Likelihood-based Density Estimation using Deep Architectures

by

Priyank Jaini

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2019

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Kristian Kersting Professor Computer Science Department, TU Darmstadt
Supervisor(s):	Pascal Poupart Professor School of Computer Science, University of Waterloo
	Yaoliang Yu Assistant Professor School of Computer Science, University of Waterloo
Internal Member:	Peter van Beek Professor School of Computer Science, University of Waterloo
	Jesse Hoey Associate Professor School of Computer Science, University of Waterloo
Internal-External Member:	Martin Lysy Associate Professor Dept. of Statistics and Actuarial Science, University of Waterloo

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Multivariate density estimation is a central problem in unsupervised machine learning that has been studied immensely in both statistics and machine learning. Several methods have thus been proposed for density estimation including classical techniques like histograms, kernel density estimation methods, mixture models, and more recently neural density estimation that leverages the recent advances in deep learning and neural networks to tractably represent a density function. In today's age, when large amounts of data are being generated in almost every field, it is of paramount importance to develop density estimation methods that are cheap both computationally and in memory cost. The main contribution of this thesis is in providing a principled study of parametric density estimation methods using mixture models and triangular maps for neural density estimation.

The first part of the thesis focuses on the compact representation of mixture models using deep architectures like latent tree models, hidden Markov models, tensorial mixture models, hierarchical tensor formats and sum-product networks. It provides a unifying view of possible representations of mixture models using such deep architectures. The unifying view allows us to prove exponential separation between deep mixture models and mixture models represented using shallow architectures, demonstrating the benefits of depth in their representation. In a surprising result thereafter, we prove that a deep mixture model can be *approximated* using the conditional gradient algorithm by a shallow architecture of polynomial size w.r.t. the inverse of the approximation accuracy.

Next, we address the more practical problem of density estimation of mixture models for streaming data by proposing an online Bayesian Moment Matching algorithm for Gaussian mixture models that can be distributed over several processors for fast computation. Exact Bayesian learning of mixture models is intractable because the number of terms in the posterior grows exponentially w.r.t. to the number of observations. We circumvent this problem by projecting the exact posterior on to a simple family of densities by matching a set of sufficient moments. Subsequently, we extend this algorithm for sequential data modeling using transfer learning by learning a hidden Markov model over the observations with Gaussian mixtures. We apply this algorithm on three diverse applications of activity recognition based on smartphone sensors, sleep stage classification for predicting neurological disorders using electroencephalography data and network size prediction for telecommunication networks.

In the second part, we focus on neural density estimation methods where we provide a unified framework for estimating densities using monotone and bijective triangular maps represented using deep neural networks. Using this unified framework we study the limitations and representation power of recent flow based and autoregressive methods. Based

on this framework, we subsequently propose a novel Sum-of-Squares polynomial flow that is interpretable, universal and easy to train.

Acknowledgements

I wish to thank my supervisors Pascal Poupart and Yaoliang Yu who supported me throughout my graduate studies and gave me complete freedom to choose my research. When I first started as a PhD student in 2015, Pascal told me in the first meeting that the plan for my PhD was to have fun. In that he has succeeded by making research projects and collaborations fun - I enjoyed every moment of my time as his student and the credit belongs to his kind and supportive nature. To Yaoliang, from whom I have easily learned more than anybody else throughout my graduate studies and for always being supportive, encouraging, and understanding.

Three cheers to Tom, Sajin, Camila, Chris, and Adrián. A large part of being able to enjoy my time in Waterloo and success as a graduate student was their presence and support which made life outside the university enjoyable. I believe finding such a great set of friends who can transform this (seemingly) long journey into an incredible adventure is often a die roll in which I certainly rolled a natural 20.

Many of my other friends, colleagues, and the wonderful and supportive staff at the university who made my time there comfortable and enjoyable.

And finally, to my family. Most of my achievements and success is due to their constant support and selfless love. No amount of gratitude can do justice to their contributions and support during this thesis and beyond, but I'll take this opportunity to say an insufficient thank you nevertheless.

Dedication

For my parents and sister.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Density estimation is a fundamental problem in machine learning – where we are interested to discern meaningful statistical patterns from observed data through the deployment of *learning* algorithms – that has been studied widely [Tsybakov, 2009] and is becoming even more relevant nowadays due to the availability of huge amounts of unlabeled data in various applications. A density function captures the complete underlying structure of the statistical properties of the data. A good estimate of the model of the density captures the essential structure of the data and can be used for several downstream applications that require the knowledge of the density function like inference, data generation, data completion, prediction, etc.

In the problem of density estimation, we are given access to a set of data points generated through a sampling procedure (or true distribution $p(\cdot)$) and we are interested in estimating the density function that can explain the data set i.e. given a data set consisting of observations $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$, we want to estimate the probability density function $\hat{p}(\cdot)$ at any arbitrary point $\mathbf{x}$ such that it matches closely with the original density $p(\mathbf{x})$.

While powerful sampling techniques like Markov Chain Monte-Carlo method [Metropolis et al., 1953] facilitate evaluating of probabilities, enables sampling from target density, compute expectations and be used in constructing tests for model against the actual process using two-sample tests, there are many applications where knowing the exact form of the density is vital. Some examples of these applications include inference in sequential data modeling for tasks like activity recognition based on smartphone sensors, sleep stage classification based on electroencephalography(EEG) data for prediction of neurological disorders, and the prediction of the direction of future packet flows between a pair servers in telecommunication networks. These applications have been explored in more

detail in this thesis in Chapter 3. Additionally, an explicit form of the density is crucial in other applications like compression, Bayesian inference, maximum likelihood training, and as sub-component for other models e.g. variational inference and importance sampling. Other applications that are intimately related with density estimation methods are clustering, classification, estimation of density level-sets and dimension reduction techniques. I discuss some these applications briefly below.

Compression [Shannon, 1948, MacKay, 2003]:Modelling densities and compressing data are very closely related as the density at the point indicates the number of bits required to optimally compress the information up to a desired precision. Formally, paraphrasing the formulation given by [Shannon, 1948], the information content $I(\mathbf{x})$ in encoding some message $\mathbf{x}$ within a pre-defined precision given by a ball $\mathcal{B}(\mathbf{x})$ centered at $\mathbf{x}$ is given by:

$$I(\mathbf{x}) = -\log \int_{\mathcal{B}(\mathbf{x})} p(\mathbf{x}) \mathrm{d}\mathbf{x} \approx -\log p(\mathbf{x}) - \log |\mathcal{B}(\mathbf{x})| \tag{1.1}$$

Furthermore, a data compression technique specifies a density model. If $q(\mathbf{x})$ is the density model used by the data compressor and $p(\mathbf{x})$ is the true density, then the expected number of bits wasted given this data compression method is:

$$-\mathbb{E}_{p(\mathbf{x})}[-\log p(\mathbf{x}) - \log |\mathcal{B}(\mathbf{x})|] + \mathbb{E}_{p(\mathbf{x})}[-\log q(\mathbf{x}) - \log |\mathcal{B}(\mathbf{x})|] = \mathsf{KL}(p||q) \tag{1.2}$$

Thus, capturing the density model more accurately leads to better data compression. Arithemetic codes [MacKay, 2003] are an example of a method that can achieve near perfect compression. Recently, an invertible neural network called integer discrete flows [Hoogeboom et al., 2019] was proposed for lossless compression by learning the density over high-dimensional data. In Chapter 4, I will provide a unified framework for density estimation using invertible neural networks that helps to approximate a high-dimensional density arbitrarily well.

Bayesian Inference [Bishop, 2006]: Bayesian inference methods help to encode and update one's beliefs about a quantity of interest $\boldsymbol{\theta}$ given certain measurements (or observations) $\mathbf{x}$ through density functions using Bayes' rule. More precisely, if our belief about a quantity $\boldsymbol{\theta}$ is modeled using a density function $p(\boldsymbol{\theta})$ (known as the prior) and the statistical dependence between $\boldsymbol{\theta}$ and $\mathbf{x}$ is captured by the conditional density $p(\mathbf{x}|\boldsymbol{\theta})$ (known as the likelihood), then we can evaluate the change in the beliefs $\boldsymbol{\theta}$ after making observations $\mathbf{x}$ via Bayes' rule by:

$$p(\boldsymbol{\theta}|\mathbf{x}) \propto p(\boldsymbol{\theta}) p(\mathbf{x}|\boldsymbol{\theta}) \tag{1.3}$$

The model for the updated beliefs after observing $\mathbf{x}$ is called the posterior. In Bayesian inference it is important to be able to evaluate the density functions representing the prior, likelihood and the posterior from data.

In practice, certain parametric assumptions are made on the prior and the likelihood functions to make Bayesian learning amenable since the precise form of the likelihood function or a reasonable prior are unknown. One of the main challenges of Bayesian learning is that in many cases the posterior distribution becomes very complex to evaluate as more information becomes available e.g. for mixture models, the number of terms in the posterior grows exponentially w.r.t. the data. Therefore, algorithms aim to construct an approximate posterior distribution by either minimizing some distance metric between the true posterior and the approximate posterior or by sampling from the true posterior. In this thesis, I will propose an online and distributed moment matching algorithm for approximate Bayesian learning for mixture models (and more generally for probabilistic graphical models) and demonstrate its utility in multiple applications.

