ERASMUS UNIVERSITY ROTTERDAM

MASTER THESIS

Using Deep Exponential Families as Generative Models in Marketing Data Fusion

---

Author:
Scipio POSTMES
Student ID:
386125

Supervisor:
Prof. Dr. Dennis FOK
Second assessor:
Prof. Dr. Philip Hans Franses

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Econometrics and Management Science, Business Analytics and Quantitative Marketing programme

Erasmus School of Economics

August 28, 2019

ERASMUS UNIVERSITY ROTTERDAM

Abstract

Erasmus School of Economics

Master of Science in Econometrics and Management Science

Using Deep Exponential Families as Generative Models in Marketing Data Fusion

by Scipio POSTMES

Nowadays more and more data is collected from a broad variety of sources. Often in marketing applications the combined information of multiple sources is of interest. Combining these different sources of information is a process called data fusion. Currently, data fusion literature in marketing applications is limited and consists of simple models that have shortcomings in their performance. This work uses deep exponential families (DEF) as a generative model for the task of data fusion in a marketing application. The DEF allows to maintain correlational structures without relying on so-called statistical matching. Using both a simulation study and a private survey data set by Nielsen, it is demonstrated that the DEF strongly outperforms existing data fusion models. As such it can be clearly concluded that DEFs form a valuable instrument in the quantitative marketing literature.

Contents

List of Symbols

Variables			
$\mathbf{W}$		Set of matrices of weights	$\{K_\ell \times (K_{\ell+1}+1)\}_{\ell=1,\dots,L}$
	$\mathbf{W}_\ell$	Matrix containing weights between layers ℓ and $\ell+1$	$((K_{\ell+1}+1) \times K_\ell)$
	$W_{\ell,k}$	Vector with the weights of how $\mathbf{Z}_{\ell+1}$ explains $Z_{\ell,k}$	$(K_{\ell+1}+1)$
$\mathbf{X}$		Matrix of observed variables	$(N \times P)$
	$\mathbf{X}_f$	Matrix of fusion variables	
	$\mathbf{X}_c$	Matrix of common variables	
	$\mathbf{X}_c^{(c)}$	One configuration of the common variables	
$\mathbf{Z}$		Set of matrices of latent variables	$\{K_\ell \times N\}_{\ell=1,\dots,L}$
	$\mathbf{Z}_\ell$	Matrix of latent variables in layer ℓ	$(K_\ell \times N)$
	$Z_{\ell,k}$	Vector with k^{th} latent variable in layer ℓ	$(\times N)$
Values			
K		Total number of latent variables	$K = \sum_{\ell=1}^{L} K_\ell$
	K_ℓ	Number of latent variables in layer ℓ	
L		Number of layers in a DEF model	
N		Number of observations	
P		Number of observed variables	
Parameters			
η		Hyperparameter of first layer of latent variables, $\mathbf{Z}_L$	
θ		Set of parameters of the probability distribution $p(\mathbf{X}, \mathbf{Z})$	
ϕ		Set of parameters of the variational distribution $q(\mathbf{Z})$	
Functions			
$\alpha(\cdot)$		Log-partition function	
$\delta(\mathbf{X})$		Markov blanket of random variable X	
$S(X)$		Sigmoid function of value X	$\frac{\exp(X)}{1+\exp(X)}$
EXPFAM$[\mathbf{X}, \theta]$		Notation: X follows a distribution from the exponential family of distributions with parameter(s) θ	
$g(\cdot)$		Link function in the DEF framework	
$h(\mathbf{X})$		Base-measure	
$KL(q\|\|p)$		Kullback-Leibler divergence between q and p	
$Pa(X_i)$		Set of parents of X_i	
$T(\mathbf{X})$		Sufficient statistic of the data $\mathbf{X}$	

Chapter 1

Introduction

Come 2020, people will generate an average of 1.7 megabytes of data per second (Gantz & Reinsel, 2012). Statistics like these show the immense expansion of data generation happening all around the world. Often in data analysis applications one is interested in joint information from different sources.

Consider an example in a marketing setting. Say a marketeer is running a marketing campaign on both TV and radio and wants to estimate how many people are reached in total. In order to do so he has a data set, R, containing information on a test panel's TV viewing behaviour and some demographic information about the test individuals. Let the TV variables be denoted $\mathbf{X}_R$ and the demographic variables $\mathbf{X}_C$, with C for common variables. In addition, the marketeer has another data set, D, with information on another test panel regarding what radio channels they have listened to as well as their demographic information. The variables in this data set will be denoted as $\mathbf{X}_D$ and $\mathbf{X}_C$ respectively. It needs to be noted that the people in the two test panels are different (although there could potentially be an overlap).

Due to privacy laws, it is moreover unknown if individuals are the same across the data sets. Combining the reach numbers of the two data sets to get an estimate of how many people were reached in total is thus not straightforward, as some people might have been reached on both radio and TV and should not be double counted. Therefore it is valuable to know what radio stations the TV viewers would have listened to, had we observed it (and vice versa). The task of combining the information from data sources is known as *data fusion* (well introduced in Ipsos, 2017), in which *recipient* data set $\mathbf{X}_R$ receives information from *donor* data set $\mathbf{X}_D$, based on the shared information in the *common* data set $\mathbf{X}_C$.

Data fusion is a naturally important topic of research for many fields, such as robotics and video or image processing. Consequently, it has been much researched in those respective fields. In marketing applications, the goals – and thus technologies – of data fusion differ from those of the more common *multi-sensor data fusion* tasks in the aforementioned fields. The main difference is lies in the fact that the multi-sensor data fusion tasks concern a single entity, where marketing data fusion tries to enrich data sets of different subjects.

Currently, the go-to solution for the task of marketing data fusion relies on connecting observations from two (or more) data sets based on a closeness metric in the overlapping set of demographic variables, such as gender and home ownership. In other words, if two observations from the different data sets 'look alike' based on the common variables, we assume that they concern the same person. This approach is often referred to as the *nearest neighbours*

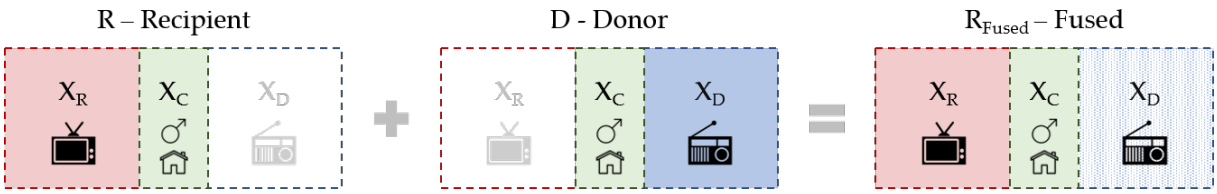

FIGURE 1.1: A visualization of the data fusion process, in which recipient data set R (on TV watching behaviour) receives information on radio listening behavior from donor data set D. The final result is a data set in which for each observation in R there exists an (estimated) value for the variables in data set D.

approach. The data fusion process is visualized in Figure 1.1, where the resulting data set contains (1) observed values for the TV variables $\mathbf{X}_R$ and demographic variables $\mathbf{X}_C$ in recipient data set R, and (2) some unobserved, but expected (or generated) values for the radio variables $\mathbf{X}_D$, which are not present in R.

This matching technique, however, becomes more and more difficult as the number of variables on which the matching occurs grows, because the bigger $\mathbf{X}_C$ gets, the more difficult it is to find an exact match. Also, the number of comparisons that needs to be made grows exponentially with the number of observations. As such there is a need for a more robust way to fuse data sets. A possible solution may be to perform data fusion using so-called *generative models*.

Generative models (GMs) aim to find a probability function that provides a feasible explanation as to why a data set is observed as is. In other words, GMs learn a probability density function $p(\mathbf{X})$ (and its parameters) that could have generated data set $\mathbf{X}$. This is illustrated in Figure 1.2. One valuable characteristic of GMs is that new data points can be generated from the probability density. As such it is possible to generate non-existent data observations, or predict values of some missing variables ($\mathbf{X}_1$) conditional on other observed variables ($\mathbf{X}_2$), based on $p(\mathbf{X}_1|\mathbf{X}_2)$. Consequently, using GMs takes away the need to find a match for each observation in a marketing data fusion tasks. Instead, GMs allow to simply perform a draw from the conditional distribution which facilitates quick and easy generation of missing data points.

In recent years GMs have been an immensely popular topic in the machine learning literature. There, much work was focused on the goal of image generation. For example, Dang

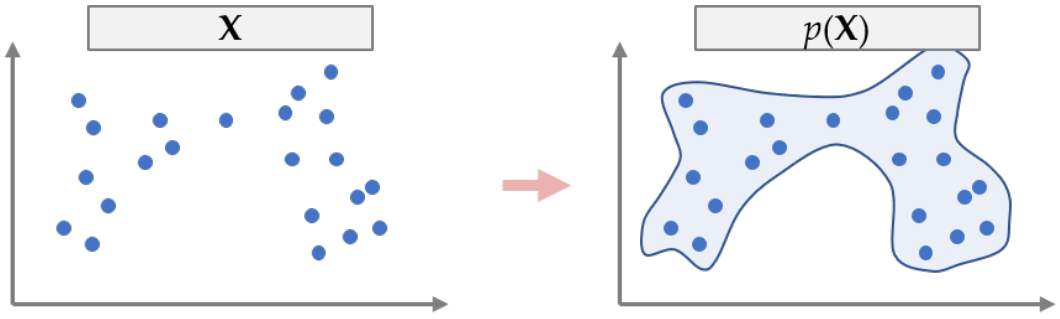

FIGURE 1.2: Illustrative example of a generative model learning a density over a data set.

et al. (2018) show the rapid advancement in image generation using a generative adversarial network (GAN), see Figure 1.3. They show that GANs are already able to create incredibly realistic faces from random noise: out of the four faces in the figure, only one is an actual photo. Work with GMs in other fields, however, is less researched and consequently less advanced.

FIGURE 1.3: One of these faces is real, the other three are generated by a generative adversarial network in work by Dang et al. (2018).

The impressive performance of GMs in the field of image generation is promising for their application on other tasks. This research, in particular, applies GMs to the task of data fusion on the Scarborough data set. The Scarborough data set is a survey conducted by Nielsen, which consists of (many) categorical variables on a broad variety of topics, including demographics, personal activities, and purchase behavior. The probabilistic nature of the methodology, in theory, is well suited to respect the dependencies between variables in the donor data set, without depending on 'matching' observations and copying the values. In this work's application a probabilistic structure is learned that explains for a group of individuals (1) the events they visited, (2) the (type of) investments they have, and (3) the (type of) organizations they contribute money to, based on some demographic information. For this purpose, a relatively new method is used as GM, named *deep exponential families* (Ranganath, Tang, Charlin, & Blei, 2014). The final goal of the research is to be able to randomly enrich samples that miss observations on some variables, such that the full observation reflects the structure of the original data set.

The structure of this work is as follows. First, related work is addressed in Chapter 2, which discusses previous work in data fusion and GMs. Consequently, the methodology used in this paper is discussed in Chapter 3. In Chapter 4 a simulation study is performed in which DEFs are used to learn the correlational structure of a data set, and afterwards the methodology is applied to the Scarborough data set by Nielsen to verify that using DEFs in data fusion tasks also works with real data, which is reported in Chapter 5. Finally, Chapter 6 concludes and discusses the implications of this work.

Chapter 2

Related Work

This work touches several areas of research, which will be addressed in this chapter. First data fusion is discussed, after which generative models are introduced. Finally recent advancements in generative modeling are reviewed.

2.1 Data Fusion

Khaleghi, Khamis, Karray, and Razavi (2013) define the task of (multi-sensor) data fusion broadly as "a technology to enable combining information from several sources in order to form a unified picture". Data fusion is an inherently important technique in many areas such as robotics, video and image processing and intelligent system design and has consequently been much researched in those respective fields. The goal and techniques in those areas, however, are often different from the needs that arise from this works task. For one, we deal with discrete variables, where the aforementioned fields often deal with continuous observations. Moreover, the marketing data fusion task comprises of enriching data sets of (likely) different subjects, where in the other fields data fusion comprises of combining information from different sources on one single entity (van der Putten & Kok, 2010). In this work, the term data fusion always stands for marketing data fusion.

In recent years data fusion has obtained (limited) traction in marketing applications. In a marketing context, data fusion is also referred to as micro data set merging, statistical record linking, multi-source imputation or ascription. Padilla, García, and Molina (2019) make use of data fusion techniques to combine information from environmental sensor data and local agricultural sales data, in order to make sales predictions for local producers. Even closer to this works application, Van Hattum and Hoijtink (2009) combine an external information source about customer gardening preferences with the internal customer data base in order to be able to perform differentiated targeting of customers. In their application they compare four methods: (1) a nearest neighbor algorithm, (2) polytomeous logistic regression of the missing data points with the common variables as explanatory information, (3) a latent class analysis based model, and (4) a mixture model approach. (1) and (2) are the most commonly applied methods in marketing data fusion tasks (van der Putten & Kok, 2010). The former depends on statistical matching, and the latter takes a probabilistic modeling approach.

Most commonly in applied data fusion a nearest neighbour approach is used, based on closeness of the common variables in a geometric hyperspace (Breur, 2011). This approach,

however, suffers the shortcomings mentioned in the introduction, where dimensionality becomes a burden in both the number of common variables and the number of observations. These shortcomings cause performance issues with large data sets in terms of the number of common variables. Moreover, the computational burden becomes heavy even with a moderate number of data points.

The polytomeous logistic regression is a probabilistic approach to the data fusion problem, like the methodology used in this work. Its performance, however, depends entirely on the direct relationships between the common variables and the fusion variables. As the draws, conditional on the common variables, are independent, further correlational information between the fusion variables is lost. It is intuitive that these direct relationships can be limited and thus restrict performance of the model.

2.2 Generative Models

Statistical learning models can generally be divided into two categories: *discriminative* and *generative* models[1] (Ng & Jordan, 2002). Say we are interested in explaining some target data set $\mathbf{Y}$ based on some observable data set $\mathbf{X}$. For example, say we want to explain the radio variables in $\mathbf{X}_D$ (from the introduction) based on the observed variables in $\mathbf{X}_C$. Discriminative models (also called conditional models) view this problem as finding the probability density function $p(\mathbf{Y}|\mathbf{X})$. In other words, discriminative models attempt to find the conditional probability distribution of $\mathbf{Y}$, after observing $\mathbf{X}$. Typical learning models that fall in this category include, amongst others, (logistic) regression, support vector machines and conditional random fields.

Generative models (GMs) take another approach to this problem, and try to determine the joint (rather than conditional) probability distribution $p(\mathbf{X}, \mathbf{Y})$. Note that in our running example this means learning a joint distribution over $\mathbf{X}_D$ and $\mathbf{X}_C$. Using a Bayesian approach, this probability can then broken up into $p(\mathbf{Y}|\mathbf{X})p(\mathbf{X})$. This approach allows to sample entire observations, and hence the name generative. Due to the fact that also the probability function for $p(\mathbf{X})$ needs to be estimated, performance of GMs in learning $P(\mathbf{Y}|\mathbf{X})$ are often worse than discriminative models. This is caused by structure assumptions that GMs need to generally make, whereas discriminative models do not require any structure assumptions (Ng & Jordan, 2002). Commonly encountered GMs are the naive Bayes classifier, Bayesian networks, and Markov random fields.

