA, PAAD	Ras*
24 (+)	KIRC, KIRP, PAAD	Ras*

3.4.4 Multiple-Tissue Gene Expression

The concluding example of this chapter investigates a single cell dataset from The Tabula Muris Consortium et al. (2017). It characterises the tissue-dependent transcriptome and consists of gene expression measurements of roughly 23,000 genes in roughly 54,000 cells taken from 10 different mice across 18 different tissues. A particular challenge here is that cells are not available for all mouse/tissue pair. For every mouse, we consider only tissues with data from more than 50 cells, which leaves us with data for 104 out of the 180 mouse/tissue pairs. We thus have a $10 \times 18 \times 23,000$ mouse-tissue-gene matrix, where we consider a triplet as one if the gene is expressed in more than 20% of the cells of the corresponding mouse-tissue

pair. In two cases it was not clear from which of two mice the cells originated. Around 42% of data-points in the resulting tensor are unobserved, around 12% are one.

The decomposition with 30 latent dimensions is shown in Fig. 3.10 for the tissue and gene factors. We also infer sets of genes that are expressed across sets of tissues and sets

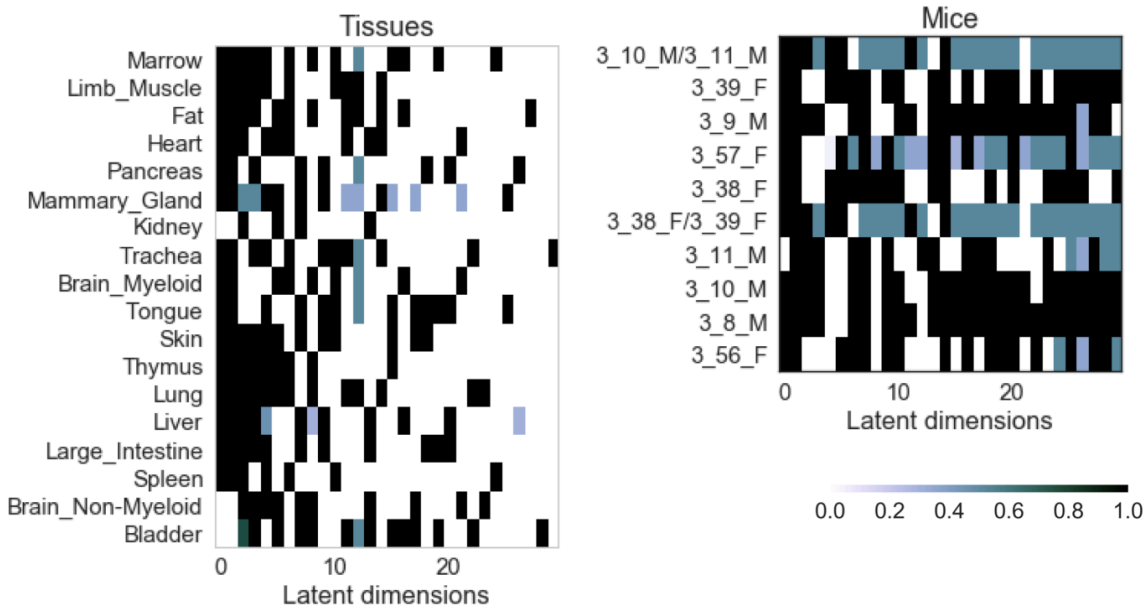


Fig. 3.10 Mouse-tissue-gene decomposition, derived from single cell sequencing data in 54.000 cells from 10 different mice across 18 different tissues and 23.000 genes.

of mice. Visual depiction of the 23.000 dimensional gene sets is clearly not useful and, instead, we conduct a gene-set enrichment analysis based on the mouse-gene-atlas (Chen et al., 2013; Su et al., 2004). The results in Table 3.2 indicate all enriched states with an adjusted p-value smaller than 0.05 alongside the corresponding tissue-sets. We observe a remarkable correspondence of the tissue types and the enriched states. Here, we point out two of the many striking examples. Firstly, dimension 26, active in pancreas tissue has a highly significant association to *min6*, which is a pancreatic cell line. Secondly, marrow and spleen tissue in dimension 24, which play important roles in immune cell production and maintenance, correspond to a gene-set that is identified with B-cells, and less significantly with natural killer cells and the spleen itself.

Table 3.2 Enriched phenotypes from gene set enrichment analysis alongside the tissues from which the cells originated. Latent dimensions correspond to Fig. 3.10.

latent dimension	**tissue**	**enriched phenotypes (-log 10 of adjusted p-value)**
29	Trachea	osteoblast_day21 (12) osteoblast_day14 (10) mouse embryonic fibroblast (3)
28	Bladder	bladder (16) osteoblast_day21 (8) osteoblast_day14 (6)
26	Pancreas	min6 (pancreas cell line) (44) pituitary (4) hypothalamus (2)
25	Mammary_Gland, Tongue	epidermis (5)
24	Marrow, Spleen	follicular_B-cells (53) NK_cells (1) spleen (1)
23	Lung, Brain_Non-Myeloid	lung (16)
20	Pancreas, Tongue, Liver, Large_Intestine	adipose_brown (5) liver (4)
19	Marrow, Tongue, Large_Intestine, Bladder	mega_erythrocyte_progenitor (1)
15	Marrow, Tongue, Skin, Thymus, Bladder	thymocyte_DP_CD4+CD8+ (4)
14	Limb_Muscle, Fat, Heart, Mammary_Gland, Trachea, Lung	MEF (3) osteoblast_day21 (3) osteoblast_day14 (2)
13	Heart, Kidney, Liver, Large_Intestine, Brain_Non-Myeloid, Bladder	kidney (2)
12	Limb_Muscle, Fat, Lung	osteoblast_day21 (5) MEF (5) lung (5)
6	Marrow, Limb_Muscle, Fat, Heart, Trachea, Brain_Myeloid, Thymus, Lung, Spleen	follicular_B-cells (4) mast_cells (1)

3.5 Conclusion

We have introduced the first probabilistic approach to Boolean tensor decomposition. Similar to the OrMachine for Boolean matrix factorisation in Chapter 2, it clearly outperforms the previously available methods in accuracy, deals with missing data, scales to large datasets and provides full uncertainty quantification. We have introduced scalable approaches to problem of selecting the appropriate latent dimensionality. Further, we demonstrated that Boolean tensor decomposition leads to insightful latent representations in several diverse real-world examples. Moreover, it enables an entirely novel view on the molecular basis of cancer by relationally encoding continuous expression data.

Chapter 4

Logical Factorisation Machines

Based on the logical operations that are used in defining their likelihoods the models described in the previous chapters can be seen as OR-AND-machines. This chapter describes their generalisation to a broaders class of logical operators, including XOR, operator negations and arbitrary pairs of operators. We detail this generalisation to *Logical Factorisation Machines* in Section 4.1 and describe conversion between and equivalence of different machines in Section 4.2. This lays the ground for a taxonomy of LFMs introduced in Section 4.3. In the following we detail general properties, factor densities, factor conditionals and probabilistic data reconstruction in sections 4.4, 4.5 and 4.6, respectively. Based on this exploration we can show that many of the proposed models do not satisfy some of our basic desiderata and focus our effort on the more revelant cases. A central result of this chapter is given in eq. (4.32), where we provide a general expression for the full conditional distribution of any Logical Factorisation Machine. Section 4.7 discusses the peculiarities of models that include XOR operations, before we treat the problem of model selection in Section 4.9. Section 4.10 showcases advanced use of our implementation including general LFMs, model hierarchies and prior distributions. Section 4.11 provides concluding remarks.

4.1 Generalisation of Logical Operators

In the previous chapters, we have discussed models that generate data following the Boolean product,

$$y_{[\mathbf{n}]} = \underset{l}{\texttt{OR}} \left[\underset{n_k \in [\mathbf{n}]}{\texttt{AND}} \left(f_{n_k l} \right) \right] \ . \tag{4.1}$$

We have proposed a likelihood that is conveniently parameterised by the logistic sigmoid with bit-flip noise controlled by $\lambda \in \mathbb{R}^+$, such that

$$p(x_{[\mathbf{n}]}|.) = \sigma \left[\lambda \tilde{x}_{[\mathbf{n}]} \tilde{y}_{[\mathbf{n}]} \right] \ . \tag{4.2}$$

Here $\tilde{x}, \tilde{y} \in \{-1, 1\}$ and $[\mathbf{n}] = [n_1, \ldots, n_K]$ denotes indices for a single entry in the data matrix or tensor. Further details of the notation have been introduced in Chapter 2. The overarching goal was to discover interpretable binary latent factors that have reasonable correspondence to the real-world data-generating processes. However, considering eq. (4.1), there is no obvious *a priori* reason for natural processes to follow the `OR-AND` logic. Instead, we consider a generalisation where we replace the `OR` and `AND` operations by any logical operator as illustrated in Fig. 4.1. We call these models *Logical Factorisation Machines* (LFMs). More formally, eq. (4.1) becomes

$$y_{[\mathbf{n}]} = \underset{l}{\texttt{LOP}_o} \left[\underset{n_k \in [\mathbf{n}]}{\texttt{LOP}_i} \left(f_{n_k l} \right) \right] \ , \tag{4.3}$$

where $\texttt{LOP}_i$ is the inner logical operator that acts on the factors along the K tensor dimensions and produces and intermediate binary vector of length L on which the outer operator $\texttt{LOP}_o$ acts to generate the deterministic output $y_{[\mathbf{n}]}$.

We consider six logical operators: `OR`, `AND`, `XOR` and their respective negations. Their ordinary definitions are extended to accommodate not only for two but for any larger number

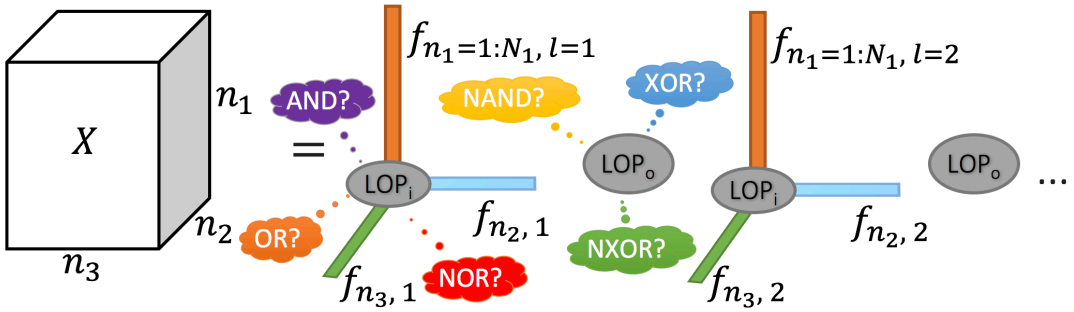


Fig. 4.1 Graphical illustration of the generative procedure for Logical Factorisation Machines. The inner, $\texttt{LOP}_i$ operator generates a rank-1 tensor for every latent dimensions. These rank-1 tensors are combined element-wise by the outer operator $\texttt{LOP}_o$

of binary arguments. For a vector of arguments $f \in \{0,1\}^L$ we define:

$$\underset{l}{\texttt{AND}}\,[f] = \prod_l f_l \tag{4.4}$$

$$\underset{l}{\texttt{OR}}\,[f] = 1 - \prod_l \left(1 - f_l\right) \tag{4.5}$$

$$\underset{l}{\texttt{XOR}}\,[f] = \mathcal{I}\left[\sum_l f_l = 1\right] \tag{4.6}$$

Here $\mathcal{I}$ is the indicator function that evaluates to 1 if its argument is true and to zero otherwise. Note, that the definition of XOR for an arbitrary number of input arguments goes beyond the customary usage.

These LOPs can be seen as subset of the general class of Boolean functions, that map inputs of Boolean variables to a single Boolean variable. For an L-dimensional input, there exist 2^{2^L} such functions (Wegener, 1987). A more restricted group of functions is the subset of *R-out-of-L* functions that are invariant under permutations of the input and that can be defined as

$$\text{R-out-of-L}(f) = \mathcal{I}\left[\sum_{l=1}^{L} f_l = R\right]. \tag{4.7}$$

The logical operators we propose are a yet more specific subgroup of these functions. They can be identified as *All-out-of-L* (AND), *1-out-of-L* (XOR) and *None-out-of-L* (NOR) and their respective negations. We leave the generalisation beyond these functions for future work. In a separate publication (Rukat et al., 2017b), we have explored yet another variety of the model, replacing the outer operator, $\texttt{LOP}_o$ by a MAX-operator and introducing a different noise parameter for every latent dimension.

Similar generalisations have previously been explored in the context of knowledge engineering. This includes generalisation of the noisy-OR to noisy-AND, noisy-XOR and noisy-MAX models (Dıez and Druzdzel, 2006; Srinivas, 2013; Vomlel, 2015). Heller and Ghahramani (2007) have devised a Bayesian nonparametric model that can be interpreted as noisy-NOR. However, to our best of knowledge, none of the previous approaches has addressed the problem of recovering different data-generating logics in the context of latent factor models. If possible, this would allow us to go beyond inferring the latent variables of a prescribed mechanism, but to infer the mechanism of latent interactions itself. To give a simplistic example, in the context of genomics data, we would be interested in inferring whether all members of a set of genetic variants are necessary to induce a clinical outcome or whether any particular variant is sufficient.

4.2 Conversion between Logical Factor Machines

With six logical operators and the two-step generating procedure, there are 36 possible LFMs. In this section we will discuss how these machines can be converted into each other by applying inversion operations at different steps of the generative procedure. We develop a hierarchy of LFMs that groups models with similar properties, helps us to develop a unified inference scheme and shows which LFM is useful in any given situation.

The negation operator, $\neg$, in $\{0,1\}$ representations is defined as

$$\neg f = 1 - f \ , \tag{4.8}$$

and can be used to establish a set of simple set conversions. Firstly, negation of the output of an operator is simple:

$$\neg\texttt{AND}\,[f] = \texttt{NAND}\,[f] \ , \quad \neg\texttt{OR}\,[f] = \texttt{NOR}\,[f] \ , \quad \neg\,\texttt{XOR}\,[f] = \texttt{NXOR}\,[f] \ . \tag{4.9}$$

More interestingly, negation of the input yields the following identities:

$$\texttt{AND}\,[\neg f] = \texttt{NOR}\,[f] \tag{4.10}$$

$$\texttt{OR}\,[\neg f] = \texttt{NAND}\,[f] \tag{4.11}$$

These can be combined to yield identities such as $\neg\texttt{AND}\,[\neg f] = \texttt{OR}\,[f]$. The set of six logical operators is closed under negations, with exception of input negation for XOR when the number of inputs is greater then two. The resulting $\texttt{XOR}\,[\neg f]$ equals an *(L-1)-out-of-L*-operation that is not in the set of logical operators.