Importance Sampling [Kahn and Harris, 1951, Kahn, 1955]: Many applications require the computation of the expectation of a function of a random variable i.e. $\mathbb{E}[f(\mathbf{X})]$. Usually, this can be done easily using Monte Carlo methods. However, consider a situation where the function $f(\mathbf{x})$ is nearly equal to 0 outside some region $\mathcal{B}$ but $P(\mathbf{X} \in \mathcal{B})$ is also very small. This can easily happen if the set $\mathcal{B}$ is in the tail region of the random variable $\mathbf{X}$. An application of a plain Monte Carlo method in this case will result in a majority of samples outside the region $\mathcal{B}$ where $f(\mathbf{x})$ is close to 0. Such problems routinely arise in areas like Bayesian inference, financial risk modelling and rare event simulation in insurance, high energy physics etc.. Thus, to solve this problem, it is required that we are able to sample from the region of interest $\mathcal{B}$. In importance sampling, samples are generated from this region by learning a density that over-weights the important region ($\mathcal{B}$ here).

Formally, the problem is to evaluate $\mu = \mathbb{E}_p[f(\mathbf{X})] = \int_{\mathcal{D}} f(\mathbf{x}) \cdot p(\mathbf{x}) \mathrm{d}\mathbf{x}$ where $p(\mathbf{x})$ is a density over the data-set $\mathcal{D} \subseteq \mathbb{R}^d$. Let $q(\mathbf{x})$ be a probability density function on $\mathbb{R}^d$, then

$$\mu = \int_{\mathcal{D}} \frac{f(\mathbf{x})p(\mathbf{x})}{q(\mathbf{x})} q(\mathbf{x}) \mathrm{d}\mathbf{x} = \mathbb{E}_q\Big[\frac{f(\mathbf{x})p(\mathbf{x})}{q(\mathbf{x})}\Big] \tag{1.4}$$

The factor $p(\mathbf{x})/q(\mathbf{x})$ is called the likelihood ratio - this multiplicative adjustment to the function f compensates for sampling from q instead of p. The density q is called the proposal (or importance) distribution and p is called the nominal distribution. Hence, the estimation of q makes this otherwise infeasible problem amenable to Monte-Carlo methods.

Other applications [Chacón and Duong, 2018, Jordan et al., 1999, Kingma and Welling, 2014]: Density estimation methods can be employed for tasks of clustering where the population clusters of the domain X can be defined by the *density modes* resulting in a neat clustering strategy that automatically determines the number of clusters unlike hierarchical clustering or k-means. Similarly, density estimation methods are vital in density level-set estimation which is useful for visualizing high-density regions, estimating the support of the variables and useful in detecting outliers.

Several other methods use density estimation as a sub-component e.g. in variational inference [Jordan et al., 1999], evaluation of the approximate posterior requires a density model over the parameters of interest whereas variational autoencoders [Kingma and Welling, 2014] require density functions as components for the prior and the encoder. Furthermore, density models are helpful as objectives for training paradigms or comparison of different models by evaluating densities on held-out test data-sets. In such scenarios, a density function is still useful even if the exact density function is not required for the task.

1.1 Preliminaries

1.1.1 Density function

Let us begin by formally defining a probability density function. Consider a random variable $\mathsf{X} \subseteq \mathbb{R}^d$ in a d-dimensional space. Informally, a probability density $p(\mathbf{x})$ at a point $\mathbf{x} \in \mathbb{R}^d$ can be thought of as the quantitative measure of the number of times samples are generated in a ball of infinitesimal volume (by some generating mechanism) near the point $\mathbf{x}$.

Let $\mathcal{B}_\epsilon(\mathbf{x})$ be a ball of radius ϵ centered at $\mathbf{x}$. The probability density at $\mathbf{x}$ is the ratio between the probability that a point falls within the ball $\mathcal{B}_\epsilon(\mathbf{x})$ and the volume of the ball as the radius of the ball shrinks to 0 i.e.

$$p(\mathbf{x}) = \lim_{\epsilon \to 0} \frac{Pr\big(\mathbf{z} \in \mathcal{B}_\epsilon(\mathbf{x})\big)}{\mathcal{V}_{\mathcal{B}_\epsilon(\mathbf{x})}} \tag{1.5}$$

Therefore, a probability density function is a function from a d-dimensional space to the density at a point i.e. $p : \mathbb{R}^d \to \mathbb{R}_+$.

I now formally define a probability density function [1] as required in the thesis' framework: Let $F(\mathsf{X} \in A) = Pr(\mathsf{X} \in A)$ be defined on all Lebesgue measurable subsets A of $\mathbb{R}^d$ and be absolutely continuous w.r.t. Lebesgue measure. The function $p(\cdot)$ is called a density function if $p(\mathbf{x}) \geq 0,\ \forall \mathbf{x} \in \mathbb{R}^d$ and $\int_{\mathbb{R}^d} p(\mathbf{x})\mathrm{d}\mathbf{x} = 1$. Furthermore, the relationship between $p(\cdot)$ and F is given by: $\int_A p(\mathbf{x})\mathrm{d}\mathbf{x} = F(\mathsf{X} \in A)$ for all Lebesgue measurable sets $A \subseteq \mathbb{R}^d$.

1.1.2 Curse of Dimensionality & Common Assumptions

High dimensional density estimation is a hard problem due to the *curse of dimensionality* which refers to the phenomenon that the amount of data required to generalize accurately grows exponentially as the number of dimension (or features) grows. Curse of dimensionality introduces sparseness in the data i.e. as the number of dimensions grow, the data available becomes sparse making the accurate estimation of density functions difficult. Consider the example of two-dimensional space represented by a unit square. The average of such a feature space is the center of the square and all points within a unit distance lie in a unit circle. The volume (in this case area) of this unit circle covers most of the area of the square ($\approx 79\%$ area). However, now let the number of dimensions grow to d. The volume of a unit hyper-cube of dimension d is still 1, however the volume inscribed by a d-dimensional hyper-sphere of radius r is:

$$\mathcal{V}_d = \frac{\pi^{\frac{d}{2}}}{\Gamma\left(\frac{d}{2}+1\right)} r^d \tag{1.6}$$

Thus, the probability of a point falling inside a ball $\mathcal{B}_r(\mathbf{x})$ of radius r centered at a point $\mathbf{x}$ such that the ball is completely inside the cube is:

$$Pr\big(\tilde{\mathbf{x}} \in \mathcal{B}_r(\mathbf{x})\big) = \mathcal{V}_d = \frac{\pi^{\frac{d}{2}}}{\Gamma\left(\frac{d}{2}+1\right)} r^d \tag{1.7}$$

We require training data points inside such a ball with a small enough radius r to estimate the density $p(\mathbf{x})$ at a point $\mathbf{x}$. But, no matter how large we make the radius r, $\mathcal{V}_d$ approaches zero as we make the dimensions d larger i.e. to say that even for radii larger than the side of the cube, almost all such balls will be empty (giving no data points). To put this into

[1] I will use probability density function and density function interchangeably to mean the same thing throughout the thesis.

perspective: for estimating density with $r = 0.01$ and $d = 10$, we will require more than 10^{16} data-points.

While the curse of dimensionality might seem like an insurmountable problem, in practice, several generic assumptions are encoded in the models to scale density estimation to high-dimensions and work around the curse of dimensionality.

Lower dimensionality of data distribution: The first common assumption used is that the actual data distribution has a lower dimension structure intrinsically i.e. the density function is truly on a lower dimensional manifold than the dimensions of observations me make to describe it. Consider the example of natural images where intuitively the natural scene can only vary in certain semantic and meaningful ways rather than each pixel varying arbitrarily. Such lower dimension structures is captured in practice by either using dimensionality reduction techniques that retain maximum information but reduce the dimensionality of the data or by introducing information bottlenecks in the structure of the model.

Symmetry, Smoothness, and Independence: We often assume that data in the real world has symmetries e.g. in an image, a cat moved by a few pixels will still be an image of a cat (translation symmetry). Reordering a dataset will still describe the same dataset leading to symmetry in order or in audio streams, playing a piece faster will still describe a similar audio stream giving scale symmetry. Such symmetries are often encoded in model designs using for example convolutions and multi-scale architectures. Symmetries can also be employed for data augmentation.

Furthermore, a very common assumption made in density estimation is the smoothness of the density function i.e. if $\|\mathbf{x}-\mathbf{y}\|$ is small then the difference of densities at these points is also small i.e. $\|p(\mathbf{x}) - p(\mathbf{y})\|$ is also small. Smoothness assumptions help to interpolate over regions with no training data instead of giving them zero density.

Finally, in some domains we can consider that certain features are independent of other features. For example, in sequential data modeling one can assume that the current observations are only dependent on observations until a few seconds ago and not on all the data stream.