In recent years, the field of machine learning in particular has picked up an interest in GMs, as they have shown to be incredibly effective tools in the fields of image recognition or generation. The successes of GMs in the aforementioned fields have caused a surge of interest from other domains into the opportunities of GMs. In machine learning, often a particular kind of GMs, called *deep generative models* (DGMs) are used, which are simply deep models used in a generative approach. Rather than complex networks of stochastic nodes, however, machine learning often uses DGMs that are made up of a complex series of (non-)linear combinations that include some (often small) stochastic component. Currently the most used DGMs

[1] For a thorough introduction on the differences between generative and discriminative modeling, see Bernardo et al. (2007).

are the *variational autoencoder* (VAE) and the *generative adversarial net* (GAN) (Husain, Nock, & Williamson, 2019).

This research uses a DGM called *deep exponential families* (DEFs, Ranganath et al., 2014) to perform data fusion, which is treated as a generative modeling task. DEFs, as opposed to VAEs and GANs, are fully probabilistic DGMs. While their deepness allows for complex relationships that give DGMs such a strong performance, the fact that DEFs are probabilistic more naturally allows for discrete variables than the more commonly used DGMs. As marketing data sets often contain discrete variables (e.g. reach numbers, binary character traits, categorical choice variables), the DEFs pose an interesting candidate model for the data fusion task. The following sections will introduce the most popular DGMs (the VAE and the GAN) and elaborate on their shortcomings in the marketing data application.

2.3 Variational Autoencoders

Variational autoencoders (VAEs) were introduced by Kingma and Welling (2013) and have ever since attracted a lot of attention. The idea of the VAE is based on regular autoencoders (Rumelhart, Hinton, & Williams, 1988), that build a neural net on data set $\mathbf{X}$ and try to reconstruct the original data set, resulting in $\mathbf{X}'$, after passing through a reduced dimensional space, see for an example Figure 2.1[2]. In this figure, on the left data set $\mathbf{X}$ is used as input. Every node represents an observed variable. Moving to the right, linear combinations of $\mathbf{X}$ are passed through an *activation function* to form different elements of the next layer, $\mathbf{Z}_1$. The activation function, denoted as $g(\cdot)$, is chosen to be non-linear to create more complex 'hidden' values for the different layers of $\mathbf{Z}$. Common examples include the *sigmoid* function and the *softmax* function. This process is then repeated by creating $\mathbf{Z}_2$ from activated linear combinations of $\mathbf{Z}_1$, and so on, until finally $\mathbf{X}'$ is created, which is supposed to resemble $\mathbf{X}$ as much as possible.

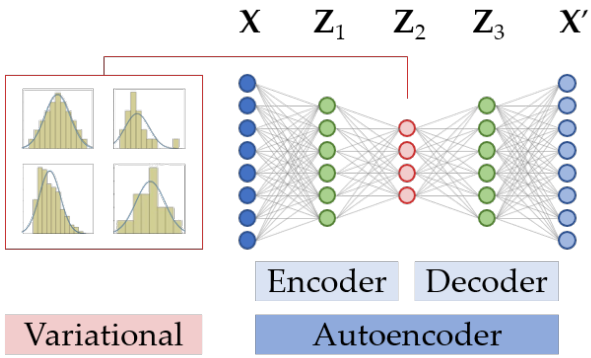

FIGURE 2.1: A five layer example of what an variational autoencoder might look like.

Because the middle layer is of a smaller dimensionality than $\mathbf{X}$, a compact, non-linear representation of the original data set is learned in the middle layer. The part which learns this

[2]The notation used in this work is largely based on the customary notation in probabilistic graphical models, and based on work by Kerbert, Postmes, Rademaker, and De Boer (2018).

representation is called the *encoder* and the second part, which tries to reconstruct the original data set, is called the *decoder*. The non-linearity, arising from the activation function, allows for richer representations than linear combinations such as principal components do.

The regular autoencoder minimizes the difference between $\mathbf{X}$ and $\mathbf{X}'$. In addition, the loss function of VAEs also includes a penalty function of how much the distribution of the elements of the middle layer differs from some prespecified probability distribution. As such, once the VAE is trained, one can sample a random configuration of the middle layer, $\mathbf{Z}$, and use it to create a randomized element of $\mathbf{X}'$.

One major drawback of applying VAEs to data fusion tasks is that, after performing the random draw in the middle layer, $\mathbf{X}'$ is created by a fully deterministic network of linear combinations of the latent variables $\mathbf{Z}$, that are passed through nonlinear activation functions. As such, the VAE is better suited for producing continuous outputs, rather than discrete variables, which are common in marketing applications. To overcome this issue, Rolfe (2016) presents a new method, using the VAE framework, to efficiently train a probabilistic model with discrete latent variables. It does so by combining latent continuous and discrete variables to accommodate a natural discrete output of the autoencoder.

A second inconvenience of the VAE model is that we do not have influence over the outcomes that we generate. That is, we cannot conditionally sample observations of the missing variables, given the variables in the overlap. When the number of variables in the overlap becomes large, this can make the process of sampling observations until one fits our search description a cumbersome one. This problem is hereditary to the model structure of the VAE and cannot be overcome.

2.4 Generative Adversarial Networks

Generative adversarial networks (GANs) are an immensely popular topic in modern day machine learning. Recent work on GANs has achieved impressive results.

GANs were introduced by Goodfellow et al. (2014), and their general idea is straightforward: We build two machine learning components, called the *generator* and the *discriminator*. It is the generator's task to randomly generate data points that are similar (even indistinguishable) to the observations in a training data set, $\mathbf{X}$. The discriminator on the other hand receives both the $\mathbf{X}$ and the machine-generated observations, $\mathbf{X}_G$. It then attempts to discriminate which observations are 'real' and which are 'fake'. These two objectives are combined into one loss function, and as such, the GAN poses creating realistic observations as a two-player game (Shen, Luo, Yan, Wang, & Tang, 2018).

With notation following Goodfellow et al. (2014), mathematically speaking the generator is a multilayer perceptron[3], $\mathcal{G}(\cdot)$, which takes random noise, Z, as input and creates the output, $X_{\mathcal{G}} = \mathcal{G}(Z|\boldsymbol{\theta}_{\mathcal{G}})$. Then both sets $\mathbf{X}$ and $\mathbf{X}_{\mathcal{G}}$ are passed to a second multilayer perceptron, the discriminator, $\mathcal{D}(\{\mathbf{X} \cup \mathbf{X}_{\mathcal{G}}\}|\boldsymbol{\theta}_{\mathcal{D}})$. The output, $\mathcal{D}(X|\boldsymbol{\theta}_{\mathcal{D}})$, for one observation, X, is then a single scalar expressing the probability of X belonging to the original data set $\mathbf{X}$. Consequently, $\mathcal{D}$ is trained to maximize the probability of assigning X to the right category - either $\mathbf{X}$ or $\mathbf{X}_{\mathcal{G}}$ - whilst

[3]Class of feedforward artificial neural network. See Bishop (2006), Chapter 5 for an introduction.

$\mathcal{G}$ is trained to minimize said probability for the observations in $\mathbf{X}_{\mathcal{G}}$. This process is visualized in Figure 2.2. The resulting loss function is given by:

$$\min_{\boldsymbol{\theta}_{\mathcal{G}}} \max_{\boldsymbol{\theta}_{\mathcal{D}}} V(\boldsymbol{\theta}_{\mathcal{D}}, \boldsymbol{\theta}_{\mathcal{G}}) = \sum_{X \in \mathbf{X}} \log(\mathcal{D}(X|\boldsymbol{\theta}_{\mathcal{D}})) + \sum_{X \in \mathbf{X}_{\mathcal{G}}} \log(1 - \mathcal{D}(\mathcal{G}[Z|\boldsymbol{\theta}_{\mathcal{G}}]|\boldsymbol{\theta}_{\mathcal{D}}) \tag{2.1}$$

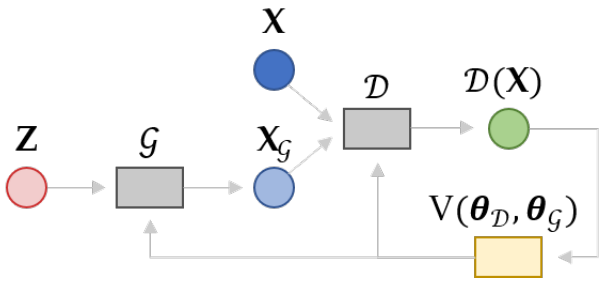

FIGURE 2.2: A schematic visualization of the working of a GAN.

The main application of GANs is image generation (Reed et al., 2016; Mao et al., 2017; Goodfellow, 2016), but recent work has also applied the methodology on the enhancement of the quality of images, called super-resolution (Ledig et al., 2017). Previously, neural nets were used to achieve state-of-the-art results in super-resolution tasks, but the remaining problem was that neural nets failed to create finer textures in the pictures. Generative models, due to their probabilistic nature, are better suited to retain a strong level of detail in the pictures. Finally, an intriguing application of GANs has been that of image style transfer, in which the style of one image (often an artwork) is combined with the content of another (often a photo). Zhang and Dana (2017) demonstrated impressive, real-time results in this field by using a GAN, some of which are displayed in Figure 2.3.

One reason GANs are not ideally suited for the task of data fusion with discrete data, is that GANs rely on the gradient estimation from the discriminator in order to learn how to slightly change the synthetic data (Chen, 2018; Goodfellow et al., 2014). Intuitively there is no way to make a 'small change' to a discrete number. No work has yet been done on applying GANs on discrete data.

Although the concept of GANs is extremely well suited for tasks such as image generation, it can be questioned whether 'being indistinguishable' is really the aim of data fusion in marketing. The discreteness of the data makes this task difficult because it strongly restricts the outcome space. It is intuitive that there is a difference between finding outlying observations from a bivariate normal distribution versus a bivariate Bernoulli distribution (which only has four possible outcomes). Also diversity in consumer behavior should be embraced, as long as it upholds the logical structure of the data. Furthermore, like VAEs, standard configurations of GANs suffer from the fact that there is no option to easily create data conditionally on some variable values. For this purpose, Mirza and Osindero (2014) define the conditional GAN. However, their application conditions MNIST data on the digit that is written, which concerns far less classes than the average data fusion task would have in their overlap. As such it is

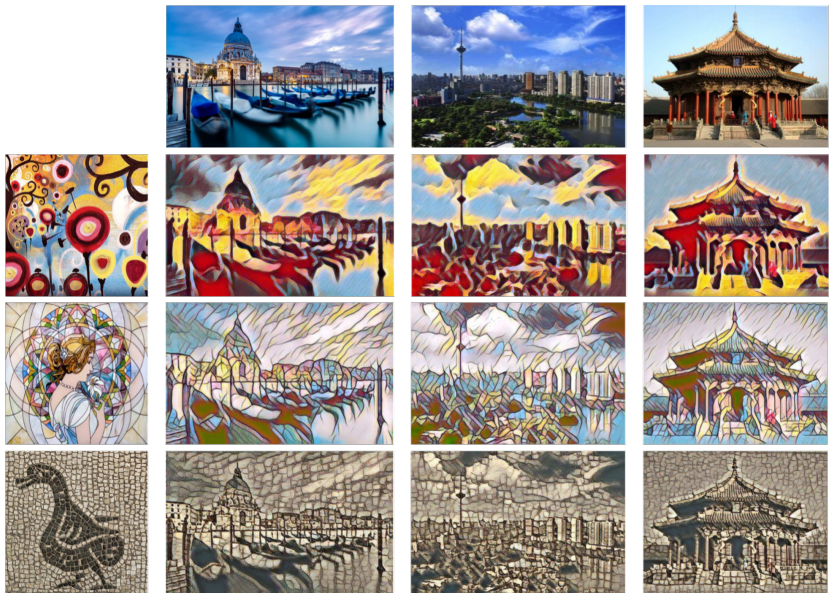

FIGURE 2.3: Photographs (top row) whose content is combined with the style of several artworks (left column) by a GAN to create pieces of arts (middle 9 pictures), in work by Zhang and Dana (2017).

highly dubitable whether the conditional GAN would be an appropriate candidate model for that task at hand.

Chapter 3

Methodology

In their recent work, Ranganath et al. (2014) introduced the concept of *deep exponential families*, henceforth referred to as DEFs. DEFs pose a very generic structure of a deep probabilistic model. A DEF consists of an observed data set, $\mathbf{X}$, and a (deep) network of hidden variables, $\mathbf{Z}$, that are assumed to follow some distribution from the exponential family of distributions. The influences between variables are modeled explicitly, and quantified by weights that are stored in $\mathbf{W}$. It is assumed that the values of $\mathbf{Z}_i$, although they are not observed, explain the observation of $\mathbf{X}_i$ for individual i. Ranganath et al. (2014) demonstrate DEFs for the task of topic detection in text analysis, but the authors explicitly express the broad range of possible applications of DEFs. These arise by virtue of the variety of options in the exponential family of distributions, including Bernoulli, Gamma, Poisson, Gaussian and categorical distributions, amongst others.

This chapter will first briefly introduce probabilistic graphical models, which form the broader class of models that DEFs fall into. It will then proceed with some introduction on the exponential family of distributions before introducing the concept of DEFs. Consequently the particular implementation of DEF used in this work, the sigmoid belief network, is introduced. Finally the problem of performing inference, some implementation details, and model evaluation are discussed.

3.1 Probabilistic Graphical Models

Probabilistic graphical models, often simply referred to as graphical models, are a popular concept in applied statistics and machine learning. The most renowned forms of graphical models are *Bayesian networks* (Pearl, 1985) and *Markov random fields* (Dobruschin, 1968), both thoroughly introduced in Bishop (2006). Less applied examples include *junction trees* (Halin, 1976), *factor graphs* (Loeliger, 2004), and *restricted Boltzmann machines* (Smolensky, 1986).

Graphical models have some desirable properties, particularly with regards to creating intuition for probability models. Bishop furthermore adds that graphical models allow to express complex computations (for performing inference and learning) in terms of graphical manipulations, in which the underlying mathematical expressions are carried along implicitly.

It is common to denote a graphical model as a set of nodes $\mathbf{V}$, which represent random variables, and a set of arcs A, that denote the dependencies in the model[1]. In the DEF scenario

[1]Notation following work on Bayesian networks by Kerbert et al. (2018).

V consists of all observed and hidden variables, **X** and **Z**, and A consists of all weights **W**. As arcs denote (direct) dependencies in graphical models, the absense of an arc implies conditional independence between variables.

With the distinctive way of modeling dependencies comes a commonly used concept in graphical models: the *Markov blanket* of a variable. A Markov blanket (a term coined by Pearl, 1988) for random variable X consists of other random variables, conditional upon which the distribution of X no longer depends on the rest of the network. That is, given variable X, its Markov blanket $\delta(X)$, and variable X_a that is not in $\delta(X)$, the following holds:

$$p(X|\delta(X), X_a) = p(X|\delta(X)) \tag{3.1}$$

This term and concept are often used in the process of performing inference in graphical models.

A common distinction between kinds of graphical models lies in whether the graphs' arcs are directed or not. DEFs are directed graphs, which brings a favorable characteristic in many forms of data analysis: *explaining away*. Explaining away, another term coined by Pearl (1988), concerns the what is called *causal asymmetry* between causes and consequences in a directed graph. Figure 3.1 shows the example used by Pearl, where a graphical model is given regarding wet grass. Two causes, sprinklers (W_1) and rain (W_2), can cause the grass to be wet (X). Wet grass, then, can lead to shiny grass (Y_1) and wet shoes (Y_2). Pearl addresses that two common causes of a consequence interact differently than two common consequences of a cause. Put differently, although the outcomes Y_1 and Y_2 support one another, the causes W_1 and W_2 compete over who explains X. This competition drives the graphical models to be more parsimonious in their representation, leading to a stronger interpretability of the model (Ranganath et al., 2014). All of these properties also hold for DEFs. Moreover, all variables in DEFs follow a distribution from the exponential family of distributions, which will be discussed in the next section.