Negations can be applied at the three different points during the generative procedure of LFMs as given in eq. (4.3). They can act on the input, the intermediate and the output stage:

$$y_{[\mathbf{n}]} = \overset{o}{\neg}\, \underset{l}{\texttt{LOP}_o} \overset{h}{\neg}\, \underset{n_k}{\texttt{LOP}_i} \overset{i}{\neg}\, f_{n_k l} \tag{4.12}$$

Here $\overset{i}{\neg}$, $\overset{h}{\neg}$ and $\overset{o}{\neg}$ stand for input, hidden and output negation, respectively. Conversion between LFMs proceeds via assignments $\overset{i,h,o}{\neg} \in \{\mathbb{1}, \neg\}$, where $\mathbb{1}$ is the identity. To give an example, we start from an OR-AND-machine, where $\texttt{LOP}_o = \texttt{OR}$ and $\texttt{LOP}_i = \texttt{AND}$. We apply negation to the hidden and to the input state, that is $\overset{o}{\neg} = \mathbb{1}$, $\overset{h}{\neg} = \neg$, $\overset{i}{\neg} = \neg$. Plugging these

assignments into eq. (4.12) and suppressing indices to avoid clutter we have

$$y_{[\mathbf{n}]} = \underset{l}{\texttt{OR}}\, \neg\, \underset{n_k}{\texttt{AND}}\, \neg\, f_{n_k l} = (\texttt{OR}\, \neg)\, (\texttt{AND}\, \neg)\, f = \texttt{NAND}\, \texttt{NOR}\, f \tag{4.13}$$

We find that inverting the hidden state and the input state of an OR-AND-machine results in an NAND-NOR-machine. Alternatively, we can apply both negations to the inner logical operator (compare to the parenthesis in eq. (4.13)), such that

$$y_{[\mathbf{n}]} = \texttt{OR}\, \neg\, \texttt{AND}\, \neg\, f = \texttt{OR}\, (\neg\, \texttt{AND}\, \neg)\, f = \texttt{OR}\, \texttt{OR}\, f\ . \tag{4.14}$$

This demonstrates equality, $\texttt{OR}\, \texttt{OR} = \texttt{NAND}\, \texttt{NOR}$. Twelve such equalities exist among the 36 LFMs, reducing the number of different models to 24. Equalities can easily be derived, multiplying the hidden state by the identity, $\mathbb{1} = \neg\, \neg$, for instance

$$\texttt{OR}\, \texttt{AND} = \texttt{OR}\, \neg\, \neg\, \texttt{AND} = \texttt{NAND}\, \texttt{NAND}\ . \tag{4.15}$$

The conversion can similarly be applied to the explicit product-representation of the logical operators:

$$y_{[\mathbf{n}]} = \overset{o}{\neg} \prod_l \overset{h}{\neg} \prod_{n_k} \overset{i}{\neg} f_{n_k l}\ . \tag{4.16}$$

Here, $\overset{0}{\neg} = \overset{h}{\neg} = \overset{i}{\neg} = \mathbb{1}$ results in an AND-AND-machine. Boolean Tensor Factorisation, as defined in Chapter 2, can be recovered by application of $\overset{o}{\neg}$ and $\overset{h}{\neg}$:

$$y^{\texttt{OR-AND}}_{[\mathbf{n}]} = \neg \prod_l \neg \prod_{n_k} f_{n_k l} = 1 - \prod_l \left(1 - \prod_{n_k} f_{n_k l}\right) \tag{4.17}$$

4.3 A Taxonomy of Logical Factorisation Machines

Based on the previous discussion, we summarise the relationships of LFMs in a taxonomy. This will aid our discussion by identifying similarities between LFMs and avoids redundancies when developing a general framework for inference. It also allows to quickly draw connections between different LFMs that help in developing intuitions and recognizing model properties, e.g. by identifying a particular LFM as being closely related to a more intuitive or more well-studied model.

Note, that negations are sufficient to convert between AND and OR operators and their negations. In contrast, we can not convert to and from XOR and NXOR operators in the same manner. Hence, there exist four distinct *classes* of LFMs, based on AND-AND, on XOR-AND, on AND-XOR, and on XOR-XOR machines.

We discuss the taxonomy by example of AND-AND LFMs, devoid of XOR operations. Other classes of LFMs are discussed in the following sections. The taxonomy of the AND-AND class is shown in Fig. 4.2. Any LFM class consists of two *clans*, where conversion between the clans is achieved by application of an inversion to the intermediate state, $\overset{i}{\neg}$. Hence, output and factor inversion map to machines in the same clan. This arrangement is useful for the following reasons. First, posterior inference (discussed in Section 4.5) for models in the same clan follows the identical procedures. For instance inference for an OR-NOR-machine is identical to inference in an AND-NAND-machine, if the data is inverted before, and the inferred factors are inverted after the procedure. It follows, that machines in the same clan share most relevant properties and that a discussion of their different properties can usually be limited to the clan-level.

The taxonomy proceeds by distinction of clan-members into *families*. Conversion between families is achieved by inversion of the output, $\overset{o}{\neg}$. Hence, families are closed only under factor inversion (if applicable). Note, that even though data inversion is trivially achieved, the resulting models are fundamentally different due to the asymmetry in how

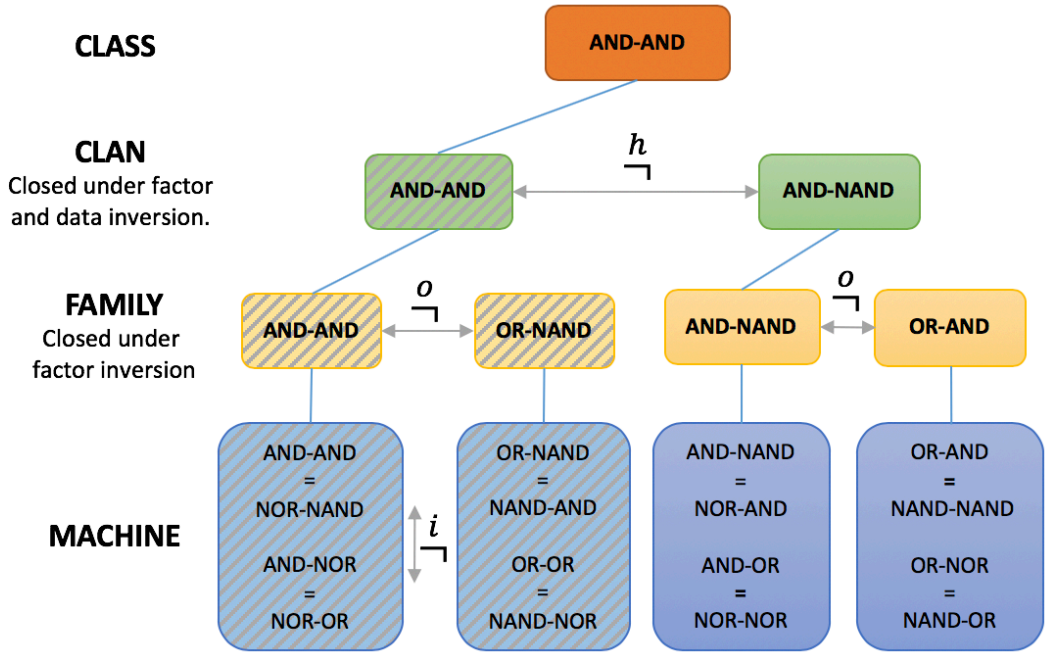


Fig. 4.2 Taxonomy of `AND-AND` based Logical Factorisation Machines. The arrows indicate how machines can be converted by application of the appropriate inversion operator. `AND-AND`-clan machines are scratched-through because of their impracticability as a factor model. See text for more details.

zeros and ones are treated. This is shown in an example for `OR-AND` and `NOR-AND` in Section 4.3.2.

Inference for machines in the same family is identical, except for subsequent inversion of the factors. Thus, machines in a family are effectively the same model and their difference comes into play, only when we care about the actual factor-semantics. The next sections discuss the two clans in more depth.

4.3.1 The `AND-AND` Clan

To begin with, consider a particular clan-member, the `OR-NAND`-machine. For a given latent dimension, a zero in any factor induces a *one* in the output of the `NAND`-operator. In the second step, the `OR` outputs a *one*, if any such latent dimension exists. This is exemplified in Fig. 4.3:

$\texttt{NAND}_{n_k} \downarrow$					
f_{n_1} :	1	0	1	…	
f_{n_2} :	1	1	1	…	
	0	1	0	…	$y_{n_1,n_2} = 1$
		$\overrightarrow{\texttt{OR}_l}$			

Fig. 4.3 Output of an OR-NAND-machine

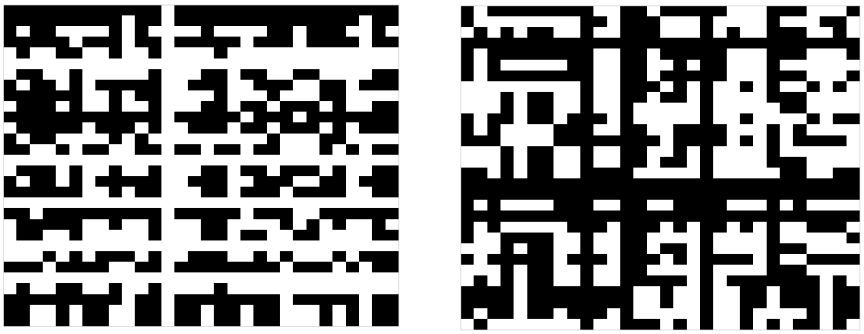
It is important to note that the *one* in the output is independent of the factor mode or the latent dimension that contains a *zero*, but rather occurs if a *zero* exists anywhere in $f_{n_1}, \ldots, f_{n_K}$. In general, if $f_{n_1,l=1:L}$ contains any zero, the output of all data points in the $K-1$-dimensional sub-tensor $\mathbf{Y}_{n_1,\{n_k=1:N_k\}_{k\neq 1}^{K}}$ is *one*. Because it is irrelevant in which latent dimensions a *zero* exists, the OR-NAND machine does not model latent correlations. All information that can be expressed in the factors, can be expressed in a single latent dimension. In particular, any OR-NAND-factorisation with more than one latent dimensions can equally be expressed by a single latent dimension.

All machines in the AND-AND-clan are related to the OR-NAND-machine via output or factor inversion. Thus, the same criticism applies and they are only ever useful for a single latent dimension. However, for a single latent dimension, an OR-NAND machine is equivalent to an AND-NAND machine and hence all $L = 1$ machines in the AND-AND-clan are equivalent to machines in the AND-NAND clan. As a result, AND-AND-clan machines can be considered entirely *impractical* as latent factor models.

4.3.2 The AND-NAND Clan

This clan includes the OR-AND-machine, the classical Boolean factorisation model as discussed in the previous chapters. The other contained family features the Boolean Factor model with inverted output, NOR-AND, which attempts to explain *zeros*, rather than the *ones* as disjunction of binary codes. Here, we can think of the latent dimensions as patterns of inhibition rather than activation.

Although output inversion is a simple process, NOR-AND constitutes distinctly different model. Data with a high density under the OR-AND logic may have a low density under NOR-AND logic and vice versa. To illustrate this Fig. 4.4 compares random draws from both machines. The most apparent difference are continuous rows/columns of *zeros* in the OR-AND product (Fig. 4.4a) and, conversely, of *ones* in the NOR-AND product (Fig. 4.4b). The attempt to reconstruct the NOR-AND generated data using an OR-AND-machine and vice versa, results in a training reconstruction accuracy of around 90%.



(a) Draw from an OR-AND machine (b) Draw from a NOR-AND machine

Fig. 4.4 Logical factor products of size 40×40 with expected density of 0.5, generated from two different pairs of Bernoulli i.i.d. random low-rank matrices with L=6 latent dimensions.

Selecting the appropriate machine is interesting in real-world settings. More specifically, inferring which logic is likely to have generated the data may provide novel insights into the underlying process, e.g. informs us about how different latent properties interact. This can lead to a broader understanding of the natural process that have generated the data and generate new hypotheses. Moreover, different factors for different logics provide a complimentary view of the data, as we demonstrate in the following real-world example, comparing OR-AND and NOR-AND machines. A more general view on the model selection problem will be taken in Section 4.9.

OR-AND-Machine versus NOR-AND-Machine

The single-cell gene expression dataset described by Pollen et al. (2014) consists of 301 cells of 9 known cell types. As for previous single-cell datasets, the number of sequencing reads per nucleotide is low, such that the essential information is presence and absence of gene expression rather than the expression count. Thus we consider binarised data as depicted, ordered by cell-type, in Fig. 4.5(a). To infer the appropriate latent dimensionality under OR-AND and NOR-AND logic, we optimise for the reconstruction accuracy on held-out data, treated as unobserved during training. In order to avoid a bias towards sparse models, the test-set is balanced, containing the same number of zeros and ones and consists of 20% of the data. The workflow is depicted in Fig. 4.5. NOR-AND achieves a test-set reconstruction accuracy of 82.5% with 10 latent dimensions, while the best OR-AND machine requires 11 latent dimensions to achieve a reconstruction accuracy of 77.5%[1]. The respective representations are shown in Fig. 4.5 (b).

Both representations feature a distinct type-specific structure. As an exception neurons in different gestational weeks (GW) are difficult to separate from each other and best distinguished from the remaining cells through the absence of expression (see, for instance, dimension 10 in the OR-AND representation and, conversely, dimensions 1 and 2 in the NOR-AND representation). We also observe heterogeneity across different cell-types, for example two types of pathologic immune cells that lack expression of a certain gene-set, given by EPV transformed lymphoblasts (2339) and leukocytes in leukemia (HL60) in dimension 3 of the NOR-AND representation.

It is apparent that both representations provide complementarity information. For instance OR-AND represents neural progenitor cells (NPC) in dimension 0, while there is no specific code for these cells in the NOR-AND representation. Conversely, dimension 1 in the NOR-AND representation corresponds mostly to cortical neurons in their earlier development

[1] For a test-set that represents the imbalance in the data, both percentages increase by around 5 points while their difference remains unchanged.