1.2 Methods for density estimation

Density estimation is a problem that has been studied widely and for a long time. The first paper dealing with the problem of statistical density estimation was published more than 60 years ago [Rosenblatt, 1956] but the earliest paper mentioning the problem of density estimation was by Fix and Hodges in 1951 [Fix and Hodges, 1951] dealing with discrimination problems. However, the earliest attempts to estimate the probability density functions seems to have been undertaken by Karl Pearson in his series of papers [Pearson, 1902a, Pearson, 1902b] published more than a century ago where the system of densities are modeled as the set of solutions to the equation:

$$\frac{\partial p}{\partial x} = \frac{(x - a_0)p}{a_1 + a_2 x + a_3 x^2} \tag{1.8}$$

where a_0, a_1, a_2, a_3 are constants and can be shown to be expressible as the first four moments of the density p. Therefore, the estimation procedure is to compute the sample moments from which the estimates of a_0, a_1, a_2, a_3 are determined to solve the differential equation to get the density estimate $\hat{p}$. Since then several methods have been developed for density estimation and these can be classified broadly into two groups: Parametric density estimation and Non-Parametric density estimation.

Parametric Methods: In these methods, prior assumptions are made about the form of the underlying density function. For example, in a parametric approach one can assume the form of density to be Gaussian distribution. In this case, density estimation procedure would require to estimate the mean and covariance matrix of the Gaussian. More flexibility can be conferred on parametric methods by using stronger parametric models like mixture of Gaussians. Another popular approach is to consider functional forms (e.g. affine transforms, higher degree polynomials or a deep neural network) on transformations that map a simple base distribution to a complex target distribution. This approach is broadly used for neural density estimation.

Non-Parametric Methods: Non-parametric approaches on the other hand do not make such assumptions about the density function and allow the data to drive the estimation process more directly. Kernel density estimation method is an example of non-parametric methods for density estimation. Other examples include histograms and order statistic density estimation. I will now discuss a few popular approaches for density estimation like histograms, mixture models, kernel density estimation, and neural density estimation.

1.2.1 Histograms

Histograms give an accurate representation of the distribution of numerical data and were first introduced by Karl Pearson in his work [Pearson, 1894]. Histograms [Pearson, 1894, Freedman et al., 1998, Scott, 2015] are the simplest form of density estimation method based on the approach of dividing the domain (or data points) in to bins and counting the number of samples that are in each bin. The probability density is then obtained as the fraction of points in a particular bin and the total number of samples and the bin width. Formally, if the bin width is w and the total number of samples are n, then the density estimate is given by:

$$\hat{p}(\mathbf{x}) = \frac{n_t}{n \cdot w}, \qquad \text{for} \quad \mathbf{x}_t \leq \mathbf{x} < \mathbf{x}_{t+1} \tag{1.9}$$

where $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$ are the end points of the t^{th} bin with width $w = \mathbf{x}_{t+1} - \mathbf{x}_t$. Since histograms directly use the data to estimate the density without any parametric assumptions, they fall under the category of non-parametric density estimation. The width of the bins w determines the granularity of the estimated density. However, there is a bias-variance trade-off controlled by the bin width: a histogram with large bin width (i.e. small number of bins) may underfit whereas a histogram with very fine bin width may not generalize to unseen data.

Histograms are attractive density estimation methods since they are easy to implement, provide interpretable and straight-forward results and also result in a valid density function that integrates to one over the entire domain. However, they suffer from several drawbacks. Histogram methods result in density functions that are non-smooth: the estimated density function has zero derivative almost everywhere except at bin transitions where the derivative is infinite. Furthermore, these methods are overly sensitive to choice of bin locations (even if the width is fixed) and can severely affect the resulting estimated density function. Moreover, the choice of bin location is a free parameter that is independent of the underlying data. In practice, histograms are mostly used for visualization purposes for very low dimensional problems (usually less than five dimensions) when there is enough data available.

1.2.2 Kernel Density Estimation

Kernel density estimation methods have a rich literature but the first ideas can be traced back to the work of Fix and Hodges [Fix and Hodges, 1951] followed by the very influential works of Rosenblatt [Rosenblatt, 1956] and Parzen [Parzen, 1962]. Other influential works

include the contributions of H. Akaike collected in the book [Akaike, 1974], [Epanechnikov, 1969], [Wand and Jones, 1994], [Nadaraya, 1964, Watson, 1964] among others. I refer the readers to the excellent book by A. Tsybakov [Tsybakov, 2009][Chapter 1] for a holistic history and development of non-parametric methods for density estimation methods. Kernel density estimation intuitively estimates a smooth version of the data distribution. Let $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\} \subseteq \mathbb{R}^d$ be a dataset. Then, the empirical distribution (denoted here by $p_0(\mathbf{x})$) of the data set is:

$$p_0(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} \delta(\mathbf{x} - \mathbf{x}_i) \tag{1.10}$$

i.e. an equally weighted mixture of n Dirac delta distributions located at the training points in the data set $\mathcal{D}$. A smooth version of this empirical distribution can be estimated and turned into a density function by substituting the Dirac delta function with a density function $K_w(\mathbf{z})$ called the smoothing kernel given by:

$$K_w(\mathbf{z}) = w^{-d} \tilde{K}\left(\frac{\mathbf{z}}{w}\right) \tag{1.11}$$

where $\tilde{K}$ is a density function bounded from above and $w > 0$ is the *width* of the kernel such that:

$$\lim_{w \to 0} \; K_w(\mathbf{z}) \to \delta(\mathbf{z}) \tag{1.12}$$

Thus, a kernel density estimate can now be defined as:

$$\hat{p}_w(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} K_w(\mathbf{x} - \mathbf{x}_i) \tag{1.13}$$

Similar to histograms, the width parameter w controls the degree of smoothness and bias-variance trade-off. Kernel density estimation has several advantages mainly because of its desirable asymptotic properties: kernel density estimators are *consistent* provided that w does not shrink too fast compared to n and *unbiased* if in the limit $w \to 0$. This can be seen from the following [Tsybakov, 2009]:

$$\begin{aligned}
\text{Var}(\hat{p}_w(\mathbf{x})) &= \frac{1}{n^2} \sum_{i=1}^{n} \text{Var}_{p(\mathbf{X}_i)}\Big(K_w(\mathbf{x} - \mathbf{X}_i)\Big) \\
&\leq \frac{1}{n} \mathbb{E}_{p(\mathbf{Z})}[K_w(\mathbf{x} - \mathbf{Z})] \\
&\leq \frac{\sup_{\mathbf{Z}} \tilde{K}^2(\mathbf{Z})}{n w^{2d}}
\end{aligned}$$

Thus, $\text{Var}(\hat{p}_w(\mathbf{x}))$ approaches zero if the number of data points n goes to infinity and w approaches 0 at a rate slower than $n^{-\frac{1}{2d}}$.

Unbiasedness can be seen from the fact that $\hat{p}_w(\mathbf{x})$ converges to the empirical distribution $p_0(\mathbf{x})$ as $w \to 0$ and the empirical distribution $p_0(\mathbf{x})$ is an unbiased estimate of the true density $p(\mathbf{x})$.

Another advantage of kernel density estimation method and non-parametric methods in general is that they do not require training. However, these methods suffer from a large memory and evaluation cost which increases linearly with the size of the data. Parametric methods on the other hand have a fixed memory and evaluation cost.

1.2.3 Mixture Models

In parametric density estimation, we usually specify a density function $q(\mathbf{x}; \varphi)$ with a fixed number of parameters φ. The aim is then to estimate these parameters φ such that the estimate $q(\mathbf{x}; \varphi)$ is close to the true density $p(\mathbf{x})$ under a suitable distance metric. A trivial choice of $q(\mathbf{x}; \varphi)$ is a simple parametric family of distributions e.g. Gaussian with mean $\boldsymbol{\mu}$ and covariance Σ. However, such models are restrictive e.g. a single Gaussian distribution cannot model multi-model density functions.

A natural solution to this is to consider a mixture of density functions as a parametric model i.e.

$$f(\mathbf{x}; \boldsymbol{\varphi}) = \sum_{j=1}^{m} w_j \; q(\mathbf{x}; \varphi_j), \qquad \sum_{j=1}^{m} w_j = 1 \quad \text{and} \quad w_j \geq 0, \; \forall j \in [m] \tag{1.14}$$

In this case, $f(\mathbf{x}; \boldsymbol{\varphi})$ is a mixture model with parameters $\boldsymbol{\varphi} = \{w_j, \varphi_j\}_{j=1}^{m}$. Mixture models are powerful density estimation methods that can approximate any density arbitrarily well given enough mixture components [McLachlan and Peel, 2004a].

The idea of mixture models and identification of the components of mixtures and its parameters have been talked about for as far back as 1846 [McLachlan and Peel, 2004a] although seemingly a common reference is the work of Karl Pearson [Pearson, 1894] who explicitly addressed the problem of decomposition in characterizing forehead to body length ratios in female crabs using non-normal attributes whose motivation was the work presented in [Tarter and Lock, 1893]. Although Pearson's work was pivotal to address the problem of distinct sub-populations and using moment matching tools to demonstrate the flexibility of mixture models, it required solving a ninth-degree polynomial which was computationally infeasible at the time. The advent of modern computers, computational power, and

the introduction of techniques like maximum likelihood estimation [Fisher et al., 1920][2] accelerated considerable research in this area. Mixture models have subsequently found applications in fields like natural language processing, speech synthesis, fisheries research, agriculture, botany, economics, medicine, genetics, finance, geology etc. [McLachlan and Peel, 2004a].