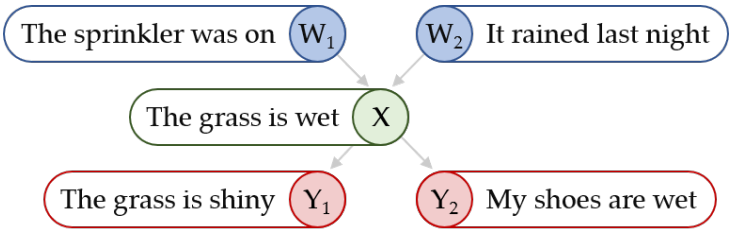

FIGURE 3.1: The explaining away example by Pearl (1988), used to explain the difference between common causes and consequences of a variable.

3.2 The Exponential Family of Distributions

The exponential family of distributions was introduced by Pitman (1936) and consists of all probability distributions that can be written in the following form[2]:

$$p(x|\eta) = h(x)\exp(\eta^T T(x) - a(\eta)) \tag{3.2}$$

Amongst the many distributions that belong to the exponential family are well-known examples such as the normal, exponential, gamma, Bernoulli and Dirichlet distributions. A sound and well-readable introduction on the concept of the exponential family of distributions is given by Vlachos (2010). The elements of (3.2) are to be interpreted as follows:

η	The **natural parameter** of the distribution, which specifies all the parameters needed for the distribution.
$T(x)$	The **sufficient statistic** of the data.
$a(\eta)$	The **log-partition function**, also the *log-normalizer* or *normalization factor*, ensures that $p(y\|\eta)$ is a proper probability distribution.
$h(x)$	The non-negative **base measure**, which is often equal to 1.

A valuable property of the members of the exponential family of distributions, which is fundamental to the DEF framework, revolves around the first moment of the probability function, for which the following holds:

$$\mathbb{E}(T(x)) = \nabla_\eta a(\eta) \tag{3.3}$$

The fact that the expectation of the sufficient statistic is equal to the gradient of the log-partition function is substantial for DEFs, which will be addressed in more detail in section 3.3.

3.3 Deep Exponential Families

Deep exponential families consist, in brief, of four elements:

1. **X**, a set of observed variables, that are assumed to follow a distribution that is part of the exponential family of distributions, of which we want to learn the probabilistic structure.
2. **Z**, a set of (layers of) hidden variables, that are assumed to follow a distribution that is part of the exponential family of distributions.
3. **W**, a set of all weights necessary to connect all the different (latent and observed) variables, as well as an intercept term.
4. $g(\cdot)$, the link function that converts previous layers to the natural parameters of the next.

As the name suggests, these networks are usually deep, meaning there is more than one latent layer. Now the DEF structure will be discussed from top to bottom, referring to Figure 3.2 as an example. The top latent layer is addressed first, which shall henceforth be denoted as $\mathbf{Z}_L$, and consists of the first latent variables, denoted $\{Z_{L,1}, \ldots, Z_{L,K_L}\}$. Here L denotes the number

[2]Notation following work by Ranganath et al. (2014).

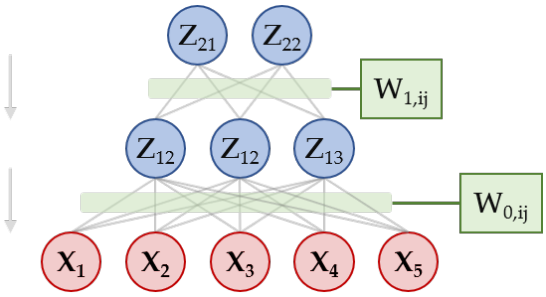

FIGURE 3.2: A small example of a possible Deep Exponential Family model, consisting on two hidden and one observed layer.

of layers, and K_L denotes the number of latent variables in the L^{th} layer. In the example, $L = 2$ and $K_L = 2$. Each latent variable has a (hidden) value for each observation in $\mathbf{X}$. The elements of $\mathbf{Z}_L$ are assumed to follow a distribution from the exponential family of distributions, with some hyperparameter η. This is denoted as $p(\mathbf{Z}_L) = \text{EXPFAM}\big[\mathbf{Z}_L, \eta\big]$. Blei (2016) clearly explains that there is a lot of freedom in picking the distributions of these latent variables, as long as they are part of the exponential family of distributions. Moreover, the distributions of the latent variables can differ between layers. Therefore we shall henceforth refer the link function g as g_ℓ, since the link function (which is dependent on the assumed distribution) can differ per layer. The values of layer $\mathbf{Z}_L$, then, are combined with the weights and passed through g_{L-1}. After that transformation, they make up the parameters of the distribution of the latent variables in the layer below, $\mathbf{Z}_{L-1}$. For $Z_{L-1,k}$ (element of $\mathbf{Z}_{L-1}$) this is denoted as[3]:

$$p(Z_{L-1,k}|\mathbf{Z}_L, W_{L-1,k}) = \text{EXPFAM}\big[Z_{L-1,k}, g_{L-1}(W_{L-1,k,0} + \mathbf{Z}'_L W_{L-1,k})\big] \tag{3.4}$$

This result is similar for all other layers of latent variables. Thus, the parameters of the distribution of latent variable $Z_{\ell,k}$ are specified by $g_\ell(W_{\ell,k,0} + \mathbf{Z}'_{\ell+1} W_{\ell,k})$, where $\ell \neq L$. In this notation, $W_{\ell,k,0}$ denotes the intercept term. Furthermore note that, by definitions embodied in the graphical model framework, the Markov blanket for $Z_{\ell,k}$ consists of the layers above and below the ℓ^{th} layer, that is:

$$\delta(Z_{\ell,k}) = \{\mathbf{Z}_{\ell+1}, \mathbf{Z}_{\ell-1}\} \tag{3.5}$$

The set of weights, $\mathbf{W}$, consists of $L-1$ elements $\mathbf{W}_\ell$. Each matrix $\mathbf{W}_\ell$ contains (1) an intercept term for each element of $\mathbf{Z}_\ell$ and (2) the weights describing how the elements of $\mathbf{Z}_{\ell+1}$ form the natural parameters of layer ℓ, and thus has dimensions $((K_{\ell+1}+1) \times K_\ell)$. The weights are assumed to follow a prior distribution, $p(\mathbf{W}_\ell)$.

Using the properties of the exponential family of distributions, discussed in Section 3.2, it can be deduced that the mean of some latent variable $Z_{\ell,k}$ is completely controlled by the gradient of the log-partition function ($a(\eta)$) and the link function g_ℓ (Brown, 1986), where η is the natural parameter of the latent variable, given by $g_\ell(W_{\ell,k,0} + \mathbf{Z}'_{\ell+1} W_{\ell,k})$. The entirety results

[3]Blei (2016) explains that taking the inner product is not necessarily the only option for combining weights with the previous latent variables. However, in the scope of this work we assume this is always the case.

in the following property in DEFs:

$$\mathbb{E}\left[T(Z_{\ell,k})|\mathbf{Z}_{\ell+1}, W_{\ell,k}\right] = \nabla_\eta a\left[g_\ell(W_{\ell,k,0} + \mathbf{Z}'_{\ell+1}W_{\ell,k})\right] \tag{3.6}$$

Finally, the probability distribution and expectation of $\mathbf{X}$, which is also part of the exponential family of distributions, are defined as follows:

$$p(X_k|\mathbf{Z}_1, W_{0,k}) = \text{EXPFAM}\left[X_k, g_0(W_{0,k,0} + \mathbf{Z}'_1 W_{0,k})\right] \tag{3.7}$$

$$\mathbb{E}[T(X_k)|\mathbf{Z}_1, W_{0,k}] = \nabla_\eta a\left[g_0(W_{0,k,0} + \mathbf{Z}'_1 W_{0,k})\right] \tag{3.8}$$

Now that the entire framework of the DEFs has been defined, the concrete implementation used in this application will be defined.

3.4 Sigmoid Belief Network

DEFs form a general framework, that can be translated into many concrete implementations by choosing distributions for the variables $\mathbf{Z}$ and $\mathbf{X}$, and defining the appropriate link functions as well as priors for the weights. One such implementation is called the *sigmoid belief network*. Sigmoid belief networks (SBNs) were introduced by Neal (1992). In SBNs all the nodes are distributed according to a Bernoulli distribution. For each node in the network, X_i, the probability parameter of the Bernoulli distribution is governed by the parents of X_i, $Pa(X_i)$. In the DEF this are the variables in the layer above X_i: $\mathbf{Z}_1$ for $\mathbf{X}$ and $\mathbf{Z}_{\ell+1}$ for $\mathbf{Z}_\ell$. The conditional probability distributions in the SBN are then defined as:

$$p(X_k|\mathbf{Z}_1, \mathbf{W}_{0,k}) = \frac{\exp\left[\left(W_{0,k,0} + \sum_{Z_j \in \mathbf{Z}_1} W_{0,k,j} Z_j\right) X_k\right]}{1 + \exp\left[W_{0,k,0} + \sum_{Z_j \in \mathbf{Z}_1} W_{0,k,j} Z_j\right]} \tag{3.9}$$

$$p(Z_{\ell,k}|\mathbf{Z}_{\ell+1}, \mathbf{W}_{\ell+1,k}) = \frac{\exp\left[\left(W_{\ell,k,0} + \sum_{Z_j \in \mathbf{Z}_{\ell+1}} W_{\ell,k,j} Z_j\right) Z_{\ell,k}\right]}{1 + \exp\left[W_{\ell,k,0} + \sum_{Z_j \in \mathbf{Z}_{\ell+1}} W_{\ell,k,j} Z_j\right]} \tag{3.10}$$

where $\mathbf{W}_{0,k}$ and $\mathbf{W}_{\ell+1,k}$ are vectors containing the weights concerning X_k and $Z_{\ell,k}$, respectively. Note that $W_{k,0}$ governs the probability parameter of X_k in the case where all parent values of X_k are equal to zero. The weights in the SBN are assigned independent and identical standard normal prior distributions. This results in an intuitive network structure, where nodes determine sequentially whether the following nodes will be activated or not. Note that the weights in $\mathbf{W}$ can be regarded as parameters in a logistic regression (McCullagh & Nelder, 1983). The entire model likelihood, as such, is defined as follows:

$$\ell(\mathbf{X}|\mathbf{Z}, \mathbf{W}) = \prod_i \left\{ \frac{\exp\left[\left(W_{i,0} + \sum_{Z_j \in Pa(X_i)} W_{i,j} Z_j\right) X_i\right]}{1 + \exp\left[W_{i,0} + \sum_{Z_j \in Pa(X_i)} W_{i,j} Z_j\right]} \right\} \tag{3.11}$$

On the topic of performing inference in SBNs, the absorption of evidence into the model quickly separates the variables into *observed* ones, for which we have evidence, and *latent* ones, for which we do not. These groups of variables are referred to as $\mathbf{X}$ and $\mathbf{Z}$, respectively, in accordance with the rest of this work. Choosing the number of latent variables in the network does not have one 'right' way, and is often subject to k-fold testing procedures. Gan, Henao, Carlson, and Carin (2015) use a number of either a quarter or half of the number of observed parameters, and Gan (2018) evaluates SBN performance on the MNIST data set, concluding that with the 784 observed variables models with 100, 200, and 500 hidden variables have similar performance. In this work the number of hidden variables is selected between 0.25 and 0.5 times the number of observed nodes.

It is well known that in large, densely connected networks the likelihood computation, $p(\mathbf{X}) = \sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z})$, becomes infeasible (Saul, Jaakkola, & Jordan, 1996). This is due to the fact that it requires summing over the $2^{|\mathbf{Z}|}$ configurations of $\mathbf{Z}$. As such it is desirable to perform inference in another way.

3.5 Performing Inference

In recent years much work has been dedicated to developing methods to perform inference even when summation over the latent configurations becomes infeasible. Two general classes of inference exist: optimization-based approaches and sampling (or Markov chain Monte Carlo) approaches. The latter are known to be unbiased, but slow in convergence, where optimization-based approaches are faster but can suffer from oversimplification of the posterior distribution of parameters (Zhang, Butepage, Kjellstrom, & Mandt, 2018). In this work we opt to use variational inference, which is an optimization-based approach.

3.5.1 Variational Inference

Variational inference (VI) is the most famous optimization-based approach, and was introduced by Jordan, Ghahramani, Jaakkola, and Saul (1999). The goal of VI, also known as *variational Bayes*, is to find the best approximation $q_\phi(\mathbf{Z})$ of the posterior distribution of the latent variables, $p_\theta(\mathbf{Z}|\mathbf{X})$. Here, θ and ϕ denote the parameters of the true and approximate probability distributions, respectively. In the two-layer SBN that is used in this work the true probability distribution, $p_\theta(\mathbf{X}, \mathbf{Z})$, and variational distribution, $q_\phi(\mathbf{Z})$, are defined as follows:

$$\begin{aligned} p_\theta(\mathbf{X}, \mathbf{Z}) &= p(\mathbf{X}|\mathbf{Z}) \times p(\mathbf{Z}) \\ &= p(\mathbf{X}|g(\mathbf{Z}_1'\mathbf{W}_0)) \times p(\mathbf{Z}_1|g(\mathbf{Z}_2'\mathbf{W}_1)) \times p(\mathbf{Z}_2|\theta_{\mathbf{Z}_2}) \qquad (3.12) \\ q_\phi(\mathbf{Z}, \mathbf{W}) &= q(\mathbf{W}_0|\phi_{\mathbf{W}_0}) \times q(\mathbf{W}_1|\phi_{\mathbf{W}_1}) \times q(\mathbf{Z}_1|\phi_{\mathbf{Z}_1}) \times q(\mathbf{Z}_2|\phi_{\mathbf{Z}_2}) \qquad (3.13) \end{aligned}$$

Note that the parameters of $p(\mathbf{X}, \mathbf{Z})$ are:

$$\theta = \left\{\mathbf{W}_0, \mathbf{W}_1, \theta_{\mathbf{Z}_2}\right\} \qquad (3.14)$$

The variational distribution uses a Bayesian approach in learning $\mathbf{W}$. This means $\mathbf{W}$ is treated as a random variable, and assigned a Normal distribution in the SBN implementation with parameters $\phi_{\mathbf{W}}$, see (3.15). These parameters are treated the same as the (independent) probabilities of $\mathbf{Z}$, $\theta_{\mathbf{Z}}$. All of these parameters are treated as point masses[4], which means that a maximum likelihood approach is used in order to learn their optimal values. This leads to the following definition of the variational parameter set ϕ:

$$\begin{aligned} \mathbf{W} &\overset{q}{\sim} N(\mu_{\mathbf{W}}, \sigma_{\mathbf{W}}) \\ \phi_{\mathbf{W}} &= \{\mu_{\mathbf{W}}, \sigma_{\mathbf{W}}\} \\ \phi &= \{\phi_{\mathbf{W}_0}, \phi_{\mathbf{W}_1}, \phi_{\theta_{\mathbf{Z}_1}}, \phi_{\theta_{\mathbf{Z}_2}}\} \end{aligned} \tag{3.15}$$

Because $\mathbf{W}$ is treated in a Bayesian way in $q(\cdot)$, it becomes part of the latent variables, $\mathbf{Z}$. Seeing as our choices for $\mathbf{Z}$ are discrete and our distributions of $\mathbf{W}$ are continuous, integrating over $q(\mathbf{Z}, \mathbf{W})$ looks as follows: $\int \sum_{\mathbf{Z}} q(\mathbf{Z}, \mathbf{W}) d\mathbf{W}$. For the sake of readability, this will henceforth be denoted as $\int\!\!\sum_{\mathbf{Z}} q(\mathbf{Z})$, where $\mathbf{W}$ is absorbed into $\mathbf{Z}$.