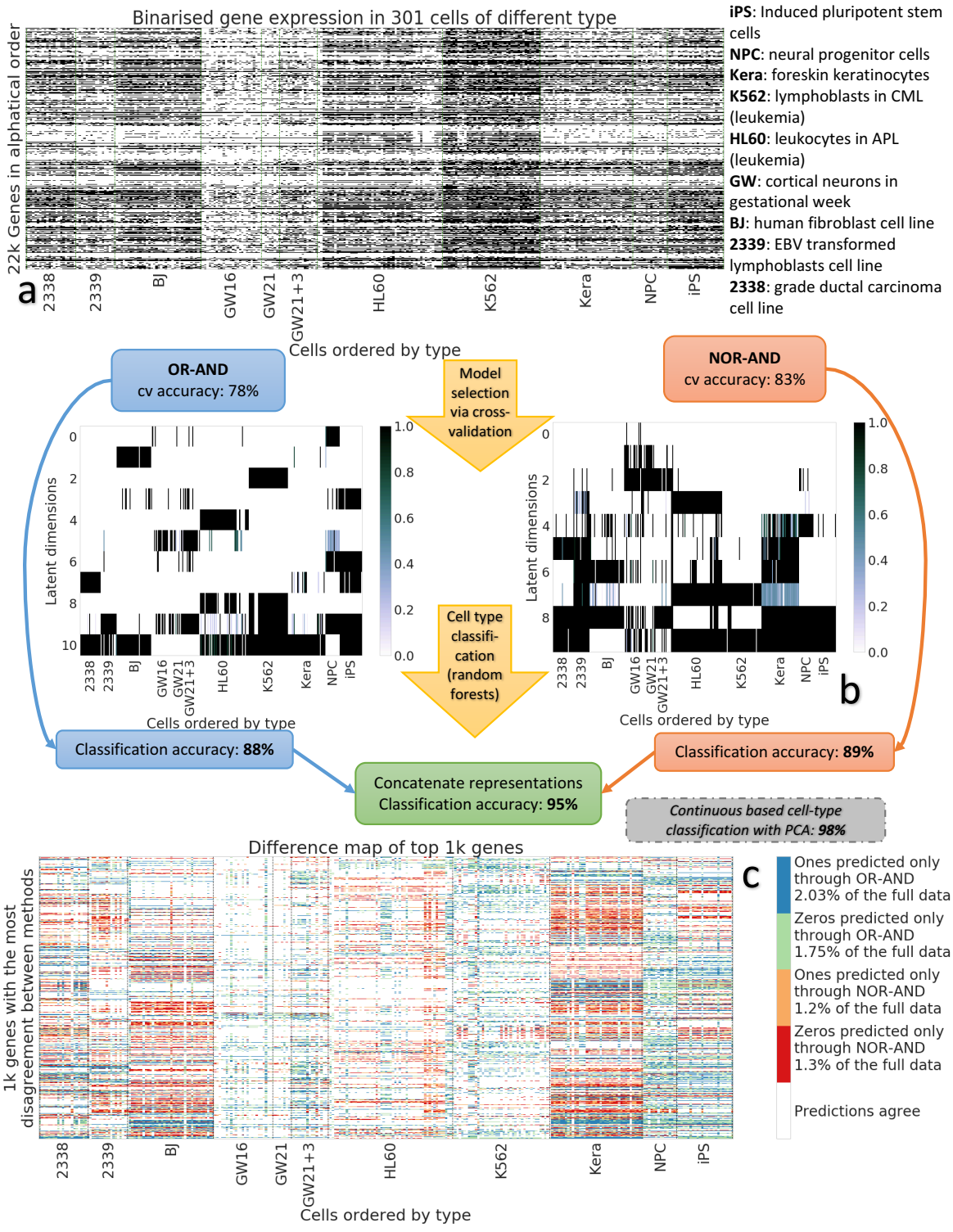


Fig. 4.5 Comparison of `OR-AND` and `NOR-AND` machines by example of binarised single-cell gene expression data with known cell types with data shown in (a). Posterior means of the compressed representation are shown in (b). Classification percentages are from random forest based cell type classification and measured on a validation set. Fig. (c) indicates location and type of disagreement between the machine's posterior predictive distribution for the 500 genes with most disagreement.

stages (*GW16, GW21*), which lack an exclusive representation under OR-AND logic. We substantiate the observation that both models discover different, significant patterns in the data by training a random forest classifier for cell-type prediction. After extensive hyper-parameter optimisation with a 50/25/25 train/test/validate split, the representations achieve a predictive accuracy of 88% (OR-AND) and 89% (NOR-AND) on the validation set. Combining both representations, that is concatenating the posterior factor means for each cell, leads to a classification accuracy of 95%. Clearly, OR-AND and NOR-AND pick up distinct and meaningful characteristics of the dataset.

We further inspect the difference in the MAP predictive in Fig. 4.5 (c). It indicates patterns of disagreement between both models for the 500 genes where the most disagreement is observed. We observe columns with noticeably larger rates of disagreement, corresponding to cells with a prominently different predictive under the two models. Similarly, we observe row-segments for particular genes that span a certain cell-type, indicating disagreement about expression of a certain gene. Some of these segments are predicted correctly by one of the two models, throughout. Thus, they indicate gene/cell-type pairs whose activity is well capture by one of the models, presumably contributing to their complementarity. However, many of the prominent row and column segments do not favour any particular model and are not amenable to complete explanation under any of the two factorisations.

4.4 Expected Factor Densities for LFMs

One important property of LFMs is the relationship of data and factor density. In the following, we derive general expressions for the expected data density, conditional on the density of all factors. We assume the factor density is constant across factors, that is $\rho(f) = \hat{f}$ for all f_k and that the factor entries are i.i.d. random. Based on this assumption we state

expressions for the expected density of the data $\rho(X)$. For AND-AND derived LOMs we have

$$\rho(Y) = \overset{o}{\neg}\left[\overset{i}{\neg}\left(\overset{f}{\neg}\hat{f}\right)^{K}\right]^{L}. \tag{4.18}$$

For an AND-AND machine, none of the inversions apply and we have $\rho(Y) = \hat{f}^{K^L}$, stating that an output is only *one*, if all factors in all dimensions are *one*. For $\overset{o}{\neg} = \overset{h}{\neg} = \neg$ and $\overset{i}{\neg} = 1$, we recover the OrMachine density,

$$\rho(Y) = 1 - \left[1 - \hat{f}^{K}\right]^{L}. \tag{4.19}$$

Based on analogous arguments we can derive expected densities for XOR-AND based LFMs

$$\rho(Y) = \overset{o}{\neg}\left[\left(\neg\overset{i}{\neg}\left(\overset{f}{\neg}\hat{f}\right)^{K}\right)^{L-1}\overset{i}{\neg}\left(\overset{f}{\neg}\hat{f}\right)^{K}\right], \tag{4.20}$$

for AND-XOR based LFMs

$$\rho(Y) = \overset{o}{\neg}\left[\overset{i}{\neg}\left(K\hat{f}\cdot\left(\neg\hat{f}\right)^{K-1}\right)\right]^{L} \tag{4.21}$$

and for XOR-XOR derived LFMs

$$\rho(Y) = \overset{o}{\neg}\left(LK\overset{i}{\neg}(\hat{f})\left(\overset{i}{\neg}\neg\hat{f}\right)^{K-1}\left[1 - K\left(\overset{i}{\neg}\hat{f}\right)\left(\overset{i}{\neg}\neg\hat{f}\right)^{K-1}\right]^{L-1}\right). \tag{4.22}$$

The inverse problem of finding the expected factor density given the data density lacks a closed form solution in many cases. If needed, we resort to a numerical strategy. The results are illustrated in Fig. 4.6 which shows the expected density of data that is genereated from i.i.d. factors with density varying on the x-axis. We compare a variety of latent dimensionalities (L) and different arities (K) for all LFM clans. Conversion to different LFMs

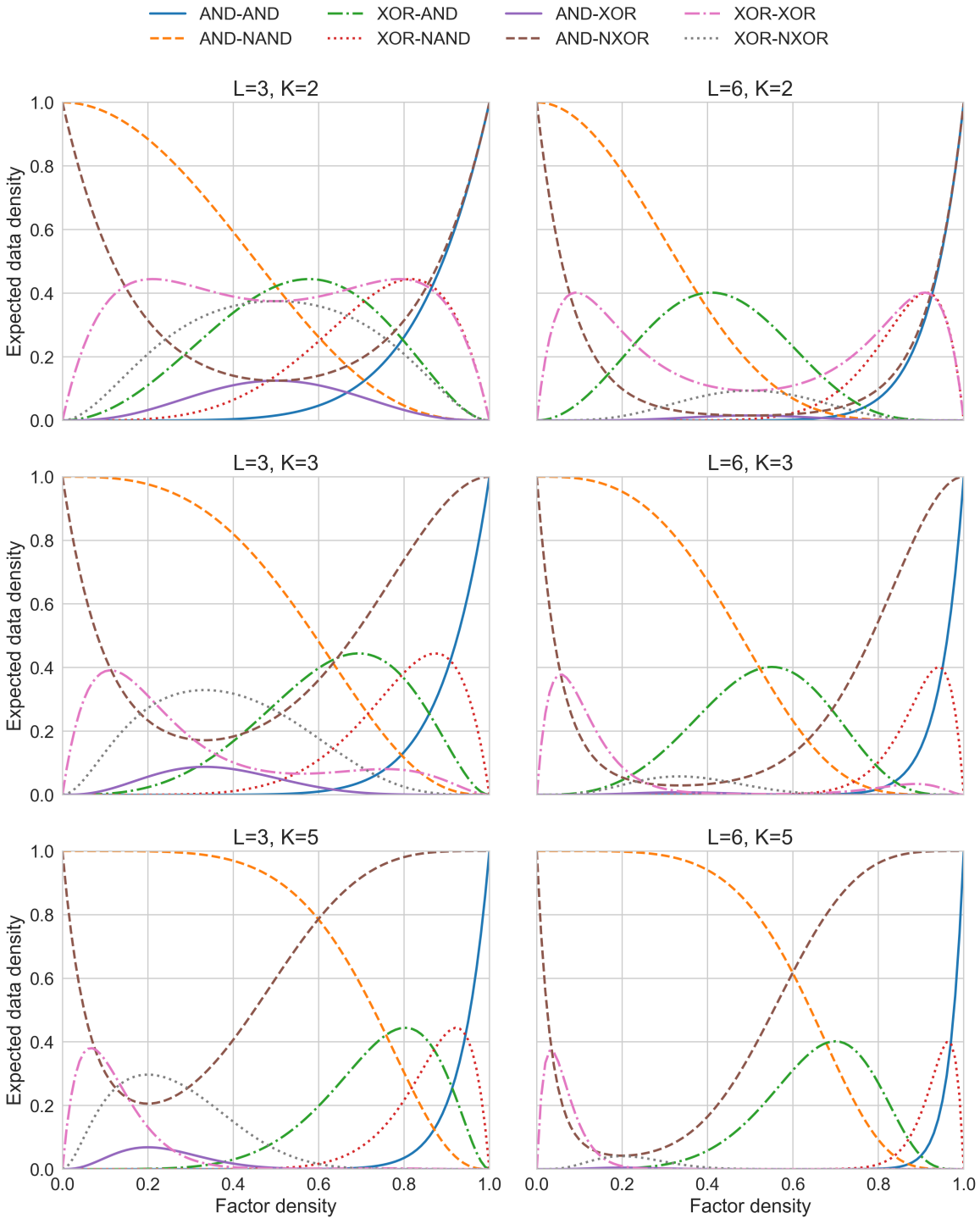


Fig. 4.6 Expected density for data generated from the 8 clans of logical factor machines.

amounts to inverting the expected density graphs. Inverting the vertical axis is equivalent to application of $\overset{o}{\neg}$ and inverting the horizontal axis is equivalent to application of $\overset{f}{\neg}$.

From the perspective of constituting a practical latent factor model, it seems desirable for the density functions to have two properties:

1. Be monotonic with respect to the factor density, such that denser factors consistently lead to denser or sparser data. While this may not be necessary, interpretation of the latent factors in a non-monotonic model would not be as intuitive.

2. Span the whole range of expected data densities, such that there is support for data of any density.

Only machines in the AND-AND class meet these two requirements and none of the other machines meets any of them. This casts some general doubt on the applicability of machines containing XOR-operations. We will discuss this supposition in more detail in the following sections.

4.5 Factor Conditionals

Inference for LFMs proceeds analogous to the TensOrMachine, which is an `OR-AND`-machine, as described in the previous chapter. We merely need to find a general expression for the factor conditionals,

$$p(z_{nl}|.) = \sigma\left[\lambda \tilde{z}_{nl} g_{nl}\right] \ , \tag{4.23}$$

where for the `AND-AND`-machine we have

$$g_{nl} = \sum_{k \in [k]\backslash n}^{K_k} \tilde{x}_{nd} \left[\prod_{[k]\backslash n} f_{kl}\right] \left[\prod_{l' \neq l} \prod_{[k]} f_{kl'}\right] \ . \tag{4.24}$$

An elegant way to explicitly write LFM conditionals makes use of the difference operators. This operator describes the change in the output of a LOP when adding a 1 to the input compared to adding a 0 to the input. The following relationships only hold in {0,1} representation. The difference operator is defined as

$$\delta\mathtt{LOP}(f) = \mathtt{LOP}(f,1) - \mathtt{LOP}(f,0)\ . \tag{4.25}$$

Its effect on different LOPs is as follows:

$$\delta\mathtt{AND}(f) = \mathtt{AND}(f,1) - \mathtt{AND}(f,0) = \mathtt{AND}(f) - 0 = \mathtt{AND}(f) \tag{4.26}$$

$$\delta\mathtt{OR} = \mathtt{OR}(f,1) - \mathtt{OR}(f,0) = 1 - \mathtt{OR}(f) = \mathtt{NOR}(f) \tag{4.27}$$

It follows from the definition in eq. (4.25) that $\delta\neg\,\mathtt{LOP} = -\delta\mathtt{LOP}$. Hence, we have

$$\delta\mathtt{NOR} = -\delta\mathtt{OR} = -\mathtt{NOR} \tag{4.28}$$

$$\delta\mathtt{NAND} = -\delta\mathtt{AND} = -\mathtt{AND}\ . \tag{4.29}$$

For XOR operators we have

$$\delta\mathtt{XOR}(f) = \begin{cases} 0 \text{ if} & \sum_k f_k > 1 \\ 1 \text{ if} & \sum_k f_k = 0 \\ -1 \text{ if} & \sum_k f_k = 1 \end{cases} \tag{4.30}$$

and

$$\delta\texttt{NXOR}(f) = -\delta\texttt{XOR} = \begin{cases} 0 \text{ if} & \sum_k f_k > 1 \\ -1 \text{ if} & \sum_k f_k = 0 \\ 1 \text{ if} & \sum_k f_k = 1 \ . \end{cases} \tag{4.31}$$

In the general expression for the conditional in eq. (4.32), for any $\texttt{LOP}_o$-$\texttt{LOP}_i$-machine, g_{nl} is given by

$$\boxed{p(z_{nl}|.) = \sigma\left[\lambda\tilde{z}_{nl}\sum_d \tilde{x}_{nd}\ \underset{k\neq n}{\delta\texttt{LOP}_i(f_{ld})}\ \underset{l'\neq l}{\delta\texttt{LOP}_o}\ \underset{k}{\texttt{LOP}_i(f_{kl'})}\right]\ .} \tag{4.32}$$

For instance the conditional in an OR-AND-machine is

$$g_{nl} = \sum_d \tilde{x}_{nd}\ \texttt{AND}(f_{lk})\ \texttt{NOR}_{l'}\texttt{AND}_k(f_{kl'}) \tag{4.33}$$

whereas the conditional in a XOR-AND-machine can be written as

$$g_{nl} = \sum_d \tilde{x}_{nd}\ \texttt{AND}(f_{lk})\ \delta\texttt{XOR}_{l'}\texttt{AND}_k(f_{kl'}) \tag{4.34}$$

Expressions for the remaining machines are given in Appendix D.1.

4.6 Data Reconstruction

Following this inference procedure, we can impute missing data based on a Monte Carlo estimate of the predictive distribution of some unobserved data-point. In previous chapters, we have instead used the mean-field reconstruction $p(x_{nd}|\hat{U}, \hat{Z}, \hat{\lambda})$ as a fast and accurate approximation of the predictive distribution. This amounts to plugging in the posterior mean

estimates for f_{nk} into the respective expressions. We provide these expressions for all LFMs in Appendix D.2.

Next, we discuss further general properties of XOR-containing machines before we apply the general inference and reconstruction schemes in practice.

4.7 Machines Containing XOR-Operations

4.7.1 The XOR-AND Class

The taxonomy for XOR-AND-class machines, in analogy to AND-AND-class machines in Section 4.3, is shown in Fig. 4.8a. In contrast to AND-AND-class machines, inversion of the inner state, $\overset{i}{\neg}$, is unambiguous. This follows from the observation that inverting the input of an XOR-operation is not well-defined in the context of LFMs. Therefore the inversion acts on the output of the respective inner operator and there are eight unique machines in the class.