A large part of this thesis is devoted to density estimation using mixture models, their compact representation through deep architectures, associated theoretical properties and online and distributed estimation of such mixture models. These are discussed in detail in Chapter 2 and Chapter 3.

1.2.4 Neural Density Estimation

Neural density estimation is a parametric density estimation method that uses a function $f(\cdot)$ parameterized by a deep neural network to estimate a density function i.e. given an input $\mathbf{x} \in \mathbb{R}^d$, the function returns the density at the point $\mathbf{x}$.

I provide a simple example for neural density estimation: Let $p(\mathbf{x})$ be a d-dimensional density function. The density can be written as a factored product of conditionals and marginals as follows:

$$p(\mathbf{x}) = p(x_1) \cdot \prod_{i=2}^{d} p(x_i|x_{<i}), \tag{1.15}$$

where $x_{<i} = \{x_1, x_2, \cdots, x_i\}$. Let us model each conditional $p(x_i|x_{<i})$ using an affine transformation whose parameters are obtained from a feed-forward neural network i.e.

$$Pr(Xv_i < x_i|x_{<i}) = \sigma\Big(a_i(x_{<i}) \cdot x_i + b_i(x_{<i})\Big) \tag{1.16}$$

where $\sigma(\cdot)$ is the sigmoid function and $(a_i, b_i) = \Phi_i(x_{<i}\ ; \mathbf{w}_i)$ are the coefficients obtained from a neural network Φ_i that take as input $x_{<i}$ and parameters $\mathbf{w}_i$ and returns as outputs the tuple (a_i, b_i). Thus, estimating the weights $\mathbf{W} = \{\mathbf{w}_i\}_{i=1}^{d}$ of the network provides a density function over the variable X. This formulation was first proposed in the seminal paper of [Neal, 1992a].

Neural density estimation methods for explicitly representing the density function have become very popular recently with various advances that use affine transformations as in

[2]For interested readers, the article by Stephen M. Stigler gives a nice history of the development of Maximum Likelihood Estimation Method.

[Kingma et al., 2016, Papamakarios et al., 2017, Dinh et al., 2017, Dinh et al., 2015, Kingma and Dhariwal, 2018], polynomial transformations as proposed in [Jaini et al., 2019], deep sigmoidal functions [Huang et al., 2018] and splines [Durkan et al., 2019].

The main challenge in neural density estimation is to design the transformations $f(\cdot)$ such that evaluation of the density can be done exactly. Furthermore, it is desirable that the transformation $f(\cdot)$ is such that it is *universal* i.e. it can approximate any density function arbitrarily well.

A major contribution of this thesis (in Chapter 4) is providing a unified framework for neural density estimation using the notion of *triangular maps*. We use this framework to provide a modular proof and required conditions for constructing universal neural density estimation methods. As a consequence of this framework, we unveil a new method called Sum-of-Squares polynomial flows that we show are universal and interpretable.

1.3 Contributions

I reviewed some popular methods for parametric density estimation like mixture models and neural density estimation in Section 1.2 which are used widely for problems in unsupervised machine learning [McLachlan and Peel, 2004a, Oord et al., 2016, Kingma and Dhariwal, 2018]. The major contributions of this thesis is in developing methods for parametric density estimation.

The first contribution detailed in Chapter 2 addresses the *representation, separation,* and *approximation* properties of mixture models. We prove that several popular unsupervised learning models like latent tree graphical models including special cases like hidden Markov Models, tensorial mixture models, hierarchical tensor formats and sum-product networks provide a compact representation of density mixtures. Subsequently, we formally establish the precise relationships between these models shedding light on their similarities and differences. Based on this connection, we provide a unified treatment of notion of exponential separation in *exact* representation size between deep mixture architectures (or deep mixture model) and shallow mixture architectures. Practically though, an *exact* representation of deep mixture model is an overkill and it suffices to *approximate* a given density mixture with reasonable accuracy. In this spirit, we show a surprising result that the conditional gradient algorithm can approximate any mixture within ϵ accuracy by combining $O(1/\epsilon^2)$ "shallow" architectures. We support our theoretical findings with experiments on synthetic and real datasets confirming the benefits of depth in density estimation using mixture models.

Chapter 2 demonstrates the powerful benefits of representing mixture models using deep hierarchical architectures. However, a major problem in practice for density estimation using any representation of mixture models is fast and robust estimation of its parameters that can handle streaming data. Bayesian learning provides a robust framework for parameter estimation that lends itself naturally to both online and distributed computation. However, exact Bayesian learning of the parameters of a mixture model is intractable due to exponential increase in the number of mixture terms w.r.t. the data. In Chapter 3, we propose an online and distributed algorithm called Bayesian Moment Matching algorithm for Gaussian mixture models that approximates the exact Bayesian posterior by projecting it on to a family of tractable distributions after each observation by matching a set of sufficient moments between the exact and the approximate posterior.

Subsequently (also in Chapter 3), we extend this online and distributed Bayesian Moment Matching algorithm for the task of sequential data modeling using transfer learning where we consider the problem of inferring a sequence of hidden states associated with a sequence of observations produced by an individual within a heterogeneous population. Instead of learning a single sequence model for the population (which does not account for variations within the population), we learn a set of basis sequence models based on different individuals. The sequence of hidden states for a new individual is inferred in an online fashion by estimating a distribution over the basis models that best explain the sequence of observations of this new individual. We explain this in the context of hidden Markov models with Gaussian mixture models that are learned based on streaming data by online Bayesian moment matching algorithm. We apply the resulting transfer learning algorithm based on Bayesian Moment Matching on three real-world applications including activity recognition based on smartphone sensors, sleep stage classification based on electroencephalography (EEG) data for prediction of neurological disorders, and the prediction of the direction of future packet flows between a pair of servers in telecommunication networks.

In Chapter 4, we turn our focus to neural density estimation where we propose a unifying and general framework for estimating complex densities using monotone and bijective triangular transformations. The main idea in this general framework is to specify one-dimensional transformations that can be equivalently seen as specifying conditional densities and iteratively extending to higher-dimensions. Using this framework, we study popular autoregressive and flow based methods and reveal their commonalities and differences. We further present a unified understanding of the limitations and representation power of these recent approaches. The framework also allows us to provide a modular proof that can help in constructing universal flow based methods. This modular proof and triangular based general framework helps us to subsequently uncover a novel Sum-of-Squares

flow that is interpretable, universal and easy to train.

I will conclude with some discussion and future directions in Chapter 5.

Most results in this thesis have been published previously: Chapter 2 in [Jaini et al., 2018] at NeurIPS 2018, Chapter 3 in [Jaini et al., 2017] at ICLR 2017 and [Jaini and Poupart, 2017] at NeurIPS workshop on Advances in Approximate Bayesian Inference 2017, and Chapter 4 in [Jaini et al., 2019] at ICML 2019. I outline my contributions in each of these publication below:

- [Jaini et al., 2018] : The co-authors for this paper were Pascal Poupart and Yaoliang Yu. I, with excellent guidance from Yaoliang Yu, proposed the initial problem, derived the main results of the paper, implemented the SPN-CG algorithm, conducted all the experiments and wrote the paper. Pascal Poupart was the second supervising author who helped with pointing out references for connection between SPNs and Bayesian Networks.

- [Jaini and Poupart, 2017] : Pascal Poupart was the coauthor on this paper and supervised the project. I proposed the problem, implemented the solution and devised the Bayesian Moment matching algorithm for Gaussian Mixtures, conducted the experiments and wrote the paper.

- [Jaini et al., 2017] : This paper had Zhitang Chen, Pablo Carbajal, Edith Law, Laura Middleton, Kayla Regan, Mike Schaekermann, George Trimponias, James Tung, and Pascal Poupart as coauthors. The initial problem was proposed by Zhitang Chen in the context of telecommunications network. I formulated the problem, developed and implemented the complete transfer learning algorithm using Bayesian Moment Matching and performed all the experiments. I also extended the application of this algorithm to other domains to include collaborations with Pablo Carbajal, Laura Middleton, Kayla Regan, and James Tung for activity recognition and they helped with the study and collection of data. Mike Schaekermann and Edith Law helped with the study and data collection for sleep stage classification. George Trimponias helped with experiments using Recurrent Neural Nets and Pascal Poupart was the supervising author.

- [Jaini et al., 2019] : The co-authors for this paper were Kira A. Selby and Yaoliang Yu. Yaoliang and I proposed the problem as an extension to our previous work [Jaini et al., 2018]. I worked very closely with Yaoliang on this project and with his expert guidance I was able to make the connection between triangular transformations and neural density estimation methods, present the unifying framework of triangular

transformations for neural density estimation linking it to several previously proposed frameworks, propose the Sum-of-Squares Polynomial (SOS) Flows, provide a modular proof of universality for SOS flows along with other flow based methods and write the paper. I also implemented the algorithm for SOS flows and performed the experiments. The current shortened proof of three lines for universality was possible due to inputs from Csaba Szepesvári. Kira helped with setting up basic code for SOS flows and with debugging its PyTorch implementation.