Another name for $q_\phi(\mathbf{Z})$ is the *variational distribution*. In its effort to get $q_\phi(\mathbf{Z})$ as close to $p_\theta(\mathbf{Z}|\mathbf{X})$ as possible, VI makes use of a decomposition of the log marginal probability distribution of $\mathbf{X}$, $p_\theta(\mathbf{X})$, as follows (Bishop, 2006)[5]:

$$\begin{aligned} \log p_\theta(\mathbf{X}) &= \mathcal{L}(q) + KL(q||p) \\ &= \int\!\!\sum_{\mathbf{Z}} q_\phi(\mathbf{Z}) \log \left\{ \frac{p_\theta(\mathbf{X}, \mathbf{Z})}{q_\phi(\mathbf{Z})} \right\} + \Big(- \int\!\!\sum_{\mathbf{Z}} q_\phi(\mathbf{Z}) \log \left\{ \frac{p_\theta(\mathbf{Z}|\mathbf{X})}{q_\phi(\mathbf{Z})} \right\} \Big) \end{aligned} \tag{3.16}$$

Here, $KL(q||p)$ denotes the Kullback-Leibler (KL) divergence between $p(\cdot)$ and $q(\cdot)$. The KL divergence is an asymmetrical measure of how much two probability distributions differ (Kullback & Leibler, 1951; Kullback, 1959). By definition the KL divergence is strictly positive and as such it is a logical consequence that the first term, $\mathcal{L}(q)$ is smaller than the likelihood of $\mathbf{X}$. As such, it is often referred to as the *evidence lower bound* (ELBO) and the goal of VI is to maximize this ELBO, thereby minimizing the KL divergence between true posterior and $q(\cdot)$ (Neiswanger, 2017). Note that the ELBO can be rewritten as:

$$\mathbb{E}_{q_\phi(\mathbf{Z})} \Big[\log p_\theta(\mathbf{X}, \mathbf{Z}) - \log q_\phi(\mathbf{Z}) \Big] \tag{3.17}$$

As the log term in the KL divergence goes to zero as $q_\phi(\mathbf{Z})$ goes to $p(\mathbf{Z}|\mathbf{X})$, it is clear that the lower bound assumes its optimum when the approximation is exactly equal to the posterior distribution of the latent variables. However, calculating the gradient of the true posterior of the latent variables is intractable due to the bottom part of (3.11), where:

$$\log(1 + \exp[W_{i,0} + \textstyle\sum_{Z_j \in Pa(X_i)} W_{i,j} Z_j])$$

[4]Or more formally: are assigned Dirac delta distributions.

[5]This formula concerns the case of discrete latent variables.

does not allow for an easy gradient calculation (the top part of the log-likelihood becomes a linear combination of latent variables). For that reason, $q_\phi(\mathbf{Z})$ is restricted to be from a certain group of distributions that leaves the gradient estimation tractable.

A common choice for $q_\phi(\mathbf{Z})$ is a factorized distribution (Ialongo et al., 2018), in which all latent variables are assumed follow independent distributions. As such, the variational probability function looks as follows (Blei, Kucukelbir, & McAuliffe, 2017):

$$q_\phi(\mathbf{Z}) = \prod_{k=1}^{K} q_{\phi,k}(\mathbf{Z}_k) \tag{3.18}$$

Here K denotes the number of latent variables.

Now, the goal is to optimize the ELBO. In order to do so efficiently, it is common to opt for a gradient descent optimization technique, which requires the calculation of the gradient of the ELBO in the trainable parameter space $\{\phi\}$ (Bingham et al., 2018; Roeder, Wu, & Duvenaud, 2017):

$$\nabla_\phi \, \mathbb{E}_{q_\phi(\mathbf{Z})} \left[\log p_\theta(\mathbf{X}, \mathbf{Z}) - \log q_\phi(\mathbf{Z}) \right] \tag{3.19}$$

Calculating the gradient can be cumbersome. Kingma and Welling (2013) have introduced the *reparameterization trick* to aid in this task, but the reparameterization trick requires continuous (differentiable) latent variables, and hence cannot be applied in the SBN application. We thus confide in the use of the REINFORCE estimator.

3.5.2 REINFORCE Estimator

The issue with the reparameterization trick is that many distributions, including all discrete ones, cannot be reparameterized. In order to arrive at a solution to this problem, first notice that our gradient (with discrete latent variables) is defined as follows:

$$\begin{aligned}
\nabla_\phi \left[\text{ELBO}\right] &= \nabla_\phi \Big[\, \mathbb{E}_{q_\phi(\mathbf{Z})} \left[\log p_\theta(\mathbf{X}, \mathbf{Z}) - \log q_\phi(\mathbf{Z}) \right] \Big] \\
&= \nabla_\phi \Big[\sum\!\!\!\!\!\!\int_{\mathbf{Z}} q_\phi(\mathbf{Z}) \left[\log p_\theta(\mathbf{X}, \mathbf{Z}) - \log q_\phi(\mathbf{Z}) \right] \Big] \\
&= \sum\!\!\!\!\!\!\int_{\mathbf{Z}} \Big[\nabla_\phi \left[q_\phi(\mathbf{Z}) \right] (\log p_\theta(\mathbf{X}, \mathbf{Z}) - \log q_\phi(\mathbf{Z})) + \\
&\qquad\qquad q_\phi(\mathbf{Z}) \nabla_\phi \left[\log p_\theta(\mathbf{X}, \mathbf{Z}) - \log q_\phi(\mathbf{Z}) \right] \Big]
\end{aligned} \tag{3.20}$$

Note that the last equality results from first switching the summation and the derivative using the *Leibniz integral rule* and then applying the chain rule. The second part of the summation in (3.20) can be approximated using draws from $q_\phi(\mathbf{Z})$. However, there exists no such approximation of the first part of the summation, because $\nabla_\phi[q_\phi(\mathbf{Z})]$ is not a valid probability density. To solve this we make use of the following property of the gradient, known as the *log-derivative trick*:

$$\nabla_\phi[q_\phi(\mathbf{Z})] = q_\phi(\mathbf{Z}) \nabla_\phi[\log q_\phi(\mathbf{Z})] \tag{3.21}$$

By using this property, we are able to rewrite the gradient of the ELBO as follows:

$$\begin{aligned}
&\nabla_\phi[\text{ELBO}] = \\
&\sum_{\mathbf{Z}} \left[\nabla_\phi[q_\phi(\mathbf{Z})](\log p_\theta(\mathbf{X},\mathbf{Z}) - \log q_\phi(\mathbf{Z})) + q_\phi(\mathbf{Z})\nabla_\phi[\log p_\theta(\mathbf{X},\mathbf{Z}) - \log q_\phi(\mathbf{Z})]\right] = \\
&\sum_{\mathbf{Z}} \left[q_\phi(\mathbf{Z})\nabla_\phi[\log q_\phi(\mathbf{Z})](\log p_\theta(\mathbf{X},\mathbf{Z}) - \log q_\phi(\mathbf{Z})) + q_\phi(\mathbf{Z})\nabla_\phi[\log p_\theta(\mathbf{X},\mathbf{Z}) - \log q_\phi(\mathbf{Z})]\right] = \\
&\mathbb{E}_{q_\phi(\mathbf{Z})}\left[\nabla_\phi[\log q_\phi(\mathbf{Z})](\log p_\theta(\mathbf{X},\mathbf{Z}) - \log q_\phi(\mathbf{Z})) + \nabla_\phi[\log p_\theta(\mathbf{X},\mathbf{Z}) - \log q_\phi(\mathbf{Z})]\right] \qquad (3.22)
\end{aligned}$$

This is known as the REINFORCE estimator[6], introduced by Williams (1992). Using the REINFORCE estimator, it has become possible to use Monte-Carlo estimation to approximate this term as follows:

$$\begin{aligned}
&\nabla_\phi[\text{ELBO}] \approx \\
&\frac{1}{N}\sum_{i=1}^{N}\left[\nabla_\phi[\log q_\phi(\mathbf{Z}_i)](\log p_\theta(\mathbf{X},\mathbf{Z}_i) - \log q_\phi(\mathbf{Z}_i)) + \nabla_\phi[\log p_\theta(\mathbf{X},\mathbf{Z}_i) - \log q_\phi(\mathbf{Z}_i)]\right] \qquad (3.23)
\end{aligned}$$

where each $\mathbf{Z}_i$ is a draw from $q(\mathbf{Z})$. It is common to use this result in the construction of the optimization objective, in which the first $(\log p_\theta(\mathbf{X},\mathbf{Z}) - \log q_\phi(\mathbf{Z}))$ term is held constant during the differentiation with regards to ϕ (Uber Technologies, 2018). This is henceforth denoted with a bar over the term. As such, the process of getting one Monte-Carlo estimate of $\nabla_\phi[\text{ELBO}]$ boils down to sampling the latent variables and computing the surrogate objective score:

$$\begin{aligned}
&\log p_\theta(\mathbf{X},\mathbf{Z}) - \log q_\phi(\mathbf{Z}) + \log q_\phi(\mathbf{Z})(\overline{\log p_\theta(\mathbf{X},\mathbf{Z}) - \log q_\phi(\mathbf{Z})}) = \\
&\log p_\theta(\mathbf{X},\mathbf{Z}) + \log q_\phi(\mathbf{Z})(\overline{\log p_\theta(\mathbf{X},\mathbf{Z}) - \log q_\phi(\mathbf{Z})} - 1) \qquad (3.24)
\end{aligned}$$

Consequently the surrogate objective would be subject to differentiation. The result is an unbiased estimate of $\nabla_\phi[\text{ELBO}]$.

3.5.3 Gradient Variance Reduction Techniques

Many probabilistic networks, including SBNs, contain non-reparameterizable distributions, for instance with discrete random variables. When that is the case the gradient of the ELBO has to be approximated using Monte-Carlo estimates. The variance of this gradient estimator, however, tends to be high. It is often even the case that the gradient variance will be so high that it leaves the estimator effectively unusable (Schulman, Heess, Weber, & Abbeel, 2015; Javed, 2018).

One common way of reducing the gradient variance makes use of *Rao-Blackwellization* of the gradient estimator (Liu, Regier, Tripuraneni, Jordan, & McAuliffe, 2018). The Rao-Blackwell theorem (Lehmann & Scheffé, 1950) uses knowledge about the dependency structure in the network to decrease the variance in the gradient estimator. Upon taking into account the factorized variational distribution, (3.18), in (3.24), it becomes clear that for each latent variable $\mathbf{Z}_k$

[6]The REINFORCE estimator is sometimes also referred to as the score function estimator or the likelihood ratio estimator.

the following term is included in the surrogate objective:

$$\log q_\phi(\mathbf{Z}_k)(\overline{\log p_\theta(\mathbf{X}, \mathbf{Z}) - \log q_\phi(\mathbf{Z})}) \tag{3.25}$$

We can remove some of the elements – those which are constant with respect to $\mathbf{Z}_k$ – of the barred part of this term whilst our estimator remains unbiased (Owen, 2013). This follows from the fact that for any constant c the following holds:

$$\mathbb{E}_{q_\phi(\mathbf{Z}_k)}\left[\nabla_\phi(\log q_\phi(\mathbf{Z}_k) \times c)\right] = 0 \tag{3.26}$$

which is a result from the fact that $q(\cdot)$ is a proper probability distribution[7]:

$$\mathbb{E}_{q_\phi(\mathbf{Z}_k)}\left[\nabla_\phi(\log q_\phi(\mathbf{Z}_k) \times c)\right] = c\sum_{\mathbf{Z}_k} q_\phi(\mathbf{Z}_k)\nabla_\phi(\log q_\phi(\mathbf{Z}_k)) = c\sum_{\mathbf{Z}_k}\nabla_\phi(q_\phi(\mathbf{Z}_k)) =$$
$$c\nabla_\phi\sum_{\mathbf{Z}_k} q_\phi(\mathbf{Z}_k) = c\nabla_\phi 1 = c \times 0 = 0$$

In particular, all elements that are not downstream of $\mathbf{Z}_k$ in the variational distribution can be removed (Schulman et al., 2015)[8].

A second method to reduce gradient variance does so by adding, rather than removing, elements. This is called adding a *data dependent baseline* (Mnih & Gregor, 2014). The method draws from the fact that (3.26) holds for any value of the constant c, due to the the fact that $q_\phi(\mathbf{Z})$ is normalized. This implies that (3.27) is also true, and thus we are free to add any constant that can reduce our variance without introducing a bias in our gradient estimator.

$$\mathbb{E}_{q_\phi(\mathbf{Z})}\left[\nabla_\phi(\log q_\phi(\mathbf{Z}_k)(\overline{\log p_\theta(\mathbf{X}, \mathbf{Z}) - \log q_\phi(\mathbf{Z})}))\right] =$$
$$\mathbb{E}_{q_\phi(\mathbf{Z})}\left[\nabla_\phi(\log q_\phi(\mathbf{Z}_k)(\overline{\log p_\theta(\mathbf{X}, \mathbf{Z}) - \log q_\phi(\mathbf{Z})} + c))\right] \tag{3.27}$$

In this work a *decaying average baseline* is used, which simply subtracts a running average of the recent values for $\log p_\theta(\mathbf{X}, \mathbf{Z}) - \log q_\phi(\mathbf{Z})$ where each value decays with a factor 0.95 per iteration (Uber Technologies, 2018). This leads the final term of the expectation in (3.27) to be centered around zero, without changing the expectation of the gradient, due to the result in (3.26). As such, the variance of the gradient estimator is decreased.

3.5.4 Optimization

In order to search the complex and high-dimensional parameter space for the best parameter values, an optimization routine has to be selected. We propose to use the AdagradRMSProp routine (Kucukelbir, Tran, Ranganath, Gelman, & Blei, 2017), which improves both the direction and the step size of the standard gradient descent scheme. The *Adagrad* routine (Amari, 1998)

[7] This is exemplified in the case of a discrete $\mathbf{Z}_k$, for continuous $\mathbf{Z}_k$ the summation is to be replaced by an integral over $\mathbf{Z}_k$.

[8] Note that in the implementation in Pyro, dependency tracking has a particular definition that differs from the 'true' analytical definition. For a discussion see *Dependency Tracking in Pyro*, 2018.

uses better informed estimates of the gradient direction by including information from past iterations' gradients in the step direction. This technique is also known as including *momentum* in the direction. Adagrad routines are known to converge well, but do not necessarily do so quickly. To solve this issue the AdagradRMSProp routine draws from work by Tieleman and Hinton (2012), who introduce *RMSProp* to dynamically control the step size of parameter updates. The size of the step in iteration i, $\boldsymbol{\rho}^{(i)}$, and the gradient direction, $\boldsymbol{g}^{(i)}$, are then defined, for the k^{th} parameter as:

$$\rho_k^{(i)} = \eta \times i^{-1/2+\epsilon} \times \left(\tau + \sqrt{s_k^{(i)}}\right)^{-1} \tag{3.28}$$

$$s_k^{(i)} = \alpha g_k^{(i)} + (1-\alpha) s_k^{(i-1)} \tag{3.29}$$

The first factor η controls the scale of the step sizes. Kucukelbir et al. suggest to empirically search for the value of η that results in the quickest convergence. In this work the value is set to 0.25 based on the simulation study results. Note that the $i^{-1/2} + \epsilon$ shrinks with a growth in i, where ϵ is set to some small value to fulfill conditions set by Robbins and Monro (1951). The value for α is set to 0.1, and governs the ratio of old and new gradient information. Finally, the last pertubation τ prevents division by zero and also down-weights early iterations. Its value is set to 1.