The XOR-AND-Clan

These machines can be interpreted in analogy to the OR-AND, that is Boolean Factorisation machines. Instead of requiring any latent dimension where all factor modes are *one*, they require exactly one such latent dimension. This can lead to interesting sparse representations. However, note that with a single latent dimension that has *ones* everywhere XOR-AND becomes an $L-1$ dimensional NOR-AND machine as shown in Fig. 4.7.

$$\begin{array}{l} \mathtt{AND}_{n_k} \downarrow \\ \hline \\ \\ \\ \underset{\longrightarrow}{\mathtt{XOR}_l} \end{array} \begin{array}{cccc} 1 & \mathrm{a} & \mathrm{b} & \dots \\ 1 & \mathrm{c} & \mathrm{d} & \dots \\ \hline 1 & \mathrm{e} & \mathrm{f} & \dots \end{array} \quad = \quad \begin{array}{l} \mathtt{AND}_{n_k} \downarrow \\ \hline \\ \\ \\ \underset{\longrightarrow}{\mathtt{NOR}_l} \end{array} \begin{array}{ccc} \mathrm{a} & \mathrm{b} & \dots \\ \mathrm{c} & \mathrm{d} & \dots \\ \hline \mathrm{e} & \mathrm{f} & \dots \end{array}$$

Fig. 4.7 XOR-AND machines encompass L-1 dimensional NOR-AND machines.

We find empirically, in particular for more than around 5 latent dimensions, the machine learns solutions which make use of this property. That is, rather than ones that are explained by exclusive latent causes, we see zeros that are explained by multiple causes.

The XOR-NAND Clan

With XOR-AND being analogous to OR-AND, XOR-NAND-clan machines are to some extent analogous to OR-NAND machines. However, the argument for their impracticability is more subtle. In an XOR-NAND-machine exactly one latent dimension that contains a *zero* makes the output *one*. Existence of a second such dimensions makes the output *zero*. Thus, we only need to consider the number of latent dimensions with a zero in any factor mode, their exact location is irrelevant. Any XOR-NAND factorisation with more than two latent dimensions can be equally be expressed with merely two latent dimensions.

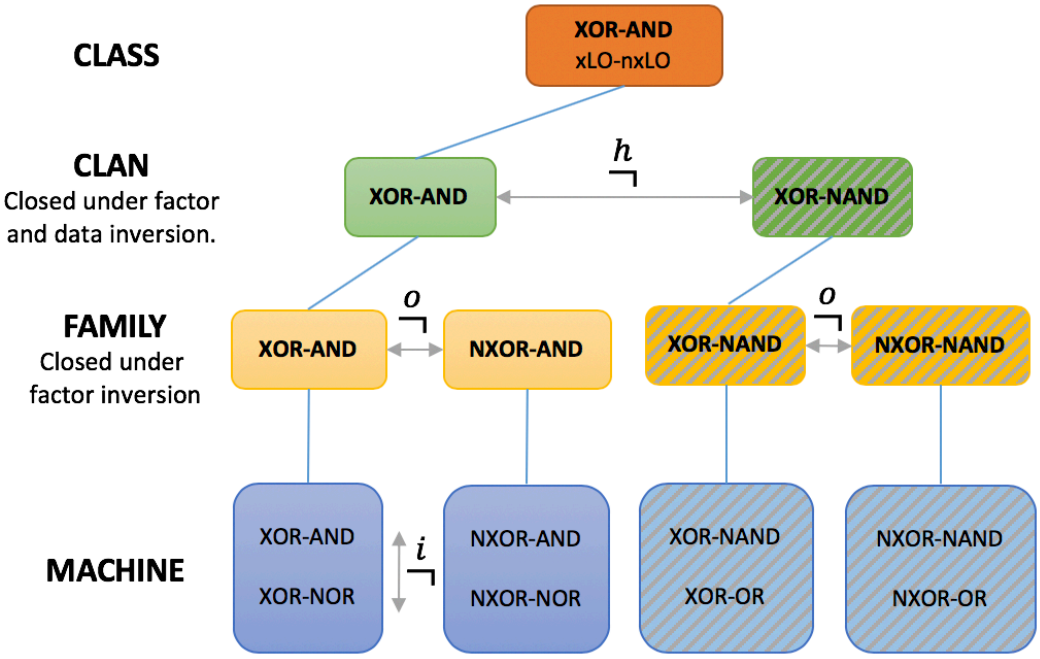
Note that for a L=1, XOR-NAND machines are equivalent to AND-NAND machines. Furthermore, for K=2 and L=2, XOR-NAND is equivalent to XOR-AND. Thus XOR-AND-clan machines have a limited but unique and non-trivial ability to model dependencies only for L=2 and K>2. Overall, machines in the XOR-AND class are *impractical* for most applications.

4.7.2 The AND-XOR-Class

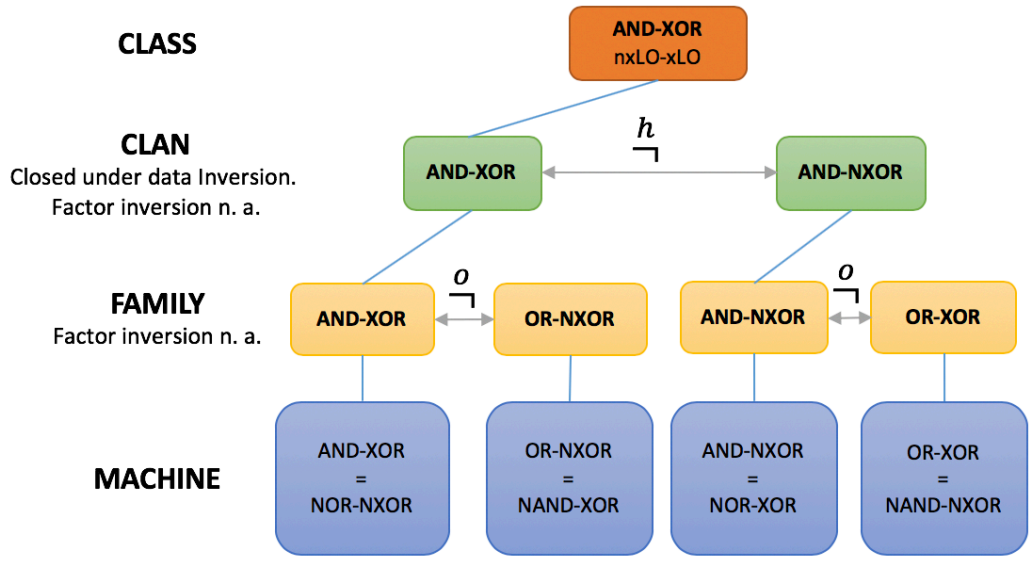
The taxonomy for AND-XOR-class machines is shown in Fig. 4.8b. Since input inversion is not defined for XOR-operators, machines in the same family are identical, and we have merely four unique machines.

AND-XOR-Clan

OR-NXOR machines are the member of the AND-XOR-clan that is most straightforward to interpret. These machines can be thought of in analogy to the OR-AND-machine. For the important case of K=2, this is particularly evident. While in OR-AND-machines, the output is *one*,



(a) XOR-AND based Logical Factorisation Machines.



(b) AND-XOR based Logical Factorisation Machines.

Fig. 4.8 Logical Factorisation Machines containing a single XOR-operation.

if there exists any latent dimension where both factor modes are *one*, OR-NXOR-machines output *one* if there exists any latent dimension where *both factor modes are equal*. This introduces a symmetry in the treatment of *zeros* and *ones* in the factors. Possible applications include observations that have a positive outcome if latent properties are aligned, for instance genes that interact when they have the same effect on a biological process (up-regulate or down-regulate). However, applicability of this machine is limited by the fact that it produces extremely dense data, as can be seen in Fig. 4.6. The density grows exponentially with the number of latent dimensions and thus only models with relatively small L are of practical use.

Note, that the above interpretation of *aligned* latent properties does not hold for $K>2$. However, an analogous machine for arbitrary K could be designed, following the same principles and replacing the NXOR by an all-arguments-equal operator. Cases with $K>2$ in the OR-NXOR-machine lack a simple interpretation for the interaction among the latent dimensions and generate data with even greater density.

The AND-NXOR-Clan

These machines are similar to OR-NXOR-machines in the previous paragraph but instead of requiring any latent dimension to be aligned they require all dimensions to be aligned. In the interesting case of $K=2$, AND-NXOR is equivalent to AND-XOR, with one of the factors being inverted.

We present an application to community detection using the Sampson's monk dataset that was analysed in Section 2.8 (Breiger et al., 1975; Sampson, 1968). Previously, we have used an OR-AND model, such that communities are defined by groups of nodes that a members shares connections with (AND). A connection is present, if the node is member of any community that has this property (OR). In the AND-NXOR logic communities are, again, defined through set if incoming and outgoing connections. Now, connections are present if

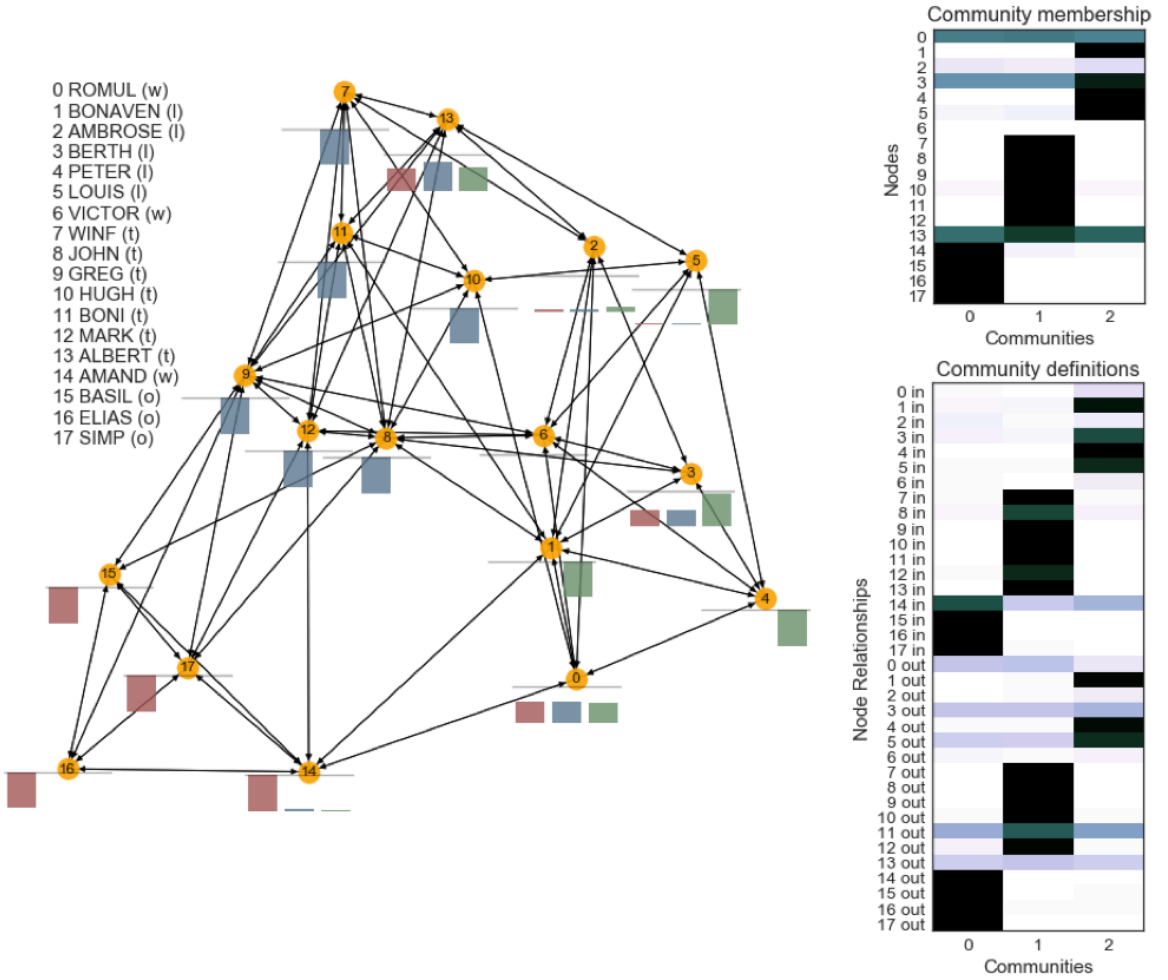


Fig. 4.9 Community detection in a network of monk relations analysed by an AND-NXOR-machine. Compare to Fig. 2.15.

the community membership and the community definitions agree across all communities. It's easier to think of this along the following metaphor. Each monk has a binary opinion (approval/disapproval) on three (number of communities) topics. He is only connected to another monk, if they both agree on all three topics. Here, agreement is modelled through the NXOR-operator and the fact that agreement in all opinions is necessary through the AND-operator. The opinions of a monk can differ depending on whether we consider incoming or outgoing connections. The resulting communities are showgen in Fig. 4.9.

We observe that the community affiliations are very similar to those under OR-AND logic in in Fig. 2.15. There are, however, a few notable differences. John (8) and Greg (9) that where previously associated with multiple groups are now only part of the young Turks (blue). Strikingly, we now find Romul (0) to be associated with all communities. As such, he was also identified by Airoldi et al. (2008) using the mixed membership stochastic block model (MMB), but not in our previous experiments under the OR-AND logic. Furthermore, Ambrose (2), Berth (3) and Albert (13) seem to play an ambiguous role, which has not been picked up by either the MMB nor the OR-AND analysis. This shows, that the AND-XOR logic provides a complimentary, yet interpretable representation.

4.7.3 The XOR-XOR-Class

The XOR-XOR-taxonomy is shown in Fig. 4.10 and composed of only 4 machines, because input inversion for any of the two operators is not defined. We have discussed XOR-AND and AND-XOR classes, describing their relation to OR-AND-machines. XOR-XOR models combine many of their characteristics. In particular, the outer XOR can represent an $L-1$-dimensional NOR operation, analogous to Fig. 4.3. Thus, for more than a few latent dimensions, inference in models of XOR-XOR class is behaves like models in the AND-XOR class. Moreover, the inner XOR is sensitive to aligned/opposing factors for $K=2$, analogous to the AND-XOR-class, while interpretability for $K>2$ is dubious.

Importantly, in the XOR-XOR-clan, any $y_{[\mathbf{n}]}=1$ is explained by exactly one entry in one factor. Any further one in any of the contributing factors results in $y_{[\mathbf{n}]}=0$, as can be understood from in Fig. 4.11.

This rigid constraint limits the expressive ability of XOR-XOR-class machines, which practically manifests in the very regular patterns that we observe for random draws from the machine. This is shown for in Fig. 4.12 for i.i.d. random factors of different expected density.

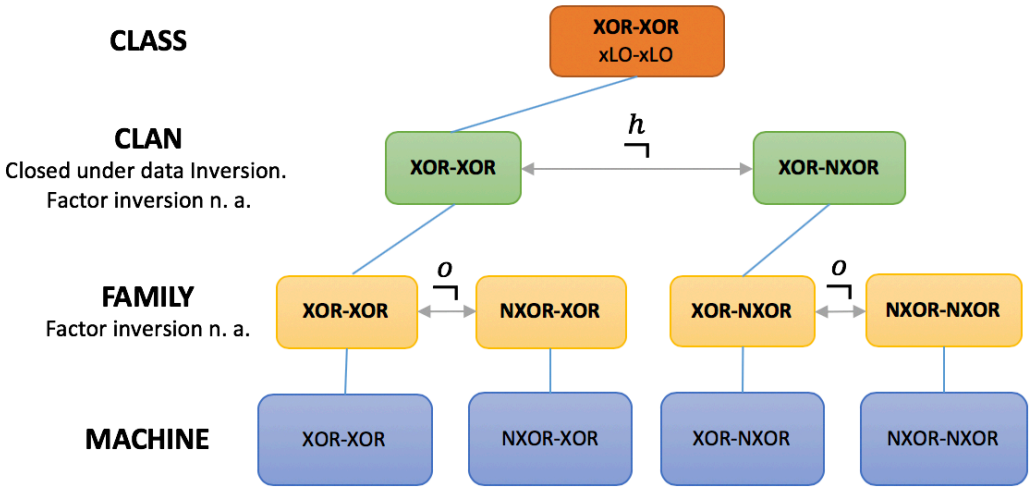


Fig. 4.10 XOR-XOR derived Logical Factorisation Machines

$$
\begin{array}{lcccccc}
\mathtt{XOR}_{n_k} \downarrow & f_{n_1}: & 0 & 1 & 0 & \dots & \\
& f_{n_2}: & 0 & 0 & 0 & \dots & \\
\hline
& & 0 & 1 & 0 & \dots & y_{n_1,n_2} = 1 \\
& & & \overrightarrow{\mathtt{XOR}_l} & & &
\end{array}
$$

Fig. 4.11 Output of an XOR−XOR-machine. Another one at any factor location induces a zero in the output.