Other relevant publications to the main underlying themes of the thesis that have not appeared in this thesis are: [Jaini et al., 2016] and [Poupart et al., 2016].

Chapter 2

Deep Homogeneous Mixture Models

2.1 Introduction

Many unsupervised and semi-supervised learning algorithms either implicitly (e.g. generative adversarial networks) or explicitly estimate (some functional of) the underlying density function. In this chapter, we study the problem of density estimation with an explicit representation through finite mixture models (FMMs) [McLachlan and Peel, 2004a], which have endured thorough scientific scrutiny over decades. The popularity of FMMs is largely due to their simplicity, interpretability, and universality, in the sense that, given sufficiently many components (satisfying mild conditions), FMMs can approximate any distribution to an arbitrary level of accuracy [Nguyen and McLachlan, 2016].

Many familiar unsupervised models in machine learning, at their core, provide a compact representation of homogeneous density mixtures. This list includes (but is not limited to) hidden Markov models (HMM), the recently proposed tensorial mixture models (TMM) [Sharir et al., 2018], latent tree graphical models (LTM)[Mourad et al., 2013], hierarchical tensor formats (HTF) [Hackbusch, 2012], and sum-product networks (SPN) [Darwiche, 2003, Poon and Domingos, 2011]. However, despite all being a certain form of FMM, the precise relationships among these models are not always well-understood. Our first contribution fills this gap: we prove (roughly) that $\{HMM, TMM\} \subseteq LTM \subseteq HTF \subseteq SPN$. Moreover, converting from a lower to an upper class can be achieved in linear time and without any increase in size. Our results not only clarify the similarities and subtle differences between these widely-used models, but also pave the way for a unified treatment of many properties of such models, using tools from linear algebra.

We next investigate the consequence of converting a deep mixture model into a shallow one. We first prove that the (nonnegative) tensor rank exactly characterizes the minimum size of a shallow SPN (or LTM or HTF due to equivalence) that represents a given homogeneous mixture. Then, we show that a *generic* "deep" SPN (with depth at least 2) can be *exactly* represented by a shallow SPN only when the latter contains exponentially many product nodes. Our result extends significantly those in [Cohen et al., 2016, Sharir et al., 2018, Delalleau and Bengio, 2011, Martens and Medabalimi, 2014, Cohen and Shashua, 2016] in various aspects, but most saliently from the restrictive full binary tree [Cohen et al., 2016, Sharir et al., 2018] to *any* rooted tree. As a consequence, our results imply that a generic HMM (whose underlying tree is "completely" unbalanced) cannot be exactly represented by any polynomially-sized shallow SPN, which, to our best knowledge, has not been shown before.

From a practical point of view, *exact* representations are an overkill: it suffices to *approximate* a given density mixture with reasonable accuracy. Our third contribution demonstrates that under the ℓ_∞ metric, we can approximate any homogeneous density mixture within ϵ accuracy by combining $O(1/\epsilon^2)$ shallow SPNs. However, our proof requires the knowledge of the target density hence is not practical. Instead, borrowing a classic idea from [Li and Barron, 2000] we show that minimizing the KL divergence using the conditional gradient algorithm can also approximate any homogeneous mixture within ϵ accuracy by combining $O(1/\epsilon^2)$ base SPNs, where the hidden constant decreases exponentially wrt the depth of the base SPNs. Each iteration of the conditional gradient algorithm amounts to learning a base SPN hence can be efficiently implemented. We conduct thorough experiments on both synthetic and real datasets and confirm the benefits of depth in density estimation.

2.2 Density Estimation using Mixture Models

Preliminaries

For any natural number d, we denote $[d] := \{1, \dots, d\}$. Let $\mathsf{V}_i, i \in [d]$, be k-dimensional vector spaces over the real field $\mathbb{R}$, then the tensor product $\mathsf{V}_1 \otimes \cdots \otimes \mathsf{V}_d$ is the canonical vector space that linearizes multilinear maps over the product space $\mathsf{V}_1 \times \cdots \times \mathsf{V}_d$. Perhaps the simplest way to construct the tensor product is to first formally define rank-1 tensors as:

$$\{\mathbf{v}_1 \otimes \cdots \otimes \mathbf{v}_d : \quad \mathbf{v}_i \in \mathsf{V}_i,\ i \in [d]\}, \tag{2.1}$$

and then take the linear span of rank-1 tensors. For each $\mathscr{T} \in \mathsf{V}_1 \otimes \cdots \otimes \mathsf{V}_d$, we define its rank as

$$\operatorname{rank}(\mathscr{T}) := \min\{r : \mathscr{T} = \sum_{\gamma=1}^{r} \mathbf{v}_1^\gamma \otimes \cdots \otimes \mathbf{v}_d^\gamma,\ \mathbf{v}_i^\gamma \in \mathsf{V}_i,\ i \in [d],\ \gamma \in [r]\}. \tag{2.2}$$

Sometimes we will further restrict each factor $\mathbf{v}_i^\gamma$ to some subset $\mathsf{U}_i \subseteq \mathsf{V}_i$, leading to a "larger" notion of rank. For instance, when $\mathsf{V}_i \equiv \mathbb{R}^k$, the above definition is called the CP-rank and if we take $\mathsf{U}_i = \mathbb{R}^k_+$, then we get the refined notion of nonnegative rank, denoted as rank_+. Obviously, $\operatorname{rank}_+ \geq \operatorname{rank}$ (whenever the former is defined).

Usually we can identify a d-order tensor $\mathscr{T} \in \mathsf{V}_1 \otimes \cdots \otimes \mathsf{V}_d$ with a multi-dimensional array

$$\mathcal{T} = [\mathcal{T}_{i_1,\ldots,i_d}]_{i_j \in [k_i], j \in [d]} \in \bigotimes_i \mathbb{R}^{k_i} \simeq \mathbb{R}^{k_1 \times \cdots \times k_d}, \tag{2.3}$$

once some bases have been chosen for each V_i. We can extend an inner product to the tensor product space: provided that some inner product $\langle \cdot, \cdot \rangle_i$ has been specified on each V_i, we first define the inner product for rank-1 tensors:

$$\langle \mathbf{u}_1 \otimes \cdots \otimes \mathbf{u}_d, \mathbf{v}_1 \otimes \cdots \otimes \mathbf{v}_d \rangle := \prod_{i=1}^{d} \langle \mathbf{u}_i, \mathbf{v}_i \rangle_i, \tag{2.4}$$

and then extend multi-linearly.

We give an explicit description of TMM [Sharir et al., 2018] here. For simplicity, let us assume $d = b^L$ for some integers b and L. Then, every d-order tensor $\mathscr{T}$ can be represented recursively as

$$\phi_\gamma^{\ell,t} = \sum_{j=1}^{r_{\ell-1}} w_j^{\ell,t,\gamma} \bigotimes_{s=1}^{b} \phi_j^{\ell-1,b(t-1)+s}, \quad \ell \in [L-1],\ \gamma \in [r_\ell],\ t \in [b^{L-\ell}], \tag{2.5}$$

$$\mathscr{T} = \phi_1^{L,1} = \sum_{j=1}^{r_{L-1}} w_j^{L,1,1} \bigotimes_{s=1}^{b} \phi_j^{L-1,s}, \tag{2.6}$$

where $\phi_\gamma^{0,i} \in \mathsf{V}_i$ for all $\gamma \in [r_0]$. Note that the tensor $\mathscr{T}$ is completely determined by

$$\{\phi_\gamma^{0,i} : \gamma \in [r_0], i \in [d]\} \bigcup \{\mathbf{w}^{\ell,t,\gamma} \in \mathbb{R}^{r_{\ell-1}} : \ell \in [L-1], \gamma \in [r_\ell], t \in [b^{L-\ell}]\} \cup \{\mathbf{w}^{L,1,1} \in \mathbb{R}^{r_{L-1}}\}, \tag{2.7}$$

where the former are the base vectors at the bottom level and the latter are the coefficient vectors at each intermediate level. Note that the representation (2.5)-(2.6) is not 1-1 (hence some redundancy). Let TMM_r^b (with default $\mathsf{TMM}_r := \mathsf{TMM}_r^2$) be the class of tensors that can be represented as in (2.5)-(2.6).

A simple counting argument reveals that the coefficient tensors in TMM_r^b have $\frac{d-b}{b-1}r^2 + r$ entries. It is clear that $\mathsf{TMM}_r^b \subseteq \mathsf{TMM}_{r+1}^b$, and TMM_1^b is exactly the set of rank-1 tensors. As shown in [Hackbusch, 2012], every tensor of rank r can be represented in TMM_r^b. Similarly, every tensor of nonnegative rank r can be represented in TMM_r^b, with all base vectors and coefficient vectors in (2.7) nonnegative. Moreover, we can normalize the base vectors $\phi_\gamma^{0,i}$ so that they have unit ℓ_1 norm.