3.6 Implementation

In order to learn the network structure, in this thesis we propose to use Pyro; a probabilistic programming language, built on top of the PyTorch framework, introduced by Uber AI Labs (Bingham et al., 2018). Pyro is aimed at being flexible and scalable. Furthermore Pyro is built primarily to encompass the use of stochastic variational inference in richly structured probabilistic models. Pyro version 0.3.0 was used in this work.

Pyro's optimization requires a lot of memory storage due to the fact that historical parameter values are saved for the optimization routine. In the simulation study a common 16 GB RAM, Intel(R) i7 core machine is used. However, running thousands of iterations on the common machine becomes infeasible. As such, a repetitive algorithm is applied in which N parameter updates are performed, resulting in $\hat{\theta}_{DEF}^{(n)}$. Then, historical knowledge about θ is discarded and $\hat{\theta}_{DEF}^{(n)}$ is used as a starting point for the next N iterations, which result in $\hat{\theta}_{DEF}^{(n+1)}$.

Still, in the data application memory storage becomes an issue, due to the large increase in the number of parameter estimates for every iteration. As such, a remote instance from Amazon Web Service is used: an *m5.4xlarge* machine which runs a 16 core processor with 128 GB RAM[9]. This allows for larger numbers of iterations and thus to learn parameter values.

[9]For full details see https://aws.amazon.com/ec2/instance-types/.

3.7 Data Fusion with DEF

Recall from the introduction that we define three sets of variables:

$\mathbf{X}_D$	The donor data set.
$\mathbf{X}_R$	The recipient data set.
$\mathbf{X}_C$	The common variables, which are available in both data sets.

The task of data fusion consists of estimating a value for each observation i in $\mathbf{X}_R$ on each variable X in $\mathbf{X}_D$. The DEF does this by learning parameters over the donor data set, and then sampling values for the donor variables conditionally on the common variables of all observations in $\mathbf{X}_R$. This process is visualized in Figure 3.3.

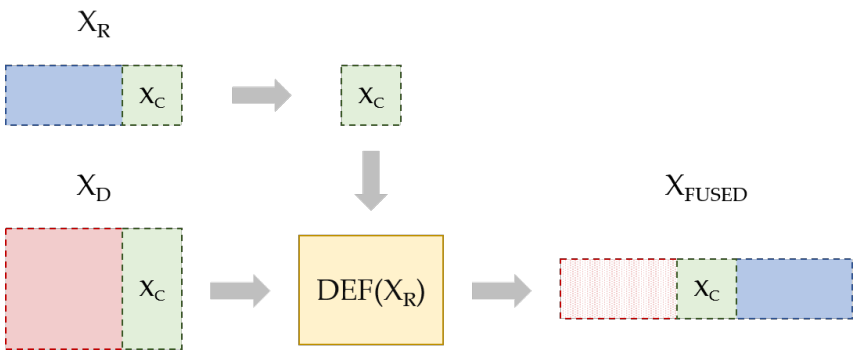

FIGURE 3.3: A schematic visualization of the how the DEF performs data fusion.

3.8 Performance Evaluation

The performance evaluation of a GM can be a tricky subject (Ermon & Grover, 2018). On the topic of image generation, Theis, Oord, and Bethge (2015) show that likelihoods can be computationally infeasible to compute in generative models. They furthermore show that being able to draw plausible[10] samples from the model does not necessarily mean the likelihood of the model is good.

In scarcely available existing applications of data fusion in marketing some focus lies in correctly predicting the fused values of customer class membership (Van Hattum & Hoijtink, 2008, 2009). Prediction is however not the main goal of this work's application. Other work focuses on maintaining correlational structures in the data set, in the form of marginal probabilities (Sun, Koehler, Remy, & Vos, 2016), conditional probabilities (Fosdick, DeYoreo, Reiter, et al., 2016), and correlation matrices (van der Putten & Kok, 2010). As these metrics are relevant to the task of this work, these will be used in the model evaluation. All metrics will be further introduced in the rest of the section.

As a comparison model, polytomeous logarithmic regressions (PLR) will be used, as suggested by Van Hattum and Hoijtink (2009). This provides a simple comparison in which logit models will be built for each individual element of the fusion variables, and and data points are

[10] An adjective aimed at the visual inspection by humans of the model-generated images.

sampled independently[11]. For the metric introduced in Section 3.8.3 (sample correlation), the performance of our method is compared to that of the most popular data fusion methodology: the nearest neighbours (NN) algorithm. In the NN method, a randomly selected observation in the training data set, with the same common variables values, is used to predict the values of the fusion variables. Now, the three evaluation metrics will be introduced.

3.8.1 Marginal Probabilities

One can easily calculate the expected marginal probabilities of the $\mathbf{X}$ variables using the values in θ and θ_{DEF}. These values can be compared to determine how well the model recovers the true marginal probabilities. Let us define the set Ω_1 and Ω_2 as the sets of all configurations of $\mathbf{Z}_1$ and $\mathbf{Z}_2$, respectively. Then, the marginal probability, conditional on $\theta^* = \{\pi^*_{\mathbf{Z}_2}, \mathbf{W}^*\}$, is defined as follows:

$$p(\mathbf{X}|\theta^*) = \sum_{\mathbf{z}_2^{(c)} \in \Omega_2} p(\mathbf{Z}_2^{(c)}|\pi^*_{\mathbf{Z}_2}) \sum_{\mathbf{z}_1^{(c)} \in \Omega_1} p(\mathbf{Z}_1^{(c)}|\mathbf{Z}_2^{(c)}, \mathbf{W}^*) \times p(\mathbf{X}|\mathbf{Z}_1^{(c)}, \mathbf{W}^*) \tag{3.30}$$

In order to compare binary probabilities, we choose to follow work by Kerbert et al. (2018) and make use of the *weighted average absolute percentage error* (WAAPE), which is defined as follows:

$$\text{WAAPE} = \frac{\sum_{\mathbf{X}_f} w_f \times |p(\mathbf{X}_f) - \hat{p}(\mathbf{X}_f)|}{\sum_{\mathbf{X}_f} w_f} \times 100\% \tag{3.31}$$

$$w_f = \sqrt{\frac{n_c}{p(\mathbf{X}_f)(1 - p(\mathbf{X}_f))}} \tag{3.32}$$

Here p denotes the true probability, $\hat{p}$ the learned probability and n_c the number of observations on which the estimate is based. Note that the weights in the WAAPE are based of the inverse of the theoretical variance of the Bernoulli probability distribution, $\text{Var}(\hat{p}(\mathbf{X})) = \frac{p(\mathbf{X})(1-p(\mathbf{X})}{n}$.

The WAAPE assigns a heavier weight to observations with more extreme possibilities, which has the following intuition: say a variable is Bernoulli distributed with probability $p = 0.01$. We would expect our estimate the be closer (in terms of absolute deviation) to this probability, than we would of a variable distributed with $p = 0.50$. The variance of the estimator takes care of this, which can be seen in Figure 3.4. Thus dividing by the standard deviation achieves exactly this desired characteristic.

3.8.2 Conditional probabilities

Ultimately, the goal of data fusion is to get a model that accurately describes the relationship between the common variables and the donor variables. Let $\mathbf{X}_f$ and $\mathbf{X}_c$ denote the fusion and common variables, respectively, and let $\mathbf{X}_c^{(c)}$ denote one configuration of the common variables.

[11] For a thorough introduction on logit models, please see Chapter 6.3 in Heij et al. (2004)

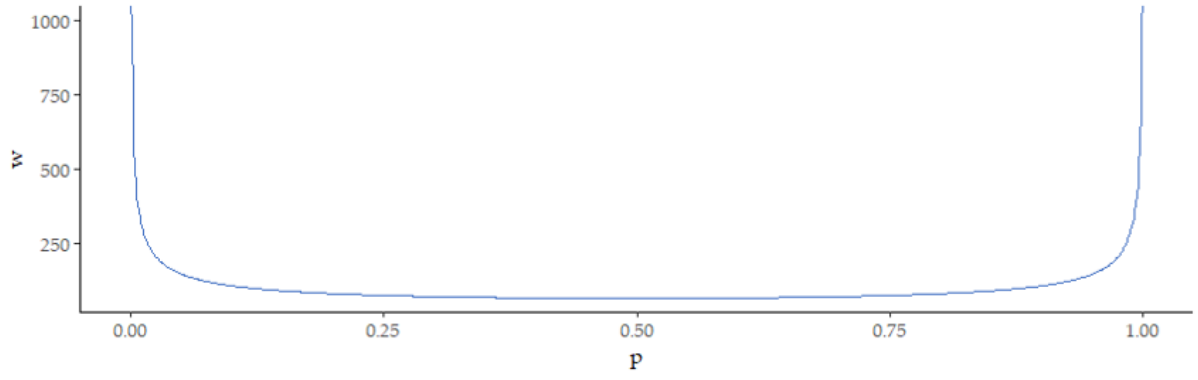

FIGURE 3.4: The WAAPE weights plotted against the true probability.

We are thus interested in learning the correct values for $p(\mathbf{X}_f|\mathbf{X}_c^{(c)})$. This can be calculated as follows[12]:

$$\begin{aligned} p(\mathbf{X}_f|\mathbf{X}_c^{(c)}) &= \sum_{\mathbf{Z}_1^{(c)} \in \Omega_1} p(\mathbf{X}_f|\mathbf{Z}_1^{(c)}, \mathbf{X}_c^{(c)}) \times p(\mathbf{Z}_1^{(c)}|\mathbf{X}_c^{(c)}) \\ &= \sum_{\mathbf{Z}_1^{(c)} \in \Omega_1} p(\mathbf{X}_f|\mathbf{Z}_1^{(c)}) \times \frac{p(\mathbf{Z}_1^{(c)}, \mathbf{X}_c^{(c)})}{p(\mathbf{X}_c^{(c)})} \\ &= \sum_{\mathbf{Z}_1^{(c)} \in \Omega_1} p(\mathbf{X}_f|\mathbf{Z}_1^{(c)}) \times \frac{p(\mathbf{X}_c^{(c)}|\mathbf{Z}_1^{(c)}) \times p(\mathbf{Z}_1^{(c)})}{p(\mathbf{X}_c^{(c)})} \\ &= \sum_{\mathbf{Z}_1^{(c)} \in \Omega_1} p(\mathbf{X}_f|\mathbf{Z}_1^{(c)}) \times \frac{p(\mathbf{X}_c^{(c)}|\mathbf{Z}_1^{(c)}) \times p(\mathbf{Z}_1^{(c)})}{\sum_{\mathbf{Z}_1^{(c)} \in \Omega_1} p(\mathbf{X}_c^{(c)}|\mathbf{Z}_1^{(c)}) \times p(\mathbf{Z}_1^{(c)})} \end{aligned} \tag{3.33}$$

All elements of (3.33) are known, if we incorporate:

$$p(\mathbf{Z}_1^{(c)}) = \sum_{\mathbf{Z}_2^{(c)} \in \Omega_2} p(\mathbf{Z}_1^{(c)}|\mathbf{Z}_2^{(c)}) \times p(\mathbf{Z}_2^{(c)}) \tag{3.34}$$

These probabilities can then be calculated using the true or estimated model parameters. As such the learning performance on the conditional probabilities can be evaluated.

3.8.3 Correlation Analysis

So far (1) the marginal probabilities and (2) the probabilities of fusion variables conditional on common variables have been taken into account. The probabilities of fusion variables, conditional on other fusion variables, however, has not been taken into account so far. The PLR method cannot learn these probabilities, as fusion variables are modeled independently of each other. Also, the nearest neighbours approach cannot calculate any probabilities. Consequently, one final metric is used in this work: the sample correlation.

[12]Note that $p(\mathbf{X}_f|\mathbf{Z}_1^{(c)}, \mathbf{X}_c^{(c)}) = p(\mathbf{X}_f|\mathbf{Z}_1^{(c)})$ because $\mathbf{Z}_1$ is the Markov blanket for all elements of $\mathbf{X}_f$.

In order to get this metric, a sampled data set, $\mathbf{X}_{DEF}$ is created using the DEF. This is done conditionally on a hold-out sample of the common variables, $\mathbf{X}_c$. In order to draw the samples using the DEF, the conditional probabilities $p(\mathbf{Z}_1^{(c)}|\mathbf{X}_c^{(c)})$ as defined in (3.33) are used to draw a configuration of the latent variables in $\mathbf{Z}_1$. Consequently, the conditional probabilities $p(\mathbf{X}_f|\mathbf{Z}_1^{(c)})$ are used to draw values for the fusion variables. For the polytomeous logistic regression model and the nearest neighbours approach these sampling approaches are straightforward.

Once the samples have been drawn, the sample correlations of the true data $\mathbf{X}$ and the generated data $\mathbf{X}_{DEF}$ can easily be calculated. If it holds that $\text{Cor}(\mathbf{X}_{DEF}) \approx \text{Cor}(\mathbf{X})$, this implies that we can conditionally sample realistic data points (in a data fusion application) whilst respecting the original data structure. This supports the conclusion that the model adequately maintains the correlational structure of the data. As such, the coefficients of both correlation matrices will be compared to one another.

Chapter 4

Simulation study

In order to assess the performance of the DEF, a simulation study is performed. For that goal a data set is generated according to the structure visualized in Figure 4.1. In this chapter the data generating process is first introduced, after which simulation results are discussed.

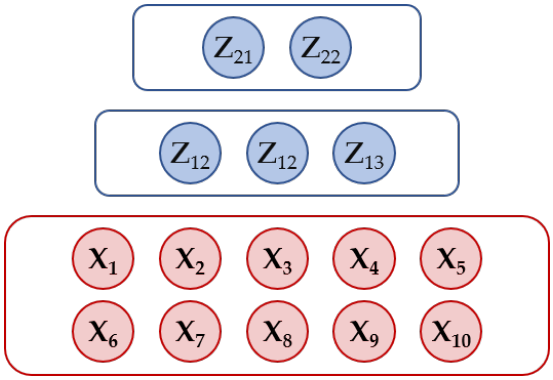

FIGURE 4.1: The structure of the data generating process, consisting of two layers of latent variables and one layer of 10 observed variables.

4.1 Data Generation Design

As mentioned, we generate a data set, $\mathbf{X}$, according to the structure in Figure 4.1. The structure consists of two layers of hidden variables, with two and three elements, and one layer of ten observed variables. The probabilities of $\mathbf{Z}_{2,1}$ and $\mathbf{Z}_{2,2}$ are set equally to $p(\mathbf{Z}_{2,1}) = p(\mathbf{Z}_{2,2}) = 0.5$. The values of the weights that determine the relationships between variables can be found in Table 4.1. Note that the intercept parameters $W_{\ell,k,0}$ are displayed in the rows preceded by c.