It is thus not surprising, that in real-world examples, XOR-XOR-class machines achieve a poor training accuracy. For instance in the single-cell dataset from Pollen et al. (2014), introduced in Section 4.3.2, the best reconstruction is achieved for only two latent dimensions. Here a gene is either predicted to be expressed in all cells or is predicted to be not expressed at all. The corresponding reconstruction for $L = 3$ latent dimensions is shown in Fig. 4.13 and highlights that the ability or XOR−XOR-class machines to reconstruct real-world data is very limited. The training reconstruction accuracy is roughly 70%. Further latent dimensions do not improve the reconstruction accuracy

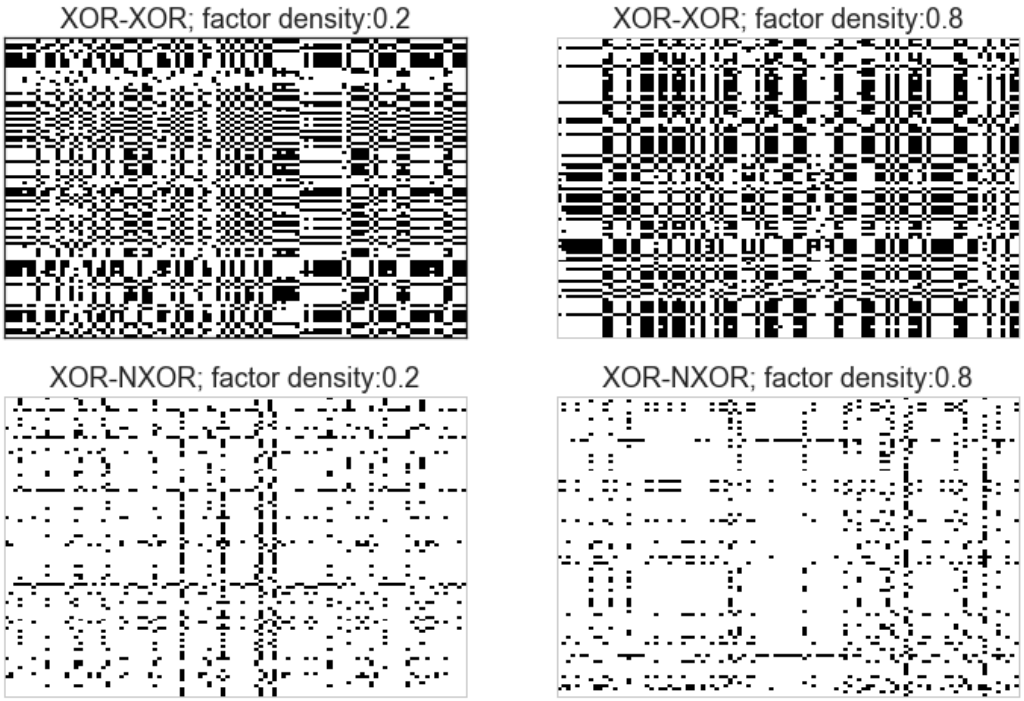


Fig. 4.12 Random draws from XOR-XOR and XOR-NXOR machines with three latent dimensions.

4.8 Reconstruction Accuracies for LFM Model Selection

In analogy to Fig. 4.15, we show the mean reconstruction accuracies from 10 random repetitions of the identical experimental setup, described in Section 4.9. They are shown for $L = 2$ and $L = 5$ latent dimensions are shown in Fig. 4.14.

In the simplest possible case of L=2 latent dimensions, data based on XOR-XOR-clan machines can be recovered by virtually all machines. This is unsurprising, given the regular structure of draws from this model, examples of which are shown in Fig. 4.13. We also observe the expected patterns of model confusion among the other machines. In the more interesting and realistic case of $L = 5$ latent dimensions, the reconstruction accuracies for most models are poor. Only OR-AND-clan machines show consistently perfect reconstructions. Accuracies for XOR-AND-clan machines are around 80% to 90%. Machines in the AND-XOR-class are only slightly better than random, while the remaining LFMs fail entirely.

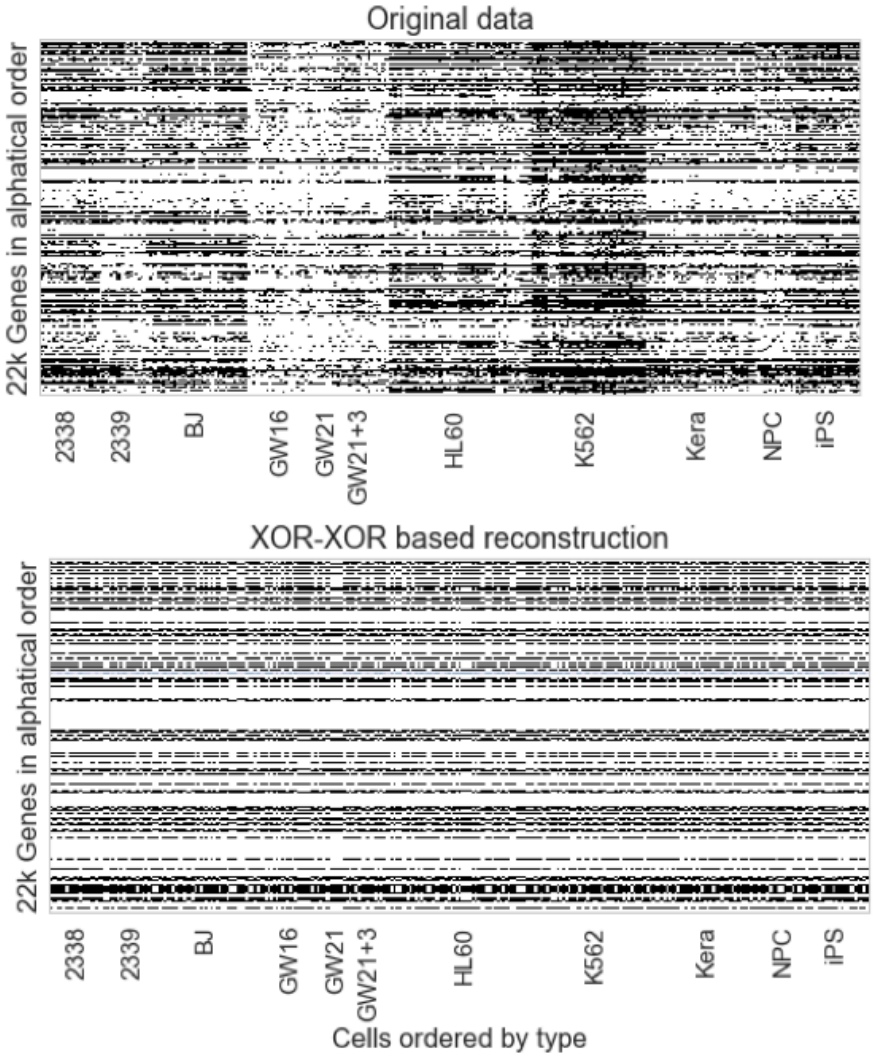


Fig. 4.13 Single cell data from Pollen et al. (2014) and its reconstruction from an XOR-XOR-machine with three latent dimensions.

4.9 Recovering the Data-Generating Logic

After having discussed the taxonomy of all Logical Factorisation Machines, we next probe their ability to recover the data-generating logic from synthetic examples. To this end, we generate data from random factors for all relevant LFM classes. The expected factor density is chosen, such that the data is as close to balanced as possible. For every dataset we fit all machines and treat 20% of the data as unobserved test-set. We then analyse the reconstruction

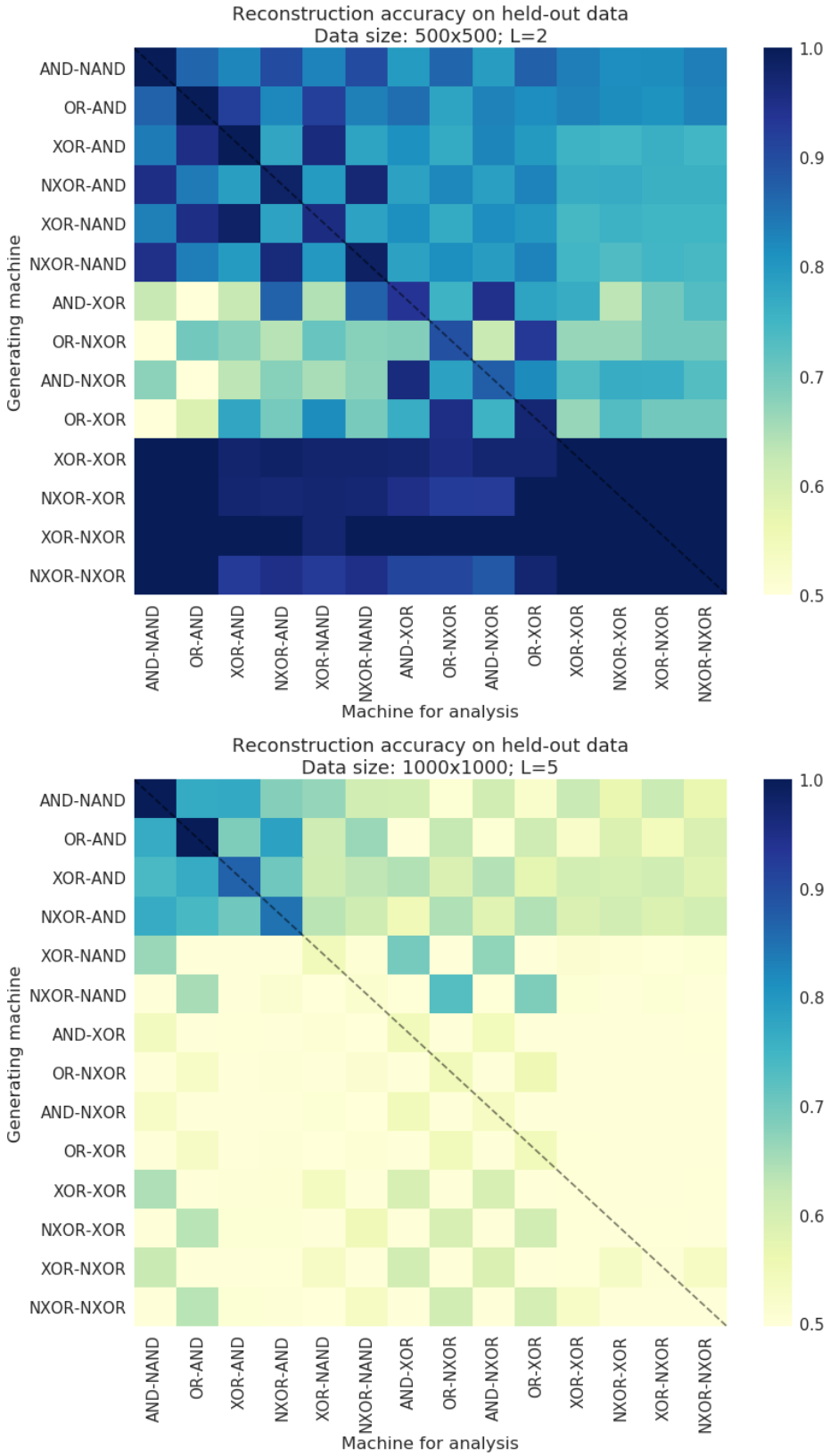


Fig. 4.14 Reconstruction accuracy of LFMs for data generated from LFMs.

accuracy with respect to the test set. Every experiment is repeated ten times and the true dimensionality, L, is assumed to be known and held fixed.

We find that inference in models with XOR operations is challenging and that the sampler gets easily stuck in inadequate regions of the parameter space. To alleviate this, we implement an annealing strategy that slowly raises the temperature during the initial burn-in phase. This improves the results significantly.

Fig. 4.15 simulates the task of selecting the appropriate model. For each of 10 random repetitions, we choose the model with the best test-set prediction and depict the fraction of times each model was chosen in a heatmap. As previously discussed several machines, such as AND-NXOR and AND-XOR, are equivalent after factor inversion in the case of K=2, which can be seen in the high off-diagonal entries in Fig. 4.15 (top). We find that model selection works perfectly for OR-AND-class machines. For XOR-AND-machines it works reasonably well. Models in the AND-XOR family recover the true model less than 50% of the time, while other machines fail entirely. The corresponding reconstruction accuracies are discussed in Appendix 4.8 and confirm that machines in the OR-AND and XOR-AND yield the best performance.

It is important to assert that we probe the fundamental possibility of recovering the latent causes rather than the peculiarities of our inference procedure. To this end we repeat the experiments, initialising all factors to the true data-generating values. Strikingly, this has no significant effect on the results presented above. Overall, this confirms the outcome of our previous discussion that, with a few notable exceptions, most LFMs are of limited use as latent variable models.