Problem Set-up

We now introduce our main problem: how to estimate a multivariate density through an explicit, finite homogeneous mixture. To set up the stage, let $\mathbf{x} = (x_1, \ldots, x_d)$, with $x_i \in \mathbb{X}_i$ where each $\mathbb{X}_i$ is a Borel (measurable) subset of the Euclidean space $\mathbb{E}_i$. We equip a Borel measure μ_i on $\mathbb{X}_i$. All our subsequent measure-theoretic definitions are w.r.t. the Borel σ-field of $\mathbb{X}_i$ and the measure μ_i. Let $\mathbb{X} = \mathbb{X}_1 \times \cdots \times \mathbb{X}_d$ and $\mu = \mu_1 \times \cdots \times \mu_d$ be the product space and product measure, respectively. For each $i \in [d] := \{1, \ldots, d\}$, let $\mathcal{F}_i$ be a class of density functions (w.r.t. μ_i) of the variable x_i, and let $\mathcal{G}_i = \mathrm{conv}(\mathcal{F}_i)$ be its convex hull. The function class $\mathcal{F}_i$ is essentially our basis of densities for the variable x_i. Our setting here follows that in [Martens and Medabalimi, 2014] and includes both continuous and discrete distributions.

We are interested in constructing a finite density mixture [McLachlan and Peel, 2004a], using component densities from the basis class $\mathcal{F} = \bigcup_{i=1}^d \mathcal{F}_i$. We assume that our finite mixture f is "**homogeneous**," i.e.

$$f(\mathbf{x}) = \sum_{j_1=1}^{k_1} \sum_{j_2=1}^{k_2} \cdots \sum_{j_d=1}^{k_d} \mathcal{W}_{j_1,j_2,\ldots,j_d} \prod_{i=1}^{d} f_{j_i}^i(x_i) = \langle \mathcal{W}, \vec{f}^1(x_1) \otimes \cdots \otimes \vec{f}^d(x_d) \rangle, \tag{2.8}$$

where $\vec{f}^i := (f_1^i, \ldots, f_{k_i}^i) \in \mathcal{F}_i^{k_i}$, $\mathcal{W} \in \bigotimes_i \mathbb{R}_+^{k_i} \simeq \mathbb{R}_+^{k_1 \times \cdots \times k_d}$ is a d-order density tensor (nonnegative and sum to 1), and $\langle \cdot, \cdot \rangle$ is the standard inner product on the tensor product space. We provided basic details about tensors at the beginning of Section 2.2 that will be sufficient for our exposition and we refer the reader to the excellent book [Hackbusch, 2012] for more in-depth details. By dropping linearly dependent densities in each $\mathcal{F}_i$ we can assume w.l.o.g. the tensor representation $\mathcal{W}$ is *unique*.

There are a number of reasons for restricting to homogeneous mixtures: Firstly, this is the most common choice for estimating a multivariate density function [Tsybakov, 2009]. Secondly, we can always apply the usual "homogenization" trick, i.e., by enlarging the function class $\mathcal{F}_i$ and appending the (improper) density 1 to each $\mathcal{F}_i$. Thirdly, homogeneous densities are "universal" if each class $\mathcal{F}_i$ is, cf. Appendix A of [Sharir et al., 2018]. In other words, any joint density can be approximated arbitrarily well by a homogeneous density, provided that each marginal class $\mathcal{F}_i$ can approximate any marginal density arbitrarily well and the size (i.e. k_i) tends to ∞. See Section 2.6.1 for some empirical verifications, where we show that convex combinations of relatively few isotropic Gaussians can approximate mixtures of Gaussians of full covariance matrices surprisingly well. Lastly, as we argue below, many known models in machine learning are simply compact representations of homogeneous mixtures.

2.3 Compact Representation of Homogeneous Mixtures

We now recall a few unsupervised learning models in machine learning and show that they have a compact representation of homogeneous mixtures at their core. We prove the precise relationship amongst them. Our results clarify the similarity and difference of these recent developments, and pave the way for a unified treatment of depth separation (Section 2.4) and model approximation (Section 2.5).

Sum-Product Networks (SPN) [Darwiche, 2003, Poon and Domingos, 2011, Martens and Medabalimi, 2014] An SPN $\mathfrak{T}$ is a rooted tree whose *leaves are density functions* $f_j^i(x_i)$ over each of the variables $x_1, \ldots, x_d$ and whose internal nodes are either a sum node or a product node. Each edge (u, v) emanating from a sum node u has an associated nonnegative weight w_{uv}. The value $\mathfrak{T}_v$ at a product node v is the product of the values of its children, $\prod_{u \in ch(v)} \mathfrak{T}_u$. The value $\mathfrak{T}_u$ at a sum node u is the weighted sum of the values of its children, $\sum_{v \in ch(u)} w_{uv} \mathfrak{T}_v$. The value of an SPN $\mathfrak{T}$ is the expression evaluated at the root node, which we denote as $\mathfrak{T}(\mathbf{x})$. The *scope* of a node v in an SPN is the set of all variables that appear in the leaves of the sub-SPN rooted at v. We only consider *decomposable* and *complete* SPNs, i.e., the children of each sum node must have the same scope and the children of each product node must have disjoint scopes. The main advantage of a decomposable and complete SPN over a generic graphical model is that joint, marginal and conditional queries can be answered by two network evaluations and

hence, exact inference takes linear time with respect to the size of the network [Darwiche, 2003, Poon and Domingos, 2011, Martens and Medabalimi, 2014]. In comparison, inference in Bayesian Networks and Markov Networks may take exponential time in terms of the size of the network. W.l.o.g. we can rearrange an SPN to have alternating sum and product layers as we prove below.

Theorem 1. *Any SPN can be rearranged to have alternating layers of sum and product nodes without any change in the size of the resultant standard SPN from the original SPN.*

Proof. It is straightforward to show that consecutive combination of either sum nodes or product nodes can be merged into one layer of the corresponding nodes. This can be seen as follows: consider a sum node v that has m sum nodes as children and denote the set as $ch(v) := \{v_i\}_{i=1}^{m}$. Then, the expression f_v evaluated at v is

$$f_v(\mathbf{x}) = \sum_{i=1}^{m} \alpha_{v_i} f_{v_i}(\mathbf{x}) \tag{2.9}$$

However, since each $v_i \ \forall i \in [m]$ is also a sum node; denote the children of v_i by the set $ch(v_i) := \{\hat{v}_{i,j}\}_{j=1}^{t_i}$ for each $i \in [m]$. Thus,

$$f_{v_i}(\mathbf{x}) = \sum_{j=1}^{t_i} \beta_{\hat{v}_{i,j}} f_{\hat{v}_{i,j}} \tag{2.10}$$

Therefore, $f_v(\mathbf{x})$ can now be re-written as

$$f_v(\mathbf{x}) = \sum_{i=1}^{m} \alpha_{v_i} \sum_{j=1}^{t_i} \beta_{\hat{v}_{i,j}} f_{\hat{v}_{i,j}} \tag{2.11}$$

$$= \sum_{i=1}^{m} \sum_{j=1}^{t_i} \alpha_{v_i} \beta_{\hat{v}_{i,j}} f_{\hat{v}_{i,j}} \tag{2.12}$$

$$\tag{2.13}$$

Define a $1-1$ mapping between the tuple (i,j) $i \in [m],\ j \in [t_i]$ and $[K]$ where $K = \sum_{i=1}^{m} t_i$ such that $k = j + \sum_{l=1}^{i-1} t_l,\ k \in [K]$. Then, we can re-write the above as

$$f_v(\mathbf{x}) = \sum_{k=1}^{K} \gamma_k f_{\hat{v}_K} \tag{2.14}$$

where $\gamma_k = \alpha_{v_i}\beta_{\hat{v}_{i,j}}$ and $f_{\hat{v}_k} = f_{\hat{v}_{i,j}}$. This shows that two consecutive layers of sum nodes can be collapsed into one layer of sum nodes while preserving the same size of the network. This can similarly be shown for consecutive layers of product nodes.

Next, we give the procedure to convert any SPN into an SPN with alternating layers of sums and products. Perform a top-down pass starting at the root node (W.l.o.g. assume the root node is a sum node). For every child of the root node, if it is a sum node, merge the node into the root node. This ensures that after this step the top layer and the next layer are alternating (including leaf nodes). Proceeding similarly for every node in the network ensures the final network has alternating layers throughout. □

The latent variable semantics [Peharz et al., 2017] as well as SPNs representing a mixture model over its leaf densities [Poon and Domingos, 2011] is well-known. It is also informally known that many tractable graphical models can be treated as SPNs, but precise characterizations are scarce (see [Zhao et al., 2015] which relates SPNs with Bayesian Networks).

Self-similar SPNs ($\mathbf{S^3PN}$) We call an SPN self-similar, if at *every* sum node, the sub-tree rooted at each of its (product node) children is the same, except the weights at corresponding sum nodes and the densities (but not the variables) at corresponding leaf nodes may differ. This special class of SPNs is exactly equivalent to some recently proposed unsupervised learning models, as we show below.