$\mathbf{W}_1$	$\mathbf{Z}_{1,1}$	$\mathbf{Z}_{1,2}$	$\mathbf{Z}_{1,3}$
c	-2	-1	-2
$\mathbf{Z}_{2,1}$	3	0	0
$\mathbf{Z}_{2,2}$	0	0	4

$\mathbf{W}_0$	$\mathbf{X}_1$	$\mathbf{X}_2$	$\mathbf{X}_3$	$\mathbf{X}_4$	$\mathbf{X}_5$	$\mathbf{X}_6$	$\mathbf{X}_7$	$\mathbf{X}_8$	$\mathbf{X}_9$	$\mathbf{X}_{10}$
c	-2	-3	-3	-2	-1	-2	-3	-3	-2	-1
$\mathbf{Z}_{1,1}$	4	-2	0	0	0	4	-2	0	0	0
$\mathbf{Z}_{1,2}$	0	9	9	0	0	0	9	9	0	0
$\mathbf{Z}_{1,3}$	0	0	0	3	-2	0	0	0	3	-2

TABLE 4.1: True parameter values for the simulation design. On the left $\mathbf{W}_1$ and on the right $\mathbf{W}_0$.

The simulation data has been specifically designed to contain interesting correlations, which is driven by the low intercept parameters and the extreme values for the weights. Take $\mathbf{X}_3$ for instance: $p(\mathbf{X}_3)$ is only affected by $\mathbf{Z}_{1,2}$, and thus has two possible values:

$$p(X_{3,i} = 1|Z_{1,2,i}) = \begin{cases} S(-3) = \frac{\exp(-3)}{\exp(-3)+1} = 0.047, & \text{if } Z_{1,2,i} = 0 \\ S(6) = \frac{\exp(6)}{\exp(6)+1} = 0.998, & \text{if } Z_{1,2,i} = 1 \end{cases}$$

Here, $S(\cdot)$ expresses the sigmoid function.

The generation design results in the correlational structure are given in Table 4.2. It is clear that there exist distinct and obvious correlations (such as $\text{Cor}(\mathbf{X}_2, \mathbf{X}_7) = 0.89$) as well as more subtle ones (such as $\text{Cor}(\mathbf{X}_5, \mathbf{X}_{10}) = 0.10$) in the data.

	$\mathbf{X}_1$	$\mathbf{X}_2$	$\mathbf{X}_3$	$\mathbf{X}_4$	$\mathbf{X}_5$	$\mathbf{X}_6$	$\mathbf{X}_7$	$\mathbf{X}_8$	$\mathbf{X}_9$	$\mathbf{X}_{10}$
$\mathbf{X}_1$	1	-0.03	0	0	0	0.58	-0.03	0	0	0
$\mathbf{X}_2$	-0.03	1	0.85	0	-0.01	-0.03	0.89	0.86	0.01	0
$\mathbf{X}_3$	0	0.85	1	0.01	-0.01	0	0.86	0.83	0	0.01
$\mathbf{X}_4$	0	0	0.01	1	-0.19	0	0.01	0	0.38	-0.19
$\mathbf{X}_5$	0	-0.01	-0.01	-0.19	1	0	-0.01	-0.01	-0.19	0.10
$\mathbf{X}_6$	0.58	-0.03	0	0	0	1	-0.02	0	0	0
$\mathbf{X}_7$	-0.03	0.89	0.86	0.01	-0.01	-0.02	1	0.86	0.01	0
$\mathbf{X}_8$	0	0.86	0.83	0	-0.01	0	0.86	1	0.01	0
$\mathbf{X}_9$	0	0.01	0	0.38	-0.19	0	0.01	0.01	1	-0.19
$\mathbf{X}_{10}$	0	0	0.01	-0.19	0.10	0	0	0	-0.19	1

TABLE 4.2: Correlation matrix of the observed data set, $\mathbf{X}$. Cell values equal to zero after rounding to two decimals are printed in grey.

4.2 Model Performance

After $\mathbf{X}$ has been constructed, the DEF estimates the model parameters that generated the data. It does so conditionally on the (correctly specified) structure. The learned parameter values by the DEF will be denoted as $\hat{\theta}_{DEF}$. After $20,000$ iterations, the ELBO no longer increases substantially and the learning process is stopped.

Parameter values

First, it is interesting to see whether the parameters learned by the DEF resemble the true model parameters given in Table 4.1. For $\mathbf{W}_1$, the DEF parameters are all zero, which can be easily be explained by the fact that there are no correlations between the variables in $\mathbf{Z}_1$. The influence of $\mathbf{Z}_2$ can thus be captured in the intercept parameters. The values for $\mathbf{W}_0$ and $\mathbf{W}_{0,DEF}$ are given in Tables 4.3 and 4.4, respectively. It is clear that the parameters differ strongly. This implies that the DEF has gotten stuck in a local optimum, which suggests that using different starting points can result in different parameter estimates and, thus, correlational structures. This does

not mean, however, that the model does not perform well, as multiple parameter values can create similar or even the same outcomes.

Notice that for $j = 4$, for instance, the outcome space for the true parameters depends on Z_3, and consists of $S(-2)$ and $S(3)$ for $Z_3 = 0$ and $Z_3 = 1$ respectively. For the DEF parameters, although the parameters differ, the outcome space, too, mainly depends on Z_3, and consists of probabilities close to $S(3)$ and $S(-2)$ for $Z_3 = 0$ and $Z_3 = 1$. The dependence on the states of Z_3 is thus reversed.

	$\mathbf{X}_1$	$\mathbf{X}_2$	$\mathbf{X}_3$	$\mathbf{X}_4$	$\mathbf{X}_5$	$\mathbf{X}_6$	$\mathbf{X}_7$	$\mathbf{X}_8$	$\mathbf{X}_9$	$\mathbf{X}_{10}$
c	-2	-3	-3	-2	-1	-2	-3	-3	-2	-1
$\mathbf{Z}_{1,1}$	4	-2	0	0	0	4	-2	0	0	0
$\mathbf{Z}_{1,2}$	0	9	9	0	0	0	9	9	0	0
$\mathbf{Z}_{1,3}$	0	0	0	3	-2	0	0	0	3	-2

TABLE 4.3: True parameter values for $\mathbf{W}_0$ in the simulation data set.

	$\mathbf{X}_1$	$\mathbf{X}_2$	$\mathbf{X}_3$	$\mathbf{X}_4$	$\mathbf{X}_5$	$\mathbf{X}_6$	$\mathbf{X}_7$	$\mathbf{X}_8$	$\mathbf{X}_9$	$\mathbf{X}_{10}$
c	-0.11	3.92	4.53	1.00	-3.05	-0.17	4.14	4.54	1.00	-2.90
$\mathbf{Z}_{1,1}$	2.10	-7.74	-7.30	-0.10	-0.07	1.88	-7.98	-7.35	-0.05	0.16
$\mathbf{Z}_{1,2}$	-3.06	-6.25	-6.94	0.08	0.03	-2.81	-6.62	-6.82	0.13	-0.16
$\mathbf{Z}_{1,3}$	-0.14	0.14	0.14	-3.01	2.01	-0.14	-0.04	-0.01	-2.95	1.99

TABLE 4.4: Learned parameter values for $\mathbf{W}_0$ in the simulation data set.

As the parameter values do not represent the true model parameters, it is interesting to analyze the model performance in other metrics. Therefore the fitness of the model for the data fusion task is evaluated.

Marginal probabilities

As a second performance metric, the expected values (or marginal probabilities) of $\mathbf{X}$ are compared for the true parameters and the learned parameters. Table 4.5 contains the expected values of $\mathbf{X}$ using θ, $\hat{\theta}_{DEF}$, and $\hat{\theta}_{PLR}$. It is clear that both the DEF and the PLR parameters very closely recreate the true marginal probabilities for all variables in the data set[1]. The average absolute difference between from the true values is 0.0027 and 0.0043 for the DEF and PLR, respectively. Moreover, the WAAPE values for the DEF and the PLR are 0.25% and 0.39%, respectively. The model thus performs well on the task of learning marginals.

[1]Note that PLR can learn the probabilities of the fusion variables, but not of the common variables.

	$\mathbf{X}_1$	$\mathbf{X}_2$	$\mathbf{X}_3$	$\mathbf{X}_4$	$\mathbf{X}_5$	$\mathbf{X}_6$	$\mathbf{X}_7$	$\mathbf{X}_8$	$\mathbf{X}_9$	$\mathbf{X}_{10}$
$\mathbb{E}(\mathbf{X}\|\theta)$	0.443	0.289	0.303	0.425	0.158	0.443	0.289	0.303	0.425	0.158
$\mathbb{E}(\mathbf{X}\|\hat{\theta}_{DEF})$	0.443	0.289	0.306	0.429	0.155	0.437	0.289	0.306	0.425	0.154
$\mathbb{E}(\mathbf{X}\|\hat{\theta}_{PLR})$	0.449	0.284	0.301	0.418	0.145	0.441	0.284	0.300	-	-

TABLE 4.5: Expected values of $\mathbf{X}$ using the true and learned simulation parameters.

Conditional probabilities

Next the conditional probabilities $p(\mathbf{X}_f|\mathbf{X}_c^{(c)})$ are compared for both the DEF and the PLR model. Both methods adequately manage to estimate the conditional probabilities, with *root mean squared errors* (RMSEs) of 0.007 and 0.010 for the DEF and PLR model, respectively. The full results for the conditional probabilities can be found in Appendix A. Finally, the WAAPE statistics are 0.42% and 0.55% for the DEF and the PLR, respectively. In conclusion both methods adequately capture the probabilities of the fusion variables, conditional on the common variables.

Sample correlation

In order to further analyze correlational structures between the fusion variables themselves, a 'fake' data set, $\mathbf{X}_{DEF}$ is sampled using $\hat{\theta}_{DEF}$. If the model performs well and we sample enough observations, the correlational structures of $\mathbf{X}$ and $\mathbf{X}_{DEF}$ should look alike. In order to evaluate how well the DEF has performed we thus look at the differences in the correlation matrices of $\mathbf{X}$, $\mathbf{X}_{DEF}$, $\mathbf{X}_{PLR}$, and $\mathbf{X}_{NN}$.

The performance of all methods is visualized in Figure 4.2. The correlation matrices of $\mathbf{X}$ and $\mathbf{X}_{DEF}$ resemble each other very closely (see also the result in Tables 4.2 and 4.6). The correlation matrix of the DEF has an average off-diagonal absolute deviation of 0.008. All estimates of the correlations made by the DEF are within 0.03 of the true value. The correlation matrices of $\mathbf{X}$ and $\mathbf{X}_{PLR}$ resemble each other on many points, but, as becomes clear for the Figure, some correlations disappear completely in $\mathbf{X}_{PLR}$. On average, the PLR coefficients differ by 0.216 from the true values. The large errors in the estimation occur between different $\mathbf{X}_f$ variables, which makes sense as each $\mathbf{X}_f$ variable is modeled independently in the PLR approach. The nearest neighbours (NN) model, like the DEF, performs really well, with an average off-diagonal absolute deviation of 0.006.

Following Patel et al. (2018) we analyze the correlations between (1) $\text{Cor}(\mathbf{X})$ and $\text{Cor}(\mathbf{X}_{DEF})$, (2) $\text{Cor}(\mathbf{X})$ and $\text{Cor}(\mathbf{X}_{PLR})$, and (3) $\text{Cor}(\mathbf{X})$ and $\text{Cor}(\mathbf{X}_{NN})$. These are equal to 0.998, 0.293, and 0.999 respectively. These results demonstrate the substantial improvement in performance that DEFs achieve on PLR with regards to the reconstruction of the correlational structure of the data. Furthermore it appears that DEFs and NN approaches have comparable performance.

In conclusion it is clear that the DEF model adequately captures the correlations of $\mathbf{X}$, which implies all conditional dependencies are well represented in the parameter values.

In summary, the DEF and PLR method both perform well in terms of learning the (conditional) probabilities in the data. However, the DEF strongly outperforms the PLR method

	$\mathbf{X}_1$	$\mathbf{X}_2$	$\mathbf{X}_3$	$\mathbf{X}_4$	$\mathbf{X}_5$	$\mathbf{X}_6$	$\mathbf{X}_7$	$\mathbf{X}_8$	$\mathbf{X}_9$	$\mathbf{X}_{10}$
$\mathbf{X}_1$	1	-0.02	0.01	0.01	0	0.57	-0.02	0.01	0.01	0
$\mathbf{X}_2$	-0.02	1	0.86	-0.01	0	-0.03	0.89	0.87	-0.01	0.01
$\mathbf{X}_3$	0.01	0.86	1	-0.01	0	0	0.86	0.84	0	0.01
$\mathbf{X}_4$	0.01	-0.01	-0.01	1	-0.19	0	0	-0.01	0.39	-0.18
$\mathbf{X}_5$	0	0	0	-0.19	1	0.01	-0.01	-0.01	-0.20	0.08
$\mathbf{X}_6$	0.57	-0.03	0	0	0.01	1	-0.03	0	0	0.01
$\mathbf{X}_7$	-0.02	0.89	0.86	0	-0.01	-0.03	1	0.86	0	0.01
$\mathbf{X}_8$	0.01	0.87	0.84	-0.01	-0.01	0	0.86	1	0	0.01
$\mathbf{X}_9$	0.01	-0.01	0	0.39	-0.20	0	0	0	1	-0.18
$\mathbf{X}_{10}$	0	0.01	0.01	-0.18	0.08	0.01	0.01	0.01	-0.18	1

TABLE 4.6: Correlation matrix of sampled data, $\mathbf{X}_{DEF}$. Cells values equal to zero after rounding to two decimals are printed in grey.

with regards to maintaining the correlations between fusion variables. In the latter it performs comparably to the NN methodology. The NN methodology, however,

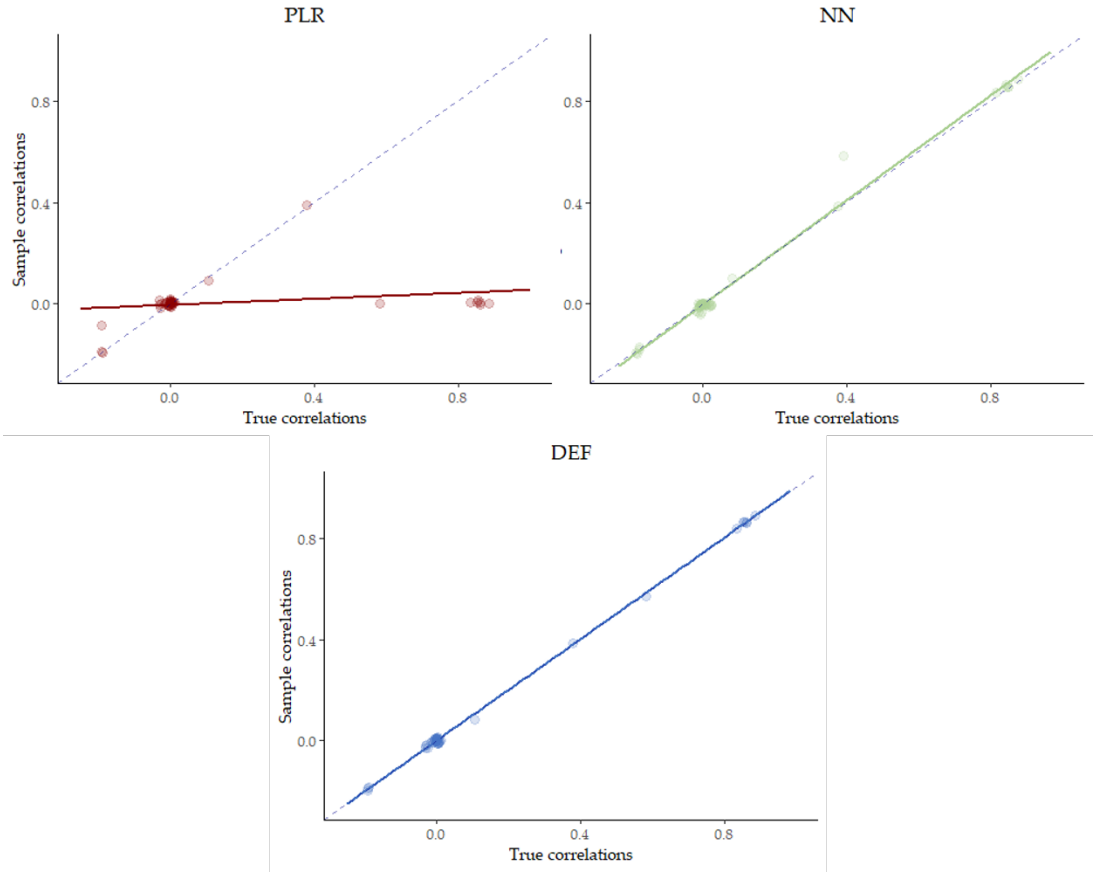

FIGURE 4.2: A scatterplot containing comparisons between the sample correlations reconstructed by the DEF (blue) and PLR (red) and their true values. Also included is a linear regression line through the points.