4.10 Advanced Usage of our Implementation

One of the most powerful benefits of the modular treatment of our implementation is the definition of complex hierarchies as we have shown for the specification of a hierarchical

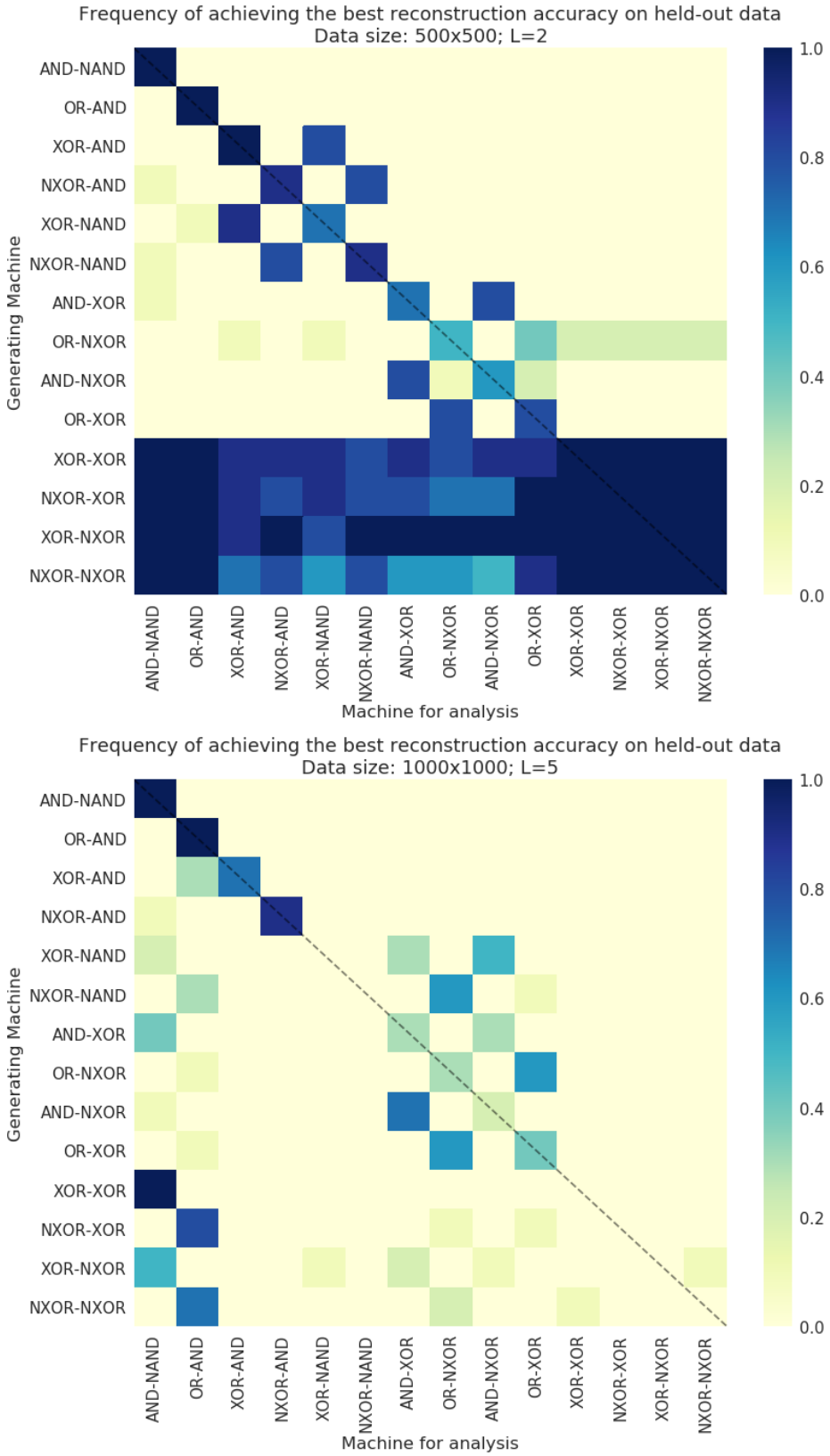


Fig. 4.15 Model selection for different LFMs for synthetic data drawn from LFMs. Shown is a LFM for every family, ordered by clan and class.

factorisation model in Fig. 2.13 on page 43. In Fig. 4.16, we give a more detailed example including usage of different logical products and specification of prior distributions. Lines 14 and 25 show how we can initialise factors and parameters, while lines 17-18 show how these entries can be held fixed during inference. This can be useful for encoding prior information, for instance the presence of a particular set of attributes in a subset of observations. Specification of hyperparameters for independent Bernoulli prior distributions on the factors and for a Beta prior distribution on the noise parameter is shown in lines 21-22. This treatment is far from exhaustive and the interested reader is referred to the GitHub repository for further information.

4.11 Conclusion

We have introduced Logical Factorisation Machines as a generalisation of Boolean Tensor Factorisation. Taxonomies of LFMs have been established that help to identify the equivalence and similarities among the 36 different models, and enabled us to derive general properties. We have developed a framework for posterior inference and prediction that applies to arbitrary LFMs.

Through a theoretical discussion and experiments on synthetic data, we have shown that many LFMs are of very limited practical use. In particular, data that is simulated from machines other than `OR-AND` and `XOR-AND` in a realistic setting of five latent dimensions but without noise, is difficult to reconstruct based on the inferred representations in the ground-truth model. The discovery of processes in the real-world, that generate data following these logics is thus expected to be challenging.

However other LFMs can provide instructive new views on the data. We have given an example that highlight the complementarity of `OR-AND` and `NOR-AND` machines in the discovery of latent patterns of gene expression. Further, we have demonstrated an application

```
# Initialise model. Assume data, X, is in scope.
N, D = X.shape
L1 = 10; L2 = 3  # number of latent dimensions
advanced_model = lfm.Machine()
data = advanced_model.add_matrix(X, fixed=True)

# add factorisation layer with XOR-AND logic
layer1 = advanced_model.add_layer(latent_size=L1, child=data,
 ↪  model='XOR-AND')

# add a second factorisation layer as prior on layer1.z
layer2 = advanced_model.add_layer(latent_size=L2, child=layer1.z,
 ↪  model='OR-AND')

# initialise the first factor explicitly as ones (optional)
layer1.factors[0].val = np.ones([N, L], dtype=np.int8)

# Fix column 0 of a factor (optional)
layer1.factors[1].fixed_entries = np.zeros([N,L], dtype=np.int8)
layer1.factors[1].fixed_entries[0,:] = 1

# Specify priors (optional)
layer1.lbda.beta_prior = (10,1)  # Beta prior for noise
layer1.factors[1].bernoulli_prior = .1  # iid prior for factor entries

# Initialise the noise parameter
layer1.lbda.val = 3.0

# run inference
advanced_model.infer(burn_in_min=100, burn_in_max=1000, no_samples=50)

# inspect the data reconstruction
layer1.output(technique='factor_mean')
```

Fig. 4.16 Advanced example, defining factorisation hierarchies and showing the explicit use of hyperparameters, as well as and fixed factor entries.

of the AND-NXOR-machine to the problem of community detection where it provides a new view on the underlying community structure.

Chapter 5

Discussion

In Section 5.1, we provide concluding thoughts and refer back to the goals we have stated at the beginning of this thesis. We finish with a brief outlook on future work in Section 5.2.

5.1 Concluding Remarks

We have commenced this thesis in Chapter 1 by asking how to gain insights from large, heterogeneous and multidimensional binary datasets. To address this question, we have introduced the *OrMachine* and its generalisation to data of arbitrary arity, the *TensOrMachine*.

We have demonstrated in a wide range of applications, that OrMachines infer sparse, composable latent representations that have an immediate causal interpretation. For instance they infer networks of gene activity, stratify cancer patients or identify work patterns in different staff groups. The inferred representations and sets of properties can lead to new discoveries and hypotheses, for instance:

> *Do inferred gene networks correspond to known biological functions?*
> *Is there a difference in the inferred patient groups with respect to treatment response or survival?*
> *Should we worry about the lack of interaction among nurses and medics?*

Being able to ask these questions can lead to follow-up investigations and can serve as a valuable tool in the process of turning data into knowledge. Moreover, previous domain-knowledge is often available but can not easily be integrated into machine learning algorithms. Due to the clear interpretation of all variables, this is easy in the proposed models. For example, we have shown how to encode prior information of the cancer type of patients into an otherwise unsupervised analysis of their somatic mutation patterns. We further have met the requirement of scalability, by developing posterior inference that scales to Billions of datapoints through exploitation of the combinatorial structure of the factor conditionals.

Besides these practical contributions, we have also made progress on approximating the combinatorial problems that underlie Boolean Matrix Factorisation and the closely related biclique cover problem. OrMachines infer decompositions with higher accuracies than all existing techniques throughout a wide range of simulated conditions and real-world examples. They also also provide full posterior inference for the factor matrices in Boolean factorisation which is relevant in applications, for instance for controlling false positive rates in collaborative filtering and which improves the interpretability of the inferred patterns. Our treatment within the framework of probabilistic inference opens further avenues for choosing the latent dimensionality or rank. We have shown that an Indian Buffet Process as prior distribution on the factors allows for a notionally unbounded number of latent features and enables us to accurately infer the true number of underlying dimensions.

Our final contribution is the generalisation of OrMachines through modification of the factor product, replacing its logical constituents, `OR` and `AND` operations, by other logical operators. This gave rise to a family of 36 models, called *Logical Factorisation Machines* (LFMs). We have developed a genealogy of LFMs that highlights equivalence and similarities among the various models and allows for a general framework for posterior inference. The practical use of various LFMs, in particular their ability to infer the generating logic from data was investigated in simulations and shown to be limited to a few practically useful

LFMs. Example of `NOR-AND`, as well as `AND-NXOR` have shown how these less established logics can provide novel insights.

5.2 Future Work

Directions for future work are manifold. Results in Chapter 4 indicate that LFM operators that return `True` if and only all inputs are equal, can be a component of new, useful models. Further extensions to LFMs include general Boolean operators, as well as ternary logic or quantum logic. Moreover, it would be interesting to combine differnt data generating logics, e.g. such that different observations are created under different logics. This could potentially uncover cells or patients that share latent properties but where different mechanisms lead to the relevant outcome.

Another direction of research should address the use of nonparametric prior distributions. The Indian Buffet Process based approach described in Section 2.9 can be extended to Tensors and higher-arity data, as well as to general Logical Factorisation Machines. It would be of particular practical interest to make it more scalable, e.g. by parallelising inference following the approach of Zhang et al. (2017). Further, it would be interesting to analyse whether the solutions are systematically different than those obtained with a finite prior distribution.

Yet another direction of research could extend our methods to cases where the data is extremely sparse, e.g. has only 1/10000 positives. This problem has recently been addressed by Neumann (2018). In this particular case, but also in general, it would be interesting to find a suitable mechanism that assigns different weight to positive and negative observations. Note, that this is different from the more conventional approach of treating positive and negative predictions differently, mentioned in Section 2.2, which is a further possible extension.

Another interesting application is in knowledge bases research, where LFMs could be used to discover or predict semantic relationships similar to the work by Erdos and Miettinen (2013a). Here, the flexibility or hierarchical machines and the favourable scalability could allow for novel use cases.

Lastly, it would be desirable to gain a better understanding of why the probabilistic formulation of Boolean matrix factorisation, together with sampling-based inference, leads to solutions of unprecedented accuracy and to assess whether the same principles apply to the approximation of similarly hard combinatorial problem.

References

Agrawal, R., Srikant, R., and Others (1994). Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, volume 1215, pages 487–499.

Airoldi, E. M., Blei, D. M., Fienberg, S. E., and Xing, E. P. (2008). Mixed Membership Stochastic Blockmodels. *Journal of Machine Learning Research*, 9(Sep):1981–2014.

Bao, J. M., He, M. Y., Liu, Y. W., Lu, Y. J., Hong, Y. Q., Luo, H. H., Ren, Z. L., Zhao, S. C., and Jiang, Y. (2015). AGE/RAGE/Akt pathway contributes to prostate cancer cell proliferation by promoting Rb phosphorylation and degradation. *Am J Cancer Res*, 5(5):1741–1750.

Belohlavek, R., Glodeanu, C., and Vychodil, V. (2013). Optimal factorization of three-way binary data using triadic concepts. *Order*, 30(2):437–454.

Belohlavek, R. and Trnecka, M. (2015). From-below approximations in Boolean matrix factorization: Geometry and new algorithm. *Journal of Computer and System Sciences*, 81(8):1678–1697.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation Learning: a review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828.

Bishop, C. M. (2013). *Pattern Recognition and Machine Learning*. Springer, 1st editio edition.

Bishop, C. M. and Tipping, M. E. (1998). A hierarchical latent variable model for data visualization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):281–293.

Blainey, P. C. and Quake, S. R. (2014). Dissecting Genomic Diversity, One Cell At a Time. *Nat. Methods*, 11(1):19–21.

Breiger, R. L., Boorman, S. A., and Arabie, P. (1975). An Algorithm for Clustering Relational Data With Applications To Social Network Analysis and Comparison With Multidimensional Scaling. *Journal of mathematical psychology*, 12(3):328–383.

Chaudhari, P., Choromanska, A., Soatto, S., and LeCun, Y. (2016). Entropy-sgd: Biasing gradient descent into wide valleys. *arXiv preprint arXiv:1611.01838*.

Chen, E. Y., Tan, C. M., Kou, Y., Duan, Q., Wang, Z., Meirelles, G., Clark, N. R., and Ma'ayan, A. (2013). Enrichr: Interactive and Collaborative Html5 Gene List Enrichment Analysis Tool. *BMC Bioinformatics*, 14(1):128.

Collins, M., Dasgupta, S., and Schapire, R. E. (2002). A Generalization of Principal Components Analysis to the Exponential Family.

Demyanenko, G. P., Siesser, P. F., Wright, A. G., Brennaman, L. H., Bartsch, U., Schachner, M., and Maness, P. F. (2010). L1 and Chl1 Cooperate in Thalamocortical Axon Targeting. *Cerebral Cortex*, 21(2):401–412.

Dıez, F. J. and Druzdzel, M. J. (2006). Canonical probabilistic models for knowledge engineering. Technical Report CISIAD-06-01, UNED, Madrid, Spain.

Doshi-Velez, F. et al. (2009). The indian buffet process: Scalable inference and extensions. *Master's thesis, The University of Cambridge*.

Downward, J. (2003). Targeting ras signalling pathways in cancer therapy. *Nature Reviews Cancer*, 3(1):11.

Eck, D., Bengio, Y., and Courville, A. C. (2009). An infinite factor model hierarchy via a noisy-or mechanism. In Bengio, Y., Schuurmans, D., Lafferty, J. D., Williams, C. K. I., and Culotta, A., editors, *Advances in Neural Information Processing Systems 22*, pages 405–413. Curran Associates, Inc.

Erdos, D. and Miettinen, P. (2013a). Discovering facts with boolean tensor tucker decomposition. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 1569–1572. ACM.

Erdos, D. and Miettinen, P. (2013b). Walk'n'merge: A scalable algorithm for boolean tensor factorization. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pages 1037–1042. IEEE.

Figel, S. and Gelman, I. H. (2011). Focal adhesion kinase controls prostate cancer progression via intrinsic kinase and scaffolding functions. *Anticancer Agents Med Chem*, 11(7):607–616.

Fortunato, S. and Hric, D. (2016). Community detection in networks: A user guide. *arXiv:1608.00163*.

Frank, M., Streich, A. P., Basin, D., and Buhmann, J. M. (2012). Multi-assignment clustering for boolean data. *Journal of Machine Learning Research*, 13(Feb):459–489.

Ganter, B. and Wille, R. (2012). *Formal concept analysis: mathematical foundations*. Springer Science & Business Media.

Geman, S. and Geman, D. (1984). Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741.

Gershman, S. J. and Blei, D. M. (2012). A tutorial on Bayesian nonparametric models. *Journal of Mathematical Psychology*, 56(1):1–12.

Ghahramani, Z. (2012). Bayesian non-parametrics and the probabilistic approach to modelling. *Philos Trans A Math Phys Eng Sci*, 371(1984):20110553.

Ghahramani, Z. and Griffiths, T. L. (2006). Infinite latent feature models and the indian buffet process. In *Advances in neural information processing systems*, pages 475–482.

Gimpel, J. F. (1974). The minimization of spatially-multiplexed character sets. *Communications of the ACM*, 17(6):315–318.

Griffiths, T. L. and Ghahramani, Z. (2011). The Indian Buffet Process: An Introduction and Review. *J. Mach. Learn. Res.*, 12:1185–1224.

Harari, G. M., Gosling, S. D., Wang, R., Chen, F., Chen, Z., and Campbell, A. T. (2017). Patterns of behavior change in students over an academic term: A preliminary study of activity and sociability behaviors using smartphone sensing methods. *Computers in Human Behavior*, 67:129 – 138.

Harvey, K. F., Zhang, X., and Thomas, D. M. (2013). The hippo pathway and human cancer. *Nature reviews Cancer*, 13(4):nrc3458.

Heller, K. A. and Ghahramani, Z. (2007). A nonparametric Bayesian approach to modeling overlapping clusters. In *Artificial Intelligence and Statistics*, pages 187–194.

Henderson, J., Ho, J. C., Kho, A. N., Denny, J. C., Malin, B. A., Sun, J., and Ghosh, J. (2017). Granite: Diversified, sparse tensor factorization for electronic health record-based phenotyping. In *2017 IEEE International Conference on Healthcare Informatics (ICHI)*, pages 214–223.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

Hinton, G. E., Sallans, B., and Ghahramani, Z. (1998). A hierarchical community of experts. In *Learning in graphical models*, pages 479–494. Springer.