Hierarchical Tensor Format (HTF) [Hackbusch, 2012] We showed in (2.8) that a homogeneous mixture can be identified with a tensor $\mathcal{W}$, whose explicit storage can, however, be quite challenging since its size is $\prod_{i=1}^{d} k_i$. HTF [Hackbusch, 2012] aims at representing tensors compactly, hence can also be used for representing homogeneous mixtures. An HTF consists of a dimension-partition rooted tree (DPT) T, d vector spaces V_i with bases[1] $\mathcal{F}_i$ at the d leaf nodes, and at most $d-1$ internal nodes which are certain *subspaces* of the tensor product of vector spaces at *disjoint* children nodes. Note that the dimension of the tensor product $\mathsf{U}\otimes\mathsf{V}$ is the product of the dimensions of U and V. The key in HTF is to truncate each tensor product with a (much smaller) subspace, hence keeping the total storage manageable. Moreover, at each internal node v with k children nodes $\{v_i\}$, instead of storing its r bases directly, we store r coefficient tensors $\{w^{v,\gamma}_{j_1,\ldots,j_k} : \gamma \in [r]\}$ such that, recursively, the γ-th basis at node v is $\sum_{j_1}\cdots\sum_{j_k} w^{v,\gamma}_{j_1,\ldots,j_k}\mathbf{v}_{j_1}\otimes\cdots\otimes\mathbf{v}_{j_k}$, where $\{\mathbf{v}_{j_i}\}$ consists of the bases at the i-th child node v_i. To our best knowledge, HTFs have not

[1] More generally frames, in particular, the elements need not be linearly independent.

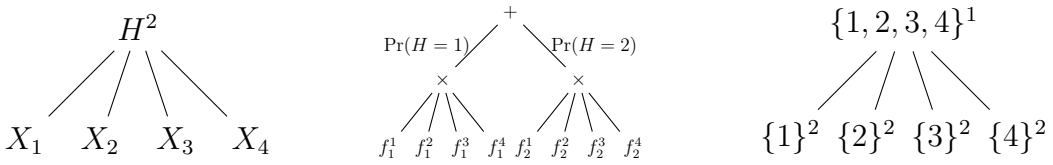


Figure 2.1: Left: A simple latent class model (special case of LTM). The superscript 2 indicates the number of values the hidden variable H can take. Middle: The equivalent S^3PN, where $f_j^i(x_i) = p(X_i = x_i|H = j)$ is from the density class $\mathcal{F}_i$. Right: The dimension-partition tree in an equivalent HTF_+. The superscript indicates the number of bases, which should be the same for sibling nodes.

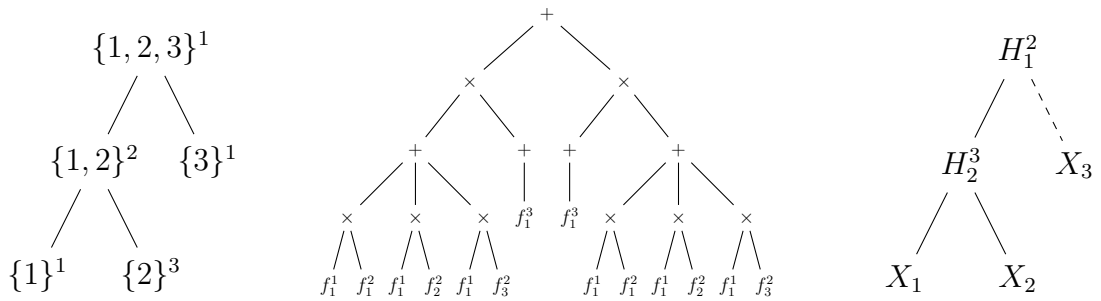


Figure 2.2: Left: A dimension-partition tree in HTF_+. The superscripts indicate the number of bases. Middle: The equivalent S^3PN. The leaf f_j^i is the j-th basis of vector space V_i. Right: An "equivalent LTM." The superscripts indicate the number of values each hidden variable can take. The two densities of X_3 are equal, i.e. $f_1^3 = f_2^3$ (hence X_3 does not actually depend on H_1).

been recognized as SPNs previously, although they have been used in a spectral method for latent variable models [Song et al., 2013].

To turn an HTF into an SPN, more precisely an S^3PN, we start from the root of the dimension-partition tree T. For each internal node v with say r bases and say k children nodes $\{v_i\}$, each of which has r_i bases themselves, we create three layers in the corresponding S^3PN: in the first layer we have r sum nodes $\{\mathsf{S}_\gamma^v\}$, each of which is (fully) connected, with respective weights $w_{j_1,\dots,j_k}^{v,\gamma}$, to the second layer of $\prod_{i=1}^k r_i$ product nodes $\{\mathsf{P}_{j_1,\dots,j_k}^v\}$, and finally the third layer consists of $\sum_{i=1}^k r_i$ sum nodes $\{\mathsf{S}_{j_i}^{v_i}\}$. The product node $\mathsf{P}_{j_1,\dots,j_k}^v$ is connected to k sum nodes $\{\mathsf{S}_{j_1}^{v_1}, \dots, \mathsf{S}_{j_k}^{v_k}\}$. Note that the weights $w_{j_1,\dots,j_k}^{v,\gamma}$ need not be positive or sum to 1 in HTF, although for representing a homogeneous mixture we can make this choice and we call this subclass HTF_+. Clearly, our construction is

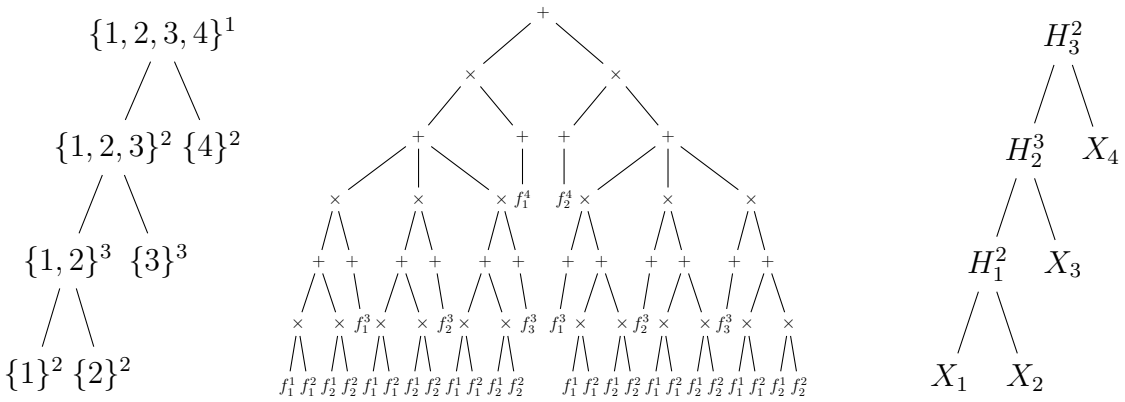


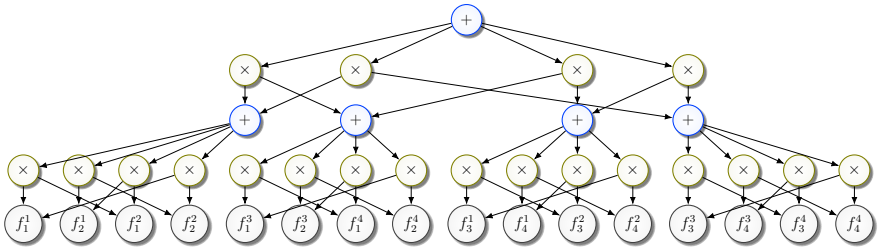
Figure 2.3: Left: A dimension-partition tree in tensor-train. The superscripts indicate the number of bases, which should remain constant for siblings. Middle: The equivalent S^{3}PN. The leaf f_j^i is the j-th basis of vector space V_i. Right: An equivalent HMM. The superscripts indicate the number of values each hidden variable can take.

reversible hence we can turn an S^{3}PN into an equivalent HTF$_+$ as well. The construction takes linear time and there is no increase of representation size. See Figs.2.1 and 2.2 for simple illustrations[2]. In summary, HTF *is exactly* S^{3}PN *with arbitrary weights*. We also note that there are other hierarchical tensor representations like tensor-train format that is already known to be equivalent to HTF [Hackbusch, 2012] hence also to S^{3}PN as we show in Figure 2.7.