Chapter 5

Application

In this section the DEF is applied on a survey data set composed by Nielsen. The first section describes the data in the Scarborough data set, alongside some descriptive statistics. The second section follows up with the model performance on the application data.

5.1 Data Description

The data used in the application is a subsection of the private Scarborough survey data set, collected by Nielsen. It consists of 72,325 observations on a broad range of categorical questions regarding, amongst others, consumer behavior, lifestyle, and interests. In total, the survey consists of 7,230 binary responses. For the application in this work a subset of questions is taken into account:

	Question
	Events attended/places visited past year
(D)	Gender
	Investments household has
	Organization types to which household donated money past year
(D)	Parent of child under 18
(D)	Race
(D)	Student

Here the demographic variables are marked as such with (D). In total the specified data set consists of 52 binary responses to these questions. Before the data set is used we clean the responses that are not of interest, which in this application are the *Any* and *None* responses. After data cleaning has been done we are left with 46 responses[1].

In order for the methodology to make sense, it is desirable that (some) sizeable correlations exist in the data set. As such the correlations of all combinations of variables in the data set are analyzed. A histogram of the correlations can be found in Figure 5.1. It appears that though many variables are not strongly correlated, there are plenty of substantial correlations in the data set. As such the DEF should be able to subtract interesting relations from the data set.

[1]Note that for the PLR method we can not use all binary race variables, as they cause multicollinearity. Therefore one race variable is discarded for the PLR data set.

The models will be given a training data set to learn the parameter values, which consists of 62,325 randomly selected observations from the data set. The other 10,000 observations are used as a hold-out sample to test the sample correlations.

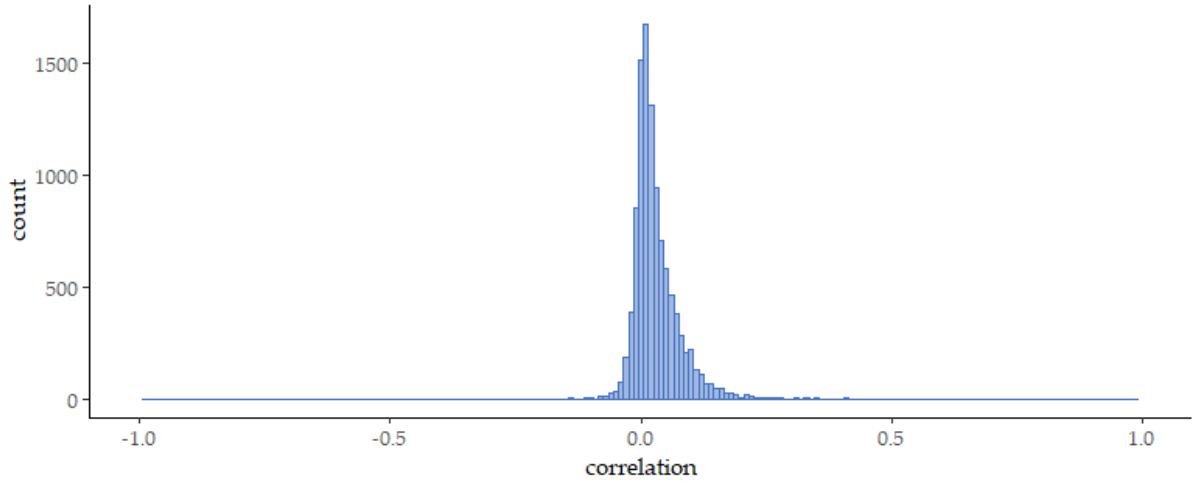

FIGURE 5.1: The histogram of correlations in the Scarborough data set show that there are enough substantial correlations for a meaningful analysis.

Following Gan et al. (2015) for the rules of thumb, we choose a model design with three latent variables in the top layer, and nine latent variables in the bottom layer.

5.2 Application Performance

In this section the DEF model performance is analyzed using the metrics specified in Section 3.8. It should be mentioned that for sampling the 10,000 observations, based on the 62,325 training observations, the DEF needs 14 seconds to compute the values in the sample. The NN method, for the same sampling task, requires 10:52 minutes, which is over 47 times as long as the DEF method. As this is a modest data size in applied quantitative marketing, this is already a substantial improvement.

Parameter values

The weights in the network cannot be compared to their true values, as the true values of the parameters do not exist. An inspection of the values of the weights shows that the bottom weights take on substantial sizes, but the top weights again are all (very close to) zero. This implies that the latent variables, as learned by the DEF, again are uncorrelated. Consequently, the deepness of the model, in this application, does not add any performance.

Marginal probabilities

Both the PLR model and the DEF model perform well on learning the marginal probabilities. The marginal probabilities found by the model are compared to the frequencies of $\mathbf{X}_f$ in the test

sample. The WAAPEs for the DEF and PLR model are very close to one another, at 0.23% and 0.21%, respectively. As these can be seen as average percentage errors, these results confirm the good performance of both methods.

The DEF reconstructs the marginal probabilities with an average absolute error of 0.0046, and the PLR also performs well with an average absolute error of 0.0031. It makes sense that the DEF model performance deteriorates somewhat with an increase in the dimensionality of the task. The PLR does not suffer from this characteristic, as it builds individual, independent models. The results, however, lead to the conclusion that also in real data applications both models manage to accurately reconstruct marginal probabilities.

Conditional probabilities

For the Scarborough data, there are no 'true' conditional probabilities, so rather, we use the hold-out sample to obtain conditional probabilities as follows:

$$p(\mathbf{X}_f|\mathbf{X}_c^{(c)}) = \frac{\sum_{i=1}^{N} I(\mathbf{X}_f = 1) I(\mathbf{X}_c = \mathbf{X}_c^{(c)})}{\sum_{i=1}^{N} I(\mathbf{X}_c = \mathbf{X}_c^{(c)})} \tag{5.1}$$

Here, $I(\cdot)$ denotes the indicator function. Since the WAAPE cannot handle probabilities equal to zero (one), we change zero (one) probabilities to a very small (large) value, 0.0001 (0.9999).

The estimated conditional probabilities are plotted against their true values in Figure 5.2. It is immediately clear from the visual inspection of the results that in the application there are more extreme outlying values than with the simulation study. This can partially be explained by the fact that some configurations of $\mathbf{X}_c$ are infrequent and thus occur less than 10 times in the test sample. This causes a tendency for the sample probabilities, as calculated in (5.1), to be less accurate. To understand the intuition behind this inaccuracy, it is important to understand that each sample probability can be regarded as a binomial trial divided over the number of experiments. The variance of this estimator is $\frac{p(1-p)}{n}$, which can become large for small values of n. The WAAPEs for the DEF and PLR are 2.09% and 2.36%, respectively. The DEF thus performs somewhat better than the PLR.

Further inspection of the conditional probabilities shows that the RMSEs of the DEF and PLR are 0.100 and 0.095, respectively. These results demonstrate that both methodologies perform worse with the Scarborough data than in the simulation environment. To dig deeper into the performance, the correlations between the true and learned conditional probabilities are calculated. These are 0.71 and 0.75 for the DEF and PLR, respectively.

Sample correlation

The correlations are estimated by sampling the fusion variables of a hold-out sample of the Scarborough data set. $10,000$ observations are conditionally sampled using the DEF and PLR model. Consequently the elements of the correlation matrices are compared to that of the true test data correlation matrix. The result of this comparison is visualized in Figure 5.3.

The figure immediately shows that the DEF maintains the correlations much better than the PLR. The PLR estimates some correlations well (those between an element of $\mathbf{X}_c$ and $\mathbf{X}_f$), but all

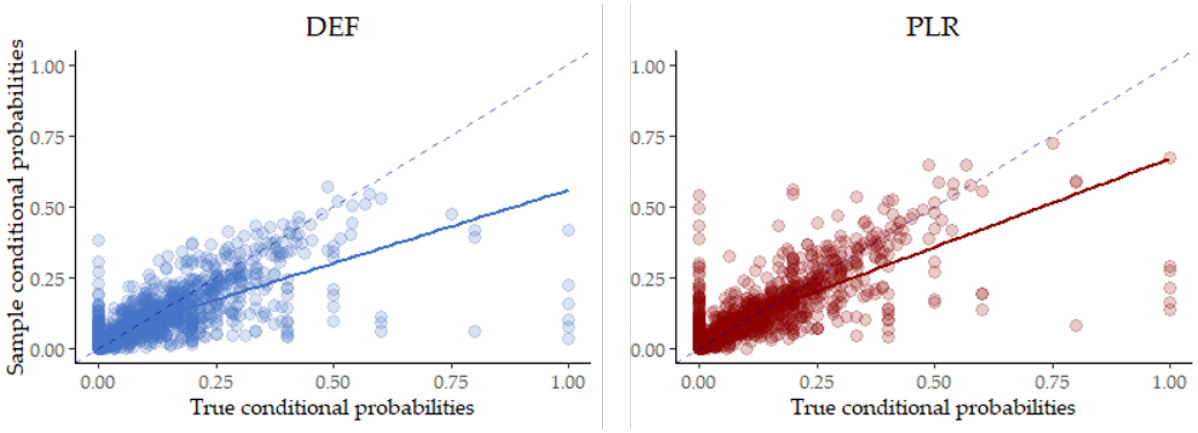

FIGURE 5.2: A scatterplot containing comparisons between the learned conditional probabilities $p(\mathbf{X}_f = 1|\mathbf{X}_c = \mathbf{X}_c^{(c)})$ reconstructed by the DEF (left) and PLR (right) and their true values. Also included is a linear regression line through the points.

correlations between fusion variables amongst one another get lost in the independency of the draws. The DEF does maintain these correlations. The correlation between the true correlation matrix and the sampled DEF matrix is 0.939, where the same correlation for the PLR is equal to 0.402. If we isolate the part of the correlation matrix that concerns correlations between different elements of $\mathbf{X}_f$, the correlations are equal to 0.944 and 0.316 for the DEF and PLR, respectively. The NN model achieves an 0.968 correlation with the true correlations. Again we conclude that the DEF model achieves similar performance to the NN approach.

In conclusion, the DEF strongly outperforms the PLR methodology and maintains the correlational structure of the data when fusing data sets. As such, it can be concluded that the DEF is more fit for being applied to marketing data fusion tasks. Also as the DEFs' performance on the Scarborough data set is highly similar to that of the NN model, it can be concluded that the two models achieve similar performance even with real data applications and uncertain model structure.

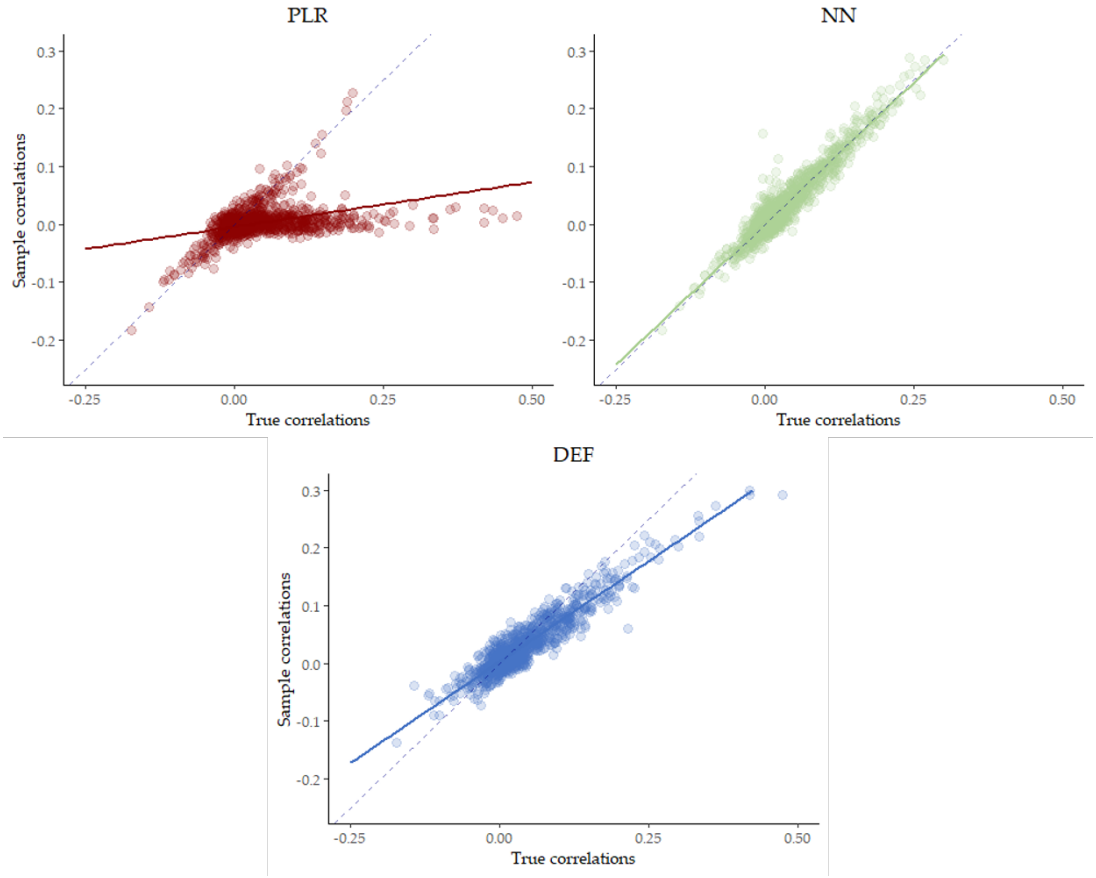

FIGURE 5.3: A scatterplot containing comparisons between the sample correlations estimated by the DEF (left) and PLR (right) and their true values. Also included is a linear regression line through the points.

Chapter 6

Conclusion

This work demonstrates the added value of using deep exponential families (DEFs) as a generative model for the task of data fusion. Following from the results on both artificial and real data, it can be concluded that using DEFs is a valuable instrument in data fusion tasks with regard to preserving (conditional) probabilities and correlational structures. As this is pivotal in marketing data fusion tasks, the DEF poses an excellent candidature for the task.