Hjort, N. L., Holmes, C. C., Müller, P., and Walker, S. G. (2010). {Bayes}ian nonparametrics. *AMC*, 10:12.

Ho, J. C., Ghosh, J., Steinhubl, S. R., Stewart, W. F., Denny, J. C., Malin, B. A., and Sun, J. (2014). Limestone: high-throughput candidate phenotype generation via tensor factorization. *J Biomed Inform*, 52:199–211.

Jaakkola, T. S. and Jordan, M. I. (1999). Variational Probabilistic Inference and the QMR-DT Network. *J. Artif. Int. Res.*, 10(1):291–322.

Juan, A. and Vidal, E. (2004). Bernoulli mixture models for binary images. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 3, pages 367–370. IEEE.

Kolda, T. G. and Bader, B. W. (2009). Tensor decompositions and applications. *SIAM REVIEW*, 51(3):455–500.

Lázaro-Gredilla, M., Liu, Y., Phoenix, D. S., and George, D. (2016). Hierarchical Compositional Feature Learning. *arXiv preprint arXiv:1611.02252*.

Leenen, I., Van Mechelen, I., De Boeck, P., and Rosenberg, S. (1999). Indclas: A three-way hierarchical classes model. *Psychometrika*, 64(1):9–24.

Lin, H. W. and Tegmark, M. (2016). Why does deep and cheap learning work so well? *arXiv preprint arXiv:1608.08225*.

Liu, J. (1996). Miscellanea. Peskun's Theorem and a modified discrete-state Gibbs sampler. *Biometrika*, 83(3):681–682.

Lopez-Bendito, G., Flames, N., Ma, L., Fouquet, C., Meglio, T. D., Chedotal, A., Tessier-Lavigne, M., and Marin, O. (2007). Robo1 and Robo2 Cooperate To Control the Guidance of Major Axonal Tracts in the Mammalian Forebrain. *Journal of Neuroscience*, 27(13):3395–3407.

MacKay, D. J. (1992). *Bayesian methods for adaptive models*. PhD thesis, California Institute of Technology.

Manias, E. and Street, A. (2001). Nurse-doctor interactions during critical care ward rounds. *J Clin Nurs*, 10(4):442–450.

Meeds, E., Ghahramani, Z., Neal, R. M., and Roweis, S. T. (2007). Modeling dyadic data with binary latent factors. *Advances in neural information processing systems*, 19:977.

Metzler, S. and Miettinen, P. (2015). Clustering boolean tensors. *Data Mining and Knowledge Discovery*, 29(5):1343–1373.

Miettinen, P. (2011). Boolean tensor factorizations. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 447–456. IEEE.

Miettinen, P., Mielikäinen, T., Gionis, A., Das, G., and Mannila, H. (2006). *The Discrete Basis Problem*, pages 335–346. Springer Berlin Heidelberg.

Miettinen, P. and Vreeken, J. (2014). Mdl4bmf: Minimum Description Length for Boolean Matrix Factorization. *ACM Trans. Knowl. Discov. Data*, 8(4):18:1—-18:31.

Mohamed, S., Ghahramani, Z., and Heller, K. A. (2009). Bayesian Exponential Family PCA. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 21*, pages 1089–1096. Curran Associates, Inc.

Murray, I. and Ghahramani, Z. (2005). A note on the evidence and bayesian occam's razor.

Nau, D. S., Markowsky, G., Woodbury, M. A., and Amos, D. B. (1978). A mathematical analysis of human leukocyte antigen serology. *Mathematical Biosciences*, 40(3-4):243–270.

Neumann, S. (2018). Bipartite stochastic block models with tiny clusters. In *Thirty-second Conference on Neural Information Processing Systems (NIPS) 2018*.

Park, N., Oh, S., and Kang, U. (2017). Fast and scalable distributed boolean tensor factorization.

Patel, A. P., Tirosh, I., Trombetta, J. J., Shalek, A. K., Gillespie, S. M., Wakimoto, H., Cahill, D. P., Nahed, B. V., Curry, W. T., Martuza, R. L., Louis, D. N., Rozenblatt-Rosen, O., Suva, M. L., Regev, A., and Bernstein, B. E. (2014). Single-Cell Rna-Seq Highlights Intratumoral Heterogeneity in Primary Glioblastoma. *Science*, 344(6190):1396–1401.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Pearl, J. et al. (2009). Causal inference in statistics: An overview. *Statistics surveys*, 3:96–146.

Peskun, P. H. (1973). Optimum monte-carlo sampling using markov chains. *Biometrika*, 60(3):607–612.

Polakis, P. (2012). Wnt signaling in cancer. *Cold Spring Harbor perspectives in biology*, 4(5):a008052.

Pollen, A. A., Nowakowski, T. J., Shuga, J., Wang, X., Leyrat, A. A., Lui, J. H., Li, N., Szpankowski, L., Fowler, B., Chen, P., et al. (2014). Low-coverage single-cell mRNA sequencing reveals cellular heterogeneity and activated signaling pathways in developing cerebral cortex. *Nature biotechnology*, 32(10):1053.

Raman, A. V., Pitts, M. W., Seyedali, A., Hashimoto, A. C., Bellinger, F. P., and Berry, M. J. (2013). Selenoprotein W Expression and Regulation in Mouse Brain and Neurons. *Brain and Behavior*, 3(5):562–574.

Ranganath, R., Tang, L., Charlin, L., and Blei, D. M. (2015). Deep Exponential Families. In *AISTATS*.

Ravanbakhsh, S., Póczos, B., and Greiner, R. (2016). Boolean Matrix Factorization and Noisy Completion via Message Passing. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *JMLR: W&CP*.

Rukat, T., Holmes, C., and Yau, C. (2018). Probabilistic boolean tensor decomposition. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4413–4422, Stockholmsmässan, Stockholm Sweden. PMLR.

Rukat, T., Holmes, C. C., Titsias, M. K., and Yau, C. (2017a). Bayesian Boolean Matrix Factorisation. *Proceedings of the 34th Annual International Conference on Machine Learning*, pages 2969–2978.

Rukat, T., Lange, D., and Archambeau, C. (2017b). An interpretable latent variable model for attribute applicability in the amazon catalogue. Proceedings of the NIPS 2017 Symposium on Interpretable Machine Learning.

Sampson, S. F. (1968). *A novitiate in a period of change: An experimental and case study of social relationships*. PhD thesis, Cornell University Ithaca, NY.

Šingliar, T. and Hauskrecht, M. (2006). Noisy-OR Component Analysis and Its Application to Link Analysis. *J. Mach. Learn. Res.*, 7:2189–2213.

Srinivas, S. (2013). A generalization of the noisy-or model.

Streich, A. P., Frank, M., Basin, D., and Buhmann, J. M. (2009). Multi-assignment clustering for Boolean data. In *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*.

Su, A. I., Wiltshire, T., Batalov, S., Lapp, H., Ching, K. A., Block, D., Zhang, J., Soden, R., Hayakawa, M., Kreiman, G., Cooke, M. P., Walker, J. R., and Hogenesch, J. B. (2004). A Gene Atlas of the Mouse and Human Protein-Encoding Transcriptomes. *Proceedings of the National Academy of Sciences*, 101(16):6062–6067.

Su, X. and Khoshgoftaar, T. M. (2009). A Survey of Collaborative Filtering Techniques. *Adv. in Artif. Intell.*, 2009:4:2—-4:2.

Teh, Y. W., Grür, D., and Ghahramani, Z. (2007). Stick-breaking construction for the indian buffet process. In Meila, M. and Shen, X., editors, *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, volume 2 of *Proceedings of Machine Learning Research*, pages 556–563, San Juan, Puerto Rico. PMLR.

The Tabula Muris Consortium, Quake, S. R., Wyss-Coray, T., and Darmanis, S. (2017). Transcriptomic characterization of 20 organs and tissues from mouse at single cell resolution creates a tabula muris. *bioRxiv*.

Tipping, M. E. (1999). Probabilistic Visualisation of High-Dimensional Binary Data. In Kearns, M. J., Solla, S. A., and Cohn, D. A., editors, *Advances in Neural Information Processing Systems 11*, pages 592–598. MIT Press.

Trapnell, C. (2015). Defining Cell Types and States With Single-Cell Genomics. *Genome Research*, 25(10):1491–1498.

Upadhya, S. C., Smith, T. K., Brennan, P. A., Mychaleckyj, J. C., and Hegde, A. N. (2011). Expression Profiling Reveals Differential Gene Induction Underlying Specific and Non-Specific Memory for Pheromones in Mice. *Neurochemistry International*, 59(6):787–803.

Vanhems, P., Barrat, A., Cattuto, C., Pinton, J.-F., Khanafer, N., Régis, C., Kim, B.-a., Comte, B., and Voirin, N. (2013). Estimating potential infection transmission routes in hospital wards using wearable proximity sensors. *PLoS ONE*, 8(9):e73970.

Vazirani, V. V. (2013). *Approximation algorithms*. Springer Science & Business Media.

Vomlel, J. (2015). Generalizations of the noisy-or model. *Kybernetika*, 51(3):508–524.

Wegener, I. (1987). *The complexity of Boolean functions*. BG Teubner.

Weinstein, J. N., Collisson, E. A., Mills, G. B., Shaw, K. R. M., Ozenberger, B. A., Ellrott, K., Shmulevich, I., Sander, C., Stuart, J. M., Network, C. G. A. R., et al. (2013). The cancer genome atlas pan-cancer analysis project. *Nature genetics*, 45(10):1113.

Wood, F., Griffiths, T., and Ghahramani, Z. (2012). A Non-Parametric Bayesian Method for Inferring Hidden Causes. *arXiv preprint arXiv:1206.6865*.

Yang, J. and Leskovec, J. (2012). Community-affiliation graph model for overlapping network community detection. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 1170–1175. IEEE.

Zhang, Y. L., Wang, R. C., Cheng, K., Ring, B. Z., and Su, L. (2017). Roles of Rap1 signaling in tumor cell migration and invasion. *Cancer Biol Med*, 14(1):90–99.

Zheng, Y., Cheng, X.-R., Zhou, W.-X., and Zhang, Y.-X. (2008). Gene Expression Patterns of Hippocampus and Cerebral Cortex of Senescence-Accelerated Mouse Treated With Huang-Lian-Jie-Du Decoction. *Neuroscience Letters*, 439(2):119–124.

Appendix A

Formal Derivation of Factor Conditionals

Here we derive the full conditional distribution for a factor entry $f_{n_k l}$ as given in eq. (3.4) and for the special case of data with two modes and with a slightly different notation in eq. (2.3).

For constant priors, $p(f_{n_k l}) = \text{const.}$, the conditional is given by normalising the likelihood for $f_{n_k l} \in \{0, 1\}$. The likelihood has the form

$$p(x_{[\mathbf{n}]}|.) = \sigma\left[\lambda \tilde{x}_{[\mathbf{n}]}\left(1 - 2\prod_l\left[1 - \prod_{n\in[\mathbf{n}]} f_{nl}\right]\right)\right] \tag{A.1}$$

and is factorial in the data-points $x_{[\mathbf{n}]}$. Terms that do not depend on $f_{n_k l}$ cancel in the conditional and thus we take the product over all $[\mathbf{n}]$ with n_k fixed. Then normalising gives

$$p(f_{n_k l} = 1|.) = \frac{\prod_{[\mathbf{n}],\, n_k \text{fixed}} p(x_{[\mathbf{n}]}|f_{n_k l} = 1, \text{rest})}{\prod_{[\mathbf{n}],\, n_k \text{fixed}} p(x_{[\mathbf{n}]}|f_{n_k l} = 1, \text{rest}) + \prod_{[\mathbf{n}],\, n_k \text{fixed}} p(x_{[\mathbf{n}]}|f_{n_k l} = 0, \text{rest})} \tag{A.2}$$

$$= \sigma\left[\sum_{\substack{[\mathbf{n}] \\ n_k \text{fixed}}} \log \frac{p(x_{[\mathbf{n}]}|f_{n_k l} = 1, \text{rest})}{p(x_{[\mathbf{n}]}|f_{n_k l} = 0, \text{rest})}\right]. \tag{A.3}$$

Considering the term inside the logarithm in eq. (A.3) and using eq. (A.1) we find

$$\frac{p(x_{[\mathbf{n}]}|f_{n_k l}=1,\text{rest})}{p(x_{[\mathbf{n}]}|f_{n_k l}=0,\text{rest})} = \begin{cases} 1;\ \text{if } \Big(\prod\limits_{n\in[\mathbf{n}]/n_k} f_{nl}\Big)\prod\limits_{l'\neq l}\Big(1-\prod\limits_{n\in[\mathbf{n}]} f_{nl'}\Big)=0 \\ \frac{1+\exp(\lambda\tilde{x}_{[\mathbf{n}]})}{1+\exp(-\lambda\tilde{x}_{[\mathbf{n}]})} = e^{\lambda\tilde{x}_{[\mathbf{n}]}};\ \text{otherwise}\ . \end{cases} \tag{A.4}$$

The first equality describes all cases where the term inside the parenthesis in eq. (A.1) takes a value that is independent of the value of $f_{n_k l}$. The second term follows in all other cases and by expanding the logistic sigmoid. Hence, we can write eq. (A.3) as

$$p(f_{n_k l}=1|.) = \sigma\left[\lambda \sum_{\substack{[\mathbf{n}]\\ n_k\text{fixed}}} \tilde{x}_{[\mathbf{n}]} \overbrace{\Big(\prod_{n\in[\mathbf{n}]/n_k} f_{nl}\Big)\prod_{l'\neq l}\Big(1-\prod_{n\in[\mathbf{n}]} f_{nl'}\Big)}^{M_{(n_k,l)\to[\mathbf{n}]}}\right]. \tag{A.5}$$

Appendix B

IBP Experiment with Noise

The ability of the IBP-OrMachine to infer the true number of latent dimensions on synthetic data is shown for noise levels of 10%, 20% and 30% in Figs. B.1, B.2 and B.3, respectively. Depicted are posterior means. Besides adding noise, the experimental conditions are identical to what is described in Section 2.9

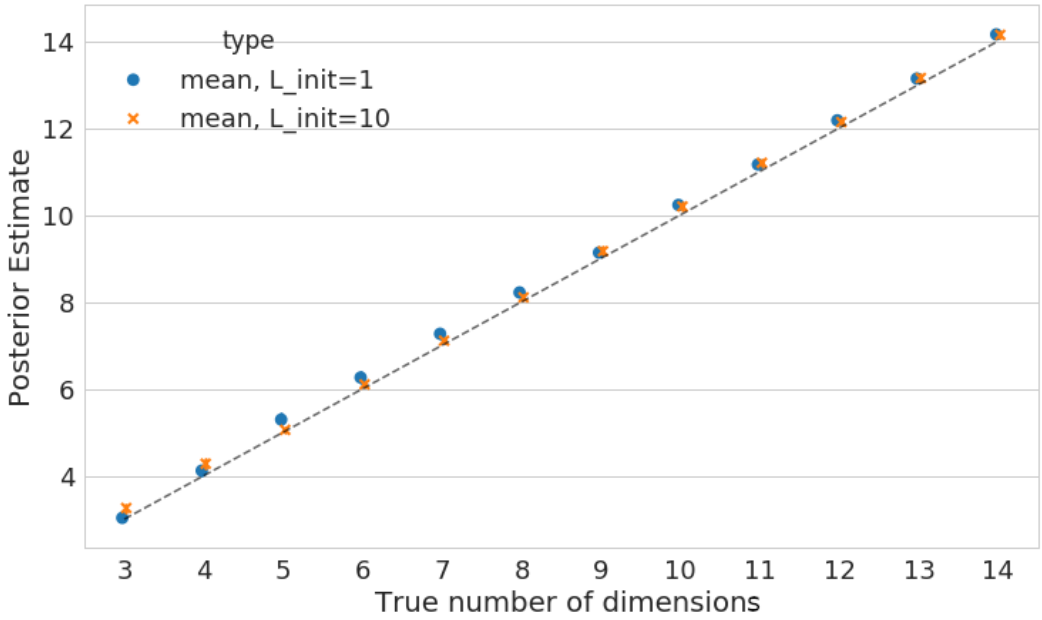


Fig. B.1 Inference over the number of latent dimensions in the infinite OrMachine with a noise level of 10%.

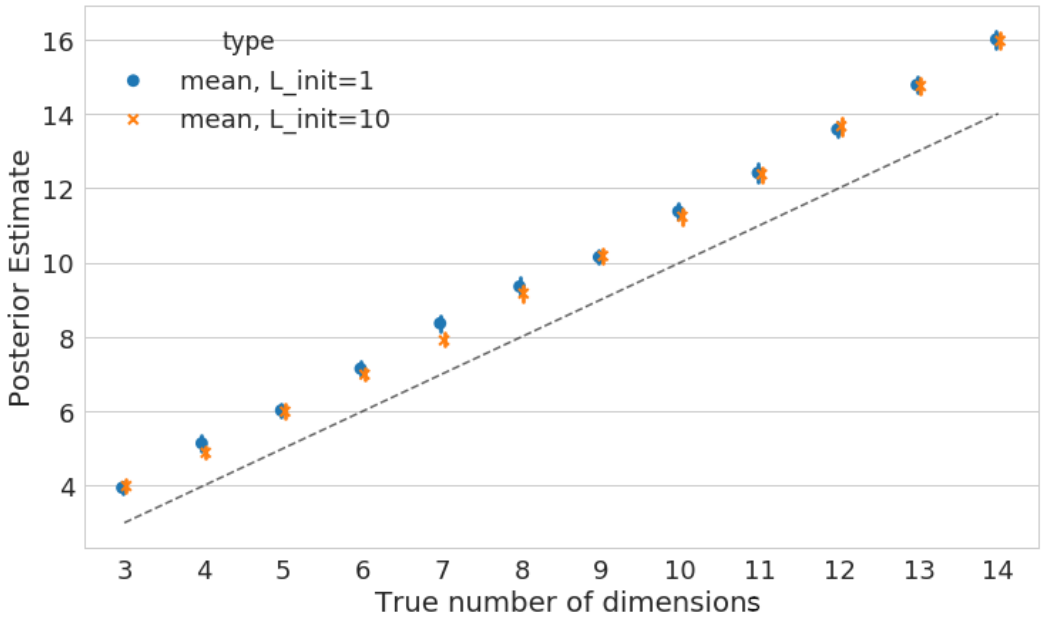


Fig. B.2 Inference over the number of latent dimensions in the infinite OrMachine with a noise level of 20%.

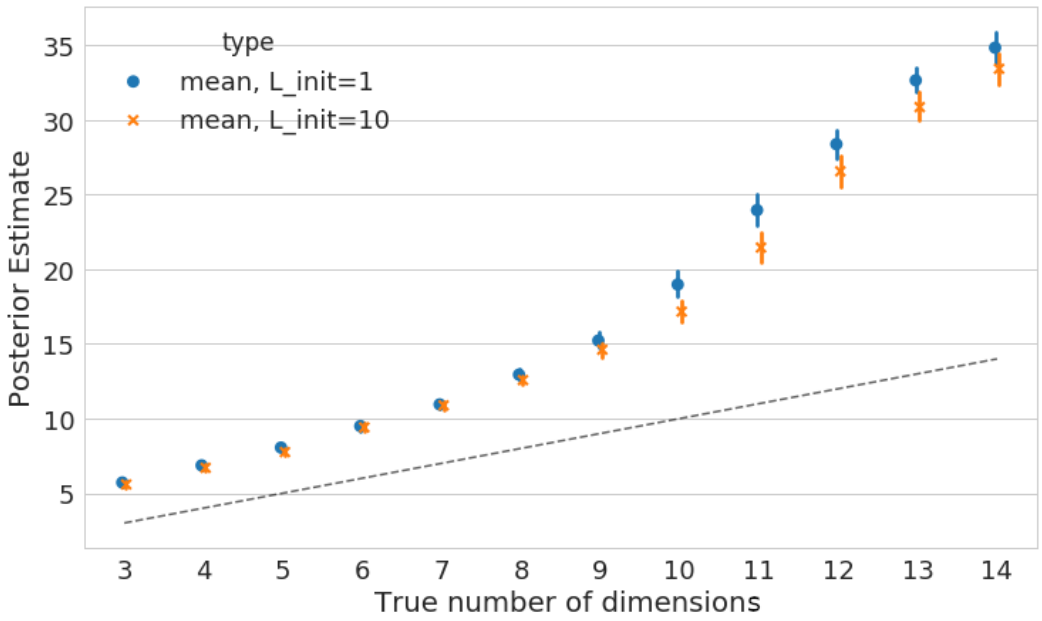


Fig. B.3 Inference over the number of latent dimensions in the infinite OrMachine with a noise level of 30%.

Appendix C

Derivations for the Dispersion Parameter

MAP updates for λ

The result from eq. (2.8),

$$\lambda_{\text{MAP}} = \text{logit}\left[\frac{\alpha + T}{\alpha + \beta + N \cdot D}\right] \tag{C.1}$$

can be derived by considering the Beta-prior on $\sigma(\lambda)$,

$$p\left(\sigma(\lambda)\right) = \text{Beta}(\sigma(\lambda)|\alpha, \beta) \propto \sigma(\lambda)^{\alpha-1}\sigma(-\lambda)^{\beta-1} \ , \tag{C.2}$$

and applying a change of variables

$$p\left(\lambda\right) = \text{Beta}(\sigma(\lambda)|\alpha, \beta)\left|\frac{d\sigma(\lambda)}{d\lambda}\right| \propto \sigma(\lambda)^{\alpha}\sigma(-\lambda)^{\beta} \ . \tag{C.3}$$

This yields the conditional

$$p(\lambda|\cdot) \propto \sigma(\lambda)^{T+\alpha}\sigma(-\lambda)^{F+\beta} \tag{C.4}$$

from which the MAP estimate follows.

Conditional Posterior Uncertainty of λ

Consider the full conditional $p(\sigma(\lambda)|.)$ with $\sigma(\lambda) \in [0.5, 1]$. Again, the expression follows from a change of variables:

$$p(\sigma(\lambda)|\cdot) \propto \sigma(\lambda)^{T+\alpha-2}\, \sigma(\lambda)^{F+\beta-2} \,. \tag{C.5}$$

Except for its domain this has the form of a Beta-density. In the relevant regime of $T \gg F$ and at least hundreds of data points, we can extend the domain of $p(\sigma(\lambda)|\cdot)$ to $[0, 1]$, introducing only a negligible error. This allows us to treat $p(\sigma(\lambda)|\cdot)$ as Beta-density with known variance,

$$\mathrm{var}(\sigma(\lambda)) \approx \frac{(T+\alpha-2)\,(F+\beta-2)}{(T+\alpha+F+\beta-4)^2\,(T+\alpha+F+\beta-3)} \,. \tag{C.6}$$

This variance decreases quadratically in the number of data points and hence it is justified to neglect the conditional posterior uncertainty of λ in practice.

Appendix D

LFMs: Derivations and Examples

D.1 Conditional Distributions

Following eqs. (4.32), expressions for the full conditionals of Logical Factorisation Machine clans take the following form. We give also give the special result for $K = 2$ dimensions, in notation reminiscent of the OrMachine in Chapter 2. For an OR-AND-machine we have

$$\begin{aligned} g_{nl} &= \sum_d \tilde{x}_{nd}\ \texttt{AND}(f_{lk})\ \texttt{NOR}_{l'}\texttt{AND}_k(f_{kl'}) \\ &\overset{K=2}{\to} \sum_d \tilde{x}_{nd}\ u_{ld}\ \prod_{l'}(1 - z_{nl}u_{ld})\ . \end{aligned} \tag{D.1}$$

For an XOR-NAND machine we have

$$\begin{aligned} g_{nl} &= \sum_d \tilde{x}_{nd}\ \texttt{AND}(f_{lk})\ \delta\texttt{XOR}_{l'}\texttt{AND}_k(f_{kl'}) \\ &\overset{K=2}{\to} \sum_d \tilde{x}_{nd}\ u_{ld}\ \delta\underset{l'}{\texttt{XOR}}(z_{nl'}u_{l'd})\ . \end{aligned} \tag{D.2}$$

For an XOR-NAND machine we have

$$\begin{aligned} g_{nl} &= \sum_d -\tilde{x}_{nd}\ \texttt{AND}(f_{lk})\ \delta\texttt{XOR}_{l'}\texttt{NAND}_k(f_{kl'}) \\ &\overset{K=2}{\to} -\sum_d \tilde{x}_{nd}\ u_{ld}\ \delta\underset{l'}{\texttt{XOR}}(1 - z_{nl'}u_{l'd}) \ . \end{aligned} \tag{D.3}$$

For an OR-XOR machine we have

$$\begin{aligned} g_{nl} &= \sum_d \tilde{x}_{nd}\ \delta\texttt{XOR}(f_{lk})\ \texttt{NOR}_{l'}\texttt{XOR}_k(f_{kl'}) \\ &\overset{K=2}{\to} \sum_d -\tilde{x}_{nd}\ \tilde{u}_{ld}\ \prod_{l'}(1 - \text{abs}(z_{nl} - u_{ld})) \ . \end{aligned} \tag{D.4}$$

For an NAND-XOR machine we have

$$\begin{aligned} g_{nl} &= \sum_d -\tilde{x}_{nd}\ \delta\texttt{XOR}(f_{lk})\ \texttt{AND}_{l'}\texttt{XOR}_k(f_{kl'}) \\ &\overset{K=2}{\to} \sum_d \tilde{x}_{nd}\ \tilde{u}_{ld}\ \prod_{l'}\text{abs}(z_{nl} - u_{ld}) \ . \end{aligned} \tag{D.5}$$

For an XOR-XOR machine we have

$$\begin{aligned} g_{nl} &= \sum_d \tilde{x}_{nd}\ \delta\texttt{XOR}(f_{lk})\ \delta\texttt{XOR}_{l'}\texttt{XOR}_k(f_{kl'}) \\ &\overset{K=2}{\to} \sum_d -\tilde{x}_{nd}\ \tilde{u}_{ld}\ \delta\texttt{XOR}_{l'}[\text{abs}(z_{nl'} - u_{l'd})] \ . \end{aligned} \tag{D.6}$$

For an XOR-NXOR machine we have

$$\begin{aligned} g_{nl} &= \sum_d \tilde{x}_{nd}\ \delta\texttt{NXOR}(f_{lk})\ \delta\texttt{XOR}_{l'}\texttt{NXOR}_k(f_{kl'}) \\ &= \sum_d -\tilde{x}_{nd}\ \delta\texttt{XOR}(f_{lk})\ \delta\texttt{XOR}_{l'}\texttt{NXOR}_k(f_{kl'}) \\ &\overset{K=2}{\to} \sum_d \tilde{x}_{nd}\ \tilde{u}_{ld}\ \delta\texttt{XOR}_{l'}\left[1 - \text{abs}(u_{dl'} - z_{nl'})\right] \ . \end{aligned} \tag{D.7}$$

For an AND-NXOR machine we have

$$\begin{aligned} g_{nl} &= \sum_d \tilde{x}_{nd}\ \delta\mathtt{NXOR}(f_{lk})\ \delta\mathtt{AND}_{l'}\mathtt{NXOR}_k(f_{kl'}) \\ &= \sum_d -\tilde{x}_{nd}\ \delta\mathtt{XOR}(f_{lk})\ \mathtt{AND}_{l'}\mathtt{NXOR}_k(f_{kl'}) \\ &\overset{K=2}{\to} \sum_d \tilde{x}_{nd}\tilde{u}_{ld}\ \prod_{l'}\left[1-\mathrm{abs}(u_{dl'}-z_{nl'})\right]\ . \end{aligned} \tag{D.8}$$

For an OR-NXOR machine we have

$$\begin{aligned} g_{nl} &= \sum_d \tilde{x}_{nd}\ \delta\mathtt{NXOR}(f_{lk})\ \delta\mathtt{OR}_{l'}\mathtt{NXOR}_k(f_{kl'}) \\ &= \sum_d -\tilde{x}_{nd}\ \delta\mathtt{XOR}(f_{lk})\ \mathtt{NOR}_{l'}\mathtt{NXOR}_k(f_{kl'}) \\ &\overset{K=2}{\to} \sum_d \tilde{x}_{nd}\tilde{u}_{ld}\ \prod_{l'}\mathrm{abs}(u_{dl'}-z_{nl'})\ . \end{aligned} \tag{D.9}$$

D.2 Mean-Field Reconstruction

In the following we give the general expressions for computing the mean-field estimate for predicting data from the posterior factor means.

For an AND-AND machine, we have

$$p(x_{[\mathbf{n}]}=1|.) = \prod_l \prod_{[\mathbf{n}]} \hat{f}_{l[k]} \tag{D.10}$$

For other machines in the AND-AND clan this generalises via

$$p(x_{[\mathbf{n}]}=1|.) = \overset{o}{\neg} \prod_l \overset{i}{\neg} \prod_{[\mathbf{n}]} \overset{f}{\neg} \hat{f}_{l[k]}\ . \tag{D.11}$$

For the other clans we find the following expressions. `XOR-AND`:

$$p(x_{[\mathbf{n}]} = 1|.) = \overset{o}{\neg} \sum_{l} \overset{i}{\neg} \left(\prod_{[\mathbf{n}]} \overset{f}{\neg} \hat{f}_{l[k]} \right) \prod_{l' \neq l} \left(\mathcal{J} \overset{i}{\neg} \prod_{[\mathbf{n}]} \overset{f}{\neg} \hat{f}_{l[k]} \right) . \tag{D.12}$$

For `AND-XOR`

$$p(x_{[\mathbf{n}]} = 1|.) = \overset{o}{\neg} \prod_{l} \left(\overset{i}{\neg} \sum_{k \in [k]} \hat{f}_{kl} \prod_{j \in [k]/k} (1 - \hat{f}_{jl}) \right) . \tag{D.13}$$

For `XOR-XOR`

$$p(x_{[\mathbf{n}]} = 1|.) = \overset{o}{\neg} \sum_{l} \left(\overset{i}{\neg} \sum_{k \in [k]} f_{kl} \prod_{j \in [k]/k} \left(1 - f_{jl}\right) \right) \left(\prod_{l' \neq l} \overset{i}{\neg} \mathcal{J} \sum_{k \in [k]} \left[f_{kl'} \prod_{j \in [k]/k} \left(1 - f_{jl}\right) \right] \right) . \tag{D.14}$$

The corresponding variances are simple functions of the expected value. Since $x \in \{0, 1\}$, we have $E(x^2) = E(x)$ and hence the variance, $\mathrm{Var}(x) = E(x) - E(x)^2$, is zero for $E(x) = 0$ and $E(x) = 1$ and has its maximum value at $E(x) = 0.5$.