A drawback of using DEFs for data fusion is the strong increase in computational power that is required to learn parameter values in comparison to polytomeous logistic regression (PLR). The increase in computational effort is one that changes minutes of computing into days. However, the strong improvement in the correlations amongst fusion variables is in many cases worth the additional computational effort.

In comparison with the nearest neighbours (NN) model, the DEF model achieves similar performance with regards to correlational structures. The sampling process, however, is much less computationally heavy once the parameter values have been learned, making the DEF a more efficient way of sampling observations. Also, when the number of common parameters grows, the NN model becomes less feasible as the probability of finding a perfect match decreases strongly. The DEF does not suffer from this shortcoming, and thus has strong advantages over the NN approach.

The data application in Chapter 5 furthermore demonstrates that knowing the true model structure is not necessarily decisive for the DEF model to perform adequately. This work provides some rules of thumb that empirically work well with regard to latent structures.

Altogether the DEF framework appears to pose an excellent candidature for the task of data fusion, as it combines the favorable characteristics of the two most commonly used models without attracting significant shortcomings.

6.1 Discussion

Although this work has obtained favorable results, several aspects of the modeling choices have not been taken into account in this work. For one, several hyperparameters of the DEF, such as model structure and optimization parameters, have not (or hardly) been tested. They might prove sources of further improvements in both model performance. Furthermore, researching the effects of different optimization routines might increase the computational performance of the model, which could decrease one of the biggest current drawbacks of the DEF.

Especially when the number of variables grows further, this could be a valuable enrichment to this work.

Another possibility of improving computational performance lies in looking for further methods to decrease the variance of the ELBO gradient estimator. Work by Roeder et al. (2017) and Miller, Foti, D'Amour, and Adams (2017) have already obtained traction with ideas to further decrease ELBO gradient variances and could be used as an initial starting point for this work.

Appendix A

Conditional Probabilities Simulation

	θ	$\hat{\theta}_{DEF}$	$\hat{\theta}_{LR}$
$p(\mathbf{X}_1 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (0,0))$	0.443	0.441	0.450
$p(\mathbf{X}_1 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (0,1))$	0.443	0.460	0.453
$p(\mathbf{X}_1 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (1,0))$	0.443	0.446	0.446
$p(\mathbf{X}_1 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (1,1))$	0.443	0.463	0.449
$p(\mathbf{X}_2 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (0,0))$	0.289	0.286	0.282
$p(\mathbf{X}_2 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (0,1))$	0.289	0.288	0.281
$p(\mathbf{X}_2 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (1,0))$	0.289	0.291	0.288
$p(\mathbf{X}_2 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (1,1))$	0.289	0.294	0.287
$p(\mathbf{X}_3 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (0,0))$	0.303	0.305	0.299
$p(\mathbf{X}_3 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (0,1))$	0.303	0.309	0.306
$p(\mathbf{X}_3 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (1,0))$	0.303	0.307	0.300
$p(\mathbf{X}_3 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (1,1))$	0.303	0.312	0.308
$p(\mathbf{X}_4 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (0,0))$	0.293	0.300	0.301
$p(\mathbf{X}_4 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (0,1))$	0.150	0.151	0.152
$p(\mathbf{X}_4 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (1,0))$	0.663	0.663	0.665
$p(\mathbf{X}_4 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (1,1))$	0.437	0.449	0.453
$p(\mathbf{X}_5 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (0,0))$	0.206	0.201	0.203
$p(\mathbf{X}_5 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (0,1))$	0.258	0.254	0.285
$p(\mathbf{X}_5 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (1,0))$	0.072	0.072	0.077
$p(\mathbf{X}_5 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (1,1))$	0.154	0.148	0.115
$p(\mathbf{X}_6 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (0,0))$	0.443	0.432	0.442
$p(\mathbf{X}_6 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (0,1))$	0.443	0.450	0.443
$p(\mathbf{X}_6 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (1,0))$	0.443	0.439	0.439
$p(\mathbf{X}_6 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (1,1))$	0.443	0.454	0.440
$p(\mathbf{X}_7 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (0,0))$	0.289	0.286	0.279
$p(\mathbf{X}_7 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (0,1))$	0.289	0.287	0.281
$p(\mathbf{X}_7 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (1,0))$	0.289	0.293	0.289
$p(\mathbf{X}_7 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (1,1))$	0.289	0.295	0.292
$p(\mathbf{X}_8 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (0,0))$	0.303	0.303	0.296
$p(\mathbf{X}_8 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (0,1))$	0.303	0.305	0.297
$p(\mathbf{X}_8 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (1,0))$	0.303	0.309	0.305
$p(\mathbf{X}_8 = 1 \mid (\mathbf{X}_9, \mathbf{X}_{10}) = (1,1))$	0.303	0.312	0.306

TABLE A.1: Conditional probabilities of the simulation study based on true parameters and the two models.

References

Amari, S. (1998). Natural gradient works efficiently in learning. *Neural computation*, *10*(2), 251–276.

Bernardo, J., Bayarri, M., Berger, J., Dawid, A., Heckerman, D., Smith, A., & West, M. (2007). Generative or discriminative? Getting the best of both worlds. *Bayesian Statistics*, *8*(3), 3–24.

Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., ... Goodman, N. D. (2018). Pyro: Deep universal probabilistic programming. *arXiv preprint arXiv:1810.09538*.

Bishop, C. (2006). *Pattern recognition and machine learning*. Springer.

Blei, D. M. (2016). *Deep exponential families.* Retrieved March 6, 2019, from `https://www.youtube.com/watch?v=m19VZXD-S1Y`

Blei, D. M., Kucukelbir, A., & McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American Statistical Association*, *112*(518), 859–877.

Breur, T. (2011). Data analysis across various media: Data fusion, direct marketing, clickstream data and social media. *Journal of Direct, Data and Digital Marketing Practice*, *13*(2), 95–105.

Brown, L. D. (1986). Fundamentals of statistical exponential families with applications in statistical decision theory. *Lecture Notes-Monograph Series*, *9*, i–279.

Chen, K. (2018). *Generative model for text: An overview of recent advancements.* Retrieved July 29, 2019, from `https://kuanchen.netlify.com/post/nlp_generative_model/`

Dang, L., Hassan, S., Im, S., Lee, J., Lee, S., & Moon, H. (2018). Deep learning based computer generated face identification using convolutional neural network. *Applied Sciences*, *8*(12), 2610.

Dependency tracking in Pyro. (2018). `https://forum.pyro.ai/t/dependency-tracking-in-pyro/500`. (Accessed: 2019-07-12)

Dobruschin, P. L. (1968). The description of a random field by means of conditional probabilities and conditions of its regularity. *Theory of Probability and its Applications*, *13*(2), 197–224.

Ermon, S., & Grover, A. (2018). Stanford - cs 236 - deep generative models: Evaluating generative models, lecture 11 notes [powerpoint slides]. Retrieved Juli 29, 2019, from `https://deepgenerativemodels.github.io/assets/slides/cs236_lecture11.pdf`

Fosdick, B. K., DeYoreo, M., Reiter, J. P., et al. (2016). Categorical data fusion using auxiliary information. *The annals of applied statistics*, *10*(4), 1907–1929.

Gan, Z. (2018). *Deep generative models for vision and language intelligence* (Unpublished doctoral dissertation). Duke University.

Gan, Z., Henao, R., Carlson, D., & Carin, L. (2015). Learning deep sigmoid belief networks with data augmentation. In *Artificial intelligence and statistics* (pp. 268–276).

Gantz, J., & Reinsel, D. (2012). The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the Far East. *IDC iView: IDC Analyze the future*, *2007*(2012), 1–16.

Goodfellow, I. (2016). NIPS 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672–2680).

Halin, R. (1976). S-functions for graphs. *Journal of Geometry*, *8*(1-2), 171–186.

Heij, C., Heij, C., de Boer, P., Franses, P. H., Kloek, T., van Dijk, H. K., et al. (2004). *Econometric methods with applications in business and economics*. Oxford University Press.

Husain, H., Nock, R., & Williamson, R. C. (2019). Adversarial networks and autoencoders: The primal-dual relationship and generalization bounds. *arXiv preprint arXiv:1902.00985*.

Ialongo, A. D., van der Wilk, M., Hensman, J., & Rasmussen, C. E. (2018). Non-factorised variational inference in dynamical systems. *arXiv preprint arXiv:1812.06067*.

Ipsos. (2017). *Ipsos encyclopedia - data fusion.* Retrieved April 18, 2019, from `https://www.ipsos.com/en/ipsos-encyclopedia-data-fusion`

Javed, S. A. (2018). *REINFORCE vs reparameterization trick.* Retrieved July 23, 2019, from `http://stillbreeze.github.io/REINFORCE-vs-Reparameterization-trick/`

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., & Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine Learning*, *37*(2), 183–233.

Kerbert, C. C. D., Postmes, M. S., Rademaker, J., & De Boer, M. (2018). *Learning Bayesian network structure from high-dimensional survey data.* Unpublished paper, Erasmus University Rotterdam, Erasmus School of Economics.

Khaleghi, B., Khamis, A., Karray, F. O., & Razavi, S. N. (2013). Multisensor data fusion: A review of the state-of-the-art. *Information fusion*, *14*(1), 28–44.

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*.

Kucukelbir, A., Tran, D., Ranganath, R., Gelman, A., & Blei, D. M. (2017). Automatic differentiation variational inference. *The Journal of Machine Learning Research*, *18*(1), 430–474.

Kullback, S. (1959). *Information theory and statistics*. Courier Corporation.

Kullback, S., & Leibler, R. A. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, *22*(1), 79–86.

Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., ... Shi, W. (2017).

Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4681–4690).

Lehmann, E. L., & Scheffé, H. (1950). Completeness, similar regions, and unbiased estimation: Part I. *Sankhyā: The Indian Journal of Statistics (1933-1960)*, *10*(4), 305–340.

Liu, R., Regier, J., Tripuraneni, N., Jordan, M. I., & McAuliffe, J. (2018). Rao-Blackwellized stochastic gradients for discrete distributions. *arXiv preprint arXiv:1810.04777*.

Loeliger, H. (2004). An introduction to factor graphs. *IEEE Signal Processing Magazine*, *21*(1), 28–41.

Mao, X., Li, Q., Xie, H., Lau, R. Y., Wang, Z., & Paul Smolley, S. (2017, Oct). Least squares generative adversarial networks. In *The IEEE international conference on computer vision (iccv)*.

McCullagh, P., & Nelder, J. (1983). *Generalized linear models, second edition*. Chapman & Hall.

Miller, A., Foti, N., D'Amour, A., & Adams, R. P. (2017). Reducing reparameterization gradient variance. In *Advances in neural information processing systems* (pp. 3708–3718).

Mirza, M., & Osindero, S. (2014). Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

Mnih, A., & Gregor, K. (2014). Neural variational inference and learning in belief networks. *arXiv preprint arXiv:1402.0030*.

Neal, R. M. (1992). Connectionist learning of belief networks. *Artificial intelligence*, *56*(1), 71–113.

Neiswanger, W. (2017). 10-708: Variational inference: Mean field approximation, lecture 13 notes [powerpoint slides]. Retrieved June 18, 2019, from `https://www.cs.cmu.edu/~epxing/Class/10708-17/notes-17/10708-scribe-lecture13.pdf`

Ng, A. Y., & Jordan, M. I. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. In *Advances in neural information processing systems* (pp. 841–848).

Owen, A. B. (2013). *Monte Carlo theory, methods and examples*.

Padilla, W. R., García, J., & Molina, J. M. (2019). Knowledge extraction and improved data fusion for sales prediction in local agricultural markets. *Sensors*, *19*(2), 286.

Patel, S., Kakadiya, A., Mehta, M., Derasari, R., Patel, R., & Gandhi, R. (2018). Correlated discrete data generation using adversarial training. *arXiv preprint arXiv:1804.00925*.

Pearl, J. (1985). *Bayesian networks: A model of self-activated memory for evidential reasoning* (Tech. Rep.). University of California, Los Angeles, Computer Science Dept.

Pearl, J. (1988). Embracing causality in default reasoning. *Artificial Intelligence*, *35*(2), 259–271.

Pitman, E. J. G. (1936). Sufficient statistics and intrinsic accuracy. *Mathematical Proceedings of the Cambridge Philosophical Society*, *32*(4), 567–579.

Ranganath, R., Tang, L., Charlin, L., & Blei, D. (2014). Deep Exponential Families. In G. Lebanon & S. V. N. Vishwanathan (Eds.), *Proceedings of the eighteenth international conference on artificial intelligence and statistics* (Vol. 38, pp. 762–771). San Diego, California, USA.

Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., & Lee, H. (2016). Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*.

Robbins, H., & Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, 400–407.

Roeder, G., Wu, Y., & Duvenaud, D. K. (2017). Sticking the landing: Simple, lower-variance gradient estimators for variational inference. In *Advances in neural information processing systems* (pp. 6925–6934).

Rolfe, J. T. (2016). Discrete variational autoencoders. *arXiv preprint arXiv:1609.02200*.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1988). Learning representations by back-propagating errors. *Cognitive Modeling*, *5*(3), 1.

Saul, L. K., Jaakkola, T., & Jordan, M. I. (1996). Mean field theory for sigmoid belief networks. *Journal of Artificial Intelligence Research*, *4*, 61–76.

Schulman, J., Heess, N., Weber, T., & Abbeel, P. (2015). Gradient estimation using stochastic computation graphs. In *Advances in neural information processing systems* (pp. 3528–3536).

Shen, Y., Luo, P., Yan, J., Wang, X., & Tang, X. (2018, June). FaceID-GAN: Learning a symmetry three-player GAN for identity-preserving face synthesis. In *The IEEE conference on computer vision and pattern recognition (CVPR)*.

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory.

Sun, Y., Koehler, J., Remy, N., & Vos, W. (2016). Data enrichment and cross panel imputation.

Theis, L., Oord, A. v. d., & Bethge, M. (2015). A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*.

Tieleman, T., & Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, *4*(2), 26–31.

Uber Technologies. (2018). *SVI part III: Elbo gradient estimators - Pyro tutorials 0.3.1 documentation*. Retrieved July 11, 2019, from `https://pyro.ai/examples/svi_part_iii.html`

van der Putten, P., & Kok, J. N. (2010). Using data fusion to enrich customer databases with survey data for database marketing. In *Marketing intelligent systems using soft computing* (pp. 113–130). Springer.

Van Hattum, P., & Hoijtink, H. (2008). The proof of the pudding is in the eating data fusion: An application in marketing. *Journal of Database Marketing & Customer Strategy Management*, *15*(4), 267–284.

Van Hattum, P., & Hoijtink, H. (2009). Improving your sales with data fusion. *Journal of Database Marketing & Customer Strategy Management*, *16*(1), 7–14.

Vlachos, A. (2010). Notes on exponential family distributions and generalized linear models. *University of Wisconsin-Madison - Computer Science Department*. Retrieved February 16, 2019, from `http://pages.cs.wisc.edu/~andrzeje/lmml/exp-family-glms.pdf`

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, *8*(3-4), 229–256.

Zhang, C., Butepage, J., Kjellstrom, H., & Mandt, S. (2018). Advances in variational inference. *IEEE transactions on pattern analysis and machine intelligence*.

Zhang, H., & Dana, K. (2017, Mar). Multi-style generative network for real-time transfer. *arXiv e-prints*, arXiv:1703.06953.