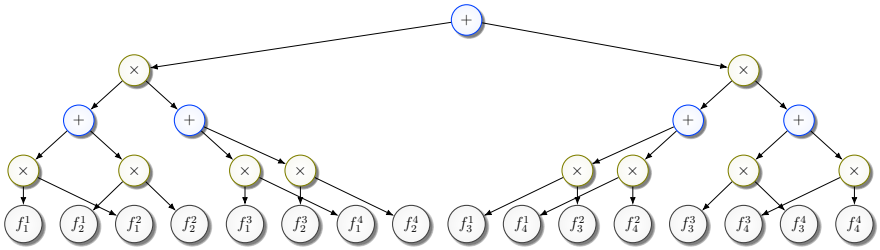
Diagonal HTF (dHTF) [Hackbusch, 2012] For later reference, let us call the subclass of HTFs whose coefficient tensors $w^{v,\gamma}_{j_1,\ldots,j_k}$ (that define bases recursively at internal nodes of the DPT, see above) are diagonal for all v and γ as dHTF, i.e., siblings in the DPT must have the same number of bases ($r_i \equiv r$) and $w^{v,\gamma}_{j_1,\ldots,j_k} \neq 0$ only when $j_1 = \ldots = j_k$. In neural network terminology, dHTFs are "locally connected." Compared to the fully connected HTF, dHTFs significantly reduce the representation size at the expense of expressiveness as shown in Figure 2.4. For instance, the $\prod_{i=1}^k r_i = r^k$ product nodes in the above conversion from HTF to S^{3}PN are reduced to merely r product nodes.

Latent Tree Models (LTM) [Mourad et al., 2013, Song et al., 2013, Choi et al., 2011] An LTM is a rooted tree graphical model with observed variables X_i on the leaves

[2]All of our illustrations of S^{3}PN in the main text are drawn with some redundant leaves, for the sake of making the self-similar property apparent. See Appendix A for the reduced (but equivalent) counterparts.



(a) HTF



(b) dHTF

Figure 2.4: Top : A general HTF representation. The network has cross connections and calculates all possible multiplications. Bottom : A dHTF with same bases functions. The dHTF representation allows for local connections.

and hidden variables H_j on the internal nodes. Note that we allow observed variables X_i to be either continuous or discrete but the hidden variables H_j can take only finitely many values. Using conditional independence, the joint density of observed variables is given as

$$f(x_1, \ldots, x_d) = \sum_{h_1} \cdots \sum_{h_t} \mathcal{W}(h_1, \ldots, h_t) \prod_{i=1}^{d} f^i_{h_{\pi_i}}(x_i), \tag{2.15}$$

where H_{π_i} is the parent of X_i. From (2.15) it is clear that an LTM is a homogeneous density mixture, whose tensor representation is given by the joint density $\mathcal{W}$ of the hidden variables. What is less known[3] is that LTMs are a special subclass of **self-similar** SPNs. It may appear that the size of S^{3}PN is larger than that of an equivalent LTM, but this is because S^{3}PN also encodes the conditional probability tables (CPT) into its structure whereas LTMs require other means to store CPTs. Note also that to evaluate an LTM, one usually needs to run a separately designed algorithm (such as message passing), while in S^{3}PN we evaluate the leaf densities and propagate in linear time to the root. In summary,

[3]As an evidence, we note that the recent survey [Mourad et al., 2013] on LTMs did not mention SPNs at all.

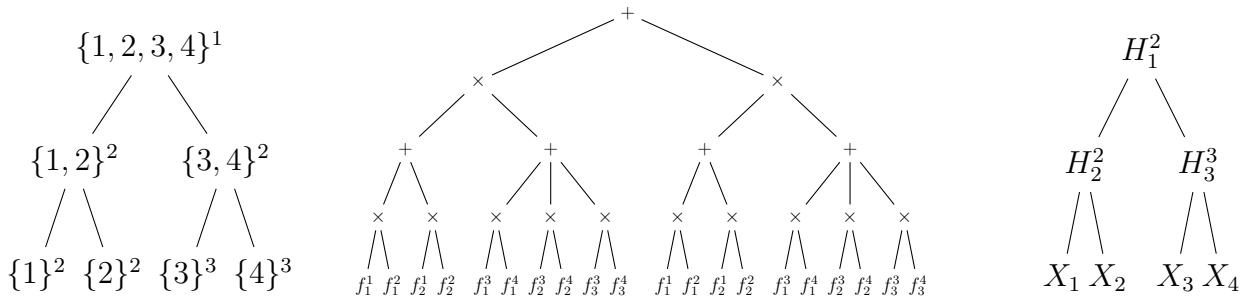


Figure 2.5: Left: A dimension-partition tree in HTF_+. The superscripts indicate the number of bases. Middle: The equivalent S^3PN. The leaf f_j^i is the j-th basis of vector space V_i. Right: An equivalent LTM. The superscripts indicate the number of values each hidden variable can take.

LTM *is a subclass of* S^3PN *with CPTs encoded as edge weights and with inference simplified as network propagation.* More precisely, LTM is exactly dHTF_+, since conditioned on the parent, all children nodes must depend on the same realization. An algorithm for converting LTMs into equivalent S^3PNs, along with more examples (Fig. 2.1 and Fig.A.1 in Appendix A.1).

Given an LTM, we can build a corresponding S^3PN as follows: starting from the root of the LTM, for each hidden variable H that takes k possible values $\{1, \ldots, k\}$ and that has r children nodes $\{V_1, \ldots, V_r\}$, we create a sum node S_H with k children product nodes $\{\mathsf{P}_{H,1}, \ldots \mathsf{P}_{H,k}\}$, each of which has r children sum nodes $\{\mathsf{S}_{V_1}, \ldots, \mathsf{S}_{V_r}\}$. We set the weight from the sum node S_H to its i-th child product node $\mathsf{P}_{H,i}$ as $\Pr(H = i|\pi(H) = j)$, if S_H connects to the j-th child product node of the parent hidden variable $\pi(H)$ (for the root, the parent is empty). If the child V_t is a hidden variable, we continue the construction similarly, while if $V_t = X_i$ is an observed variable, then we replace the sum node S_{V_t} with the density $f_j^i(x_i)$, assuming S_{V_t} is connected to the j-th child product node of the parent hidden variable H. Algorithm 4 summarizes this construction, and Figure 2.1 illustrates the idea using a simple latent class model (LCM) [Mourad et al., 2013]. In Figure A.1 we give another example to illustrate Algorithm 4.

Tensorial Mixture Models (TMM) [Sharir et al., 2018, Cohen et al., 2016, Cohen et al., 2017] TMM [Sharir et al., 2018] is a recently proposed subclass of dHTF_+ where nodes on the same *level* of the dimension-partition tree must have the same number of bases. Clearly, TMM is a strict subclass of LTM since the latter only requires sibling nodes in the DPT to have the same number of bases. We note that TMM, as defined in

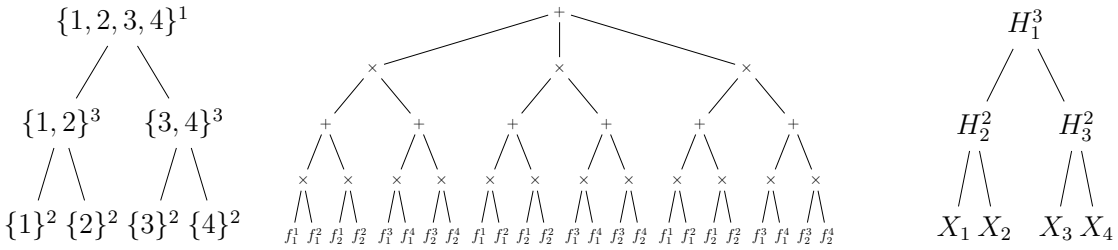


Figure 2.6: Left: A dimension-partition tree in HTF. The superscripts indicate the number of bases, which should remain constant on each level. Middle: The equivalent S^3PN. The leaf f_j^i is the j-th basis of vector space V_i. Right: An equivalent TMM. The superscripts indicate the number of values each hidden variable can take (again, remaining constant on each level).

[Sharir et al., 2018], also assumes the DPT to be binary and balanced, i.e. each internal node has exactly two children, although this condition can be easily relaxed. See Figure 2.6 and its reduced form in Appendix A.2 for a simple example. Further, in Figure 2.5, we give an example of an LTM that is not a TMM.

Hidden Markov Models (HMM) [Baum and Petrie, 1966, Rabiner, 1989] HMM is a strict subclass of LTM. [Ishteva, 2015] recently observed that HMM is equivalent to the tensor-train format, a special subclass of dHTF_+ where the DPT is binary and completely "imbalanced." See Figure 2.7 for a simple example. In some sense, TMM and HMM are the two opposite extremes within dHTF_+ (or equivalently LTM).

Further, in Figure 2.8, we give an example of an SPN that is not an S^3PN, leading to the following summary:

Theorem 2. *$\{TMM, HMM\} \subseteq LTM = dHTF_+ \subseteq HTF_+ = S^3PN \subseteq SPN$, in the sense that we can convert in linear time from a lower representation class to an upper one, without any increase in size.*

It is important to point out one subtlety here: any (complete and decomposable) SPN, if expanded at the root, is a homogeneous mixture (cf. (2.8)). Hence, any SPN is even equivalent to an LCM (i.e. an LTM with one hidden variable taking many values, like in Figure 2.1), at the expense of potentially increasing the size (significantly). Thus, the containment in Theorem 2 should be understood under the premise of not increasing the representation size. It would be interesting to understand if the containment is strict if only polynomial increase in size is allowed. We provide more comparing examples in Appendix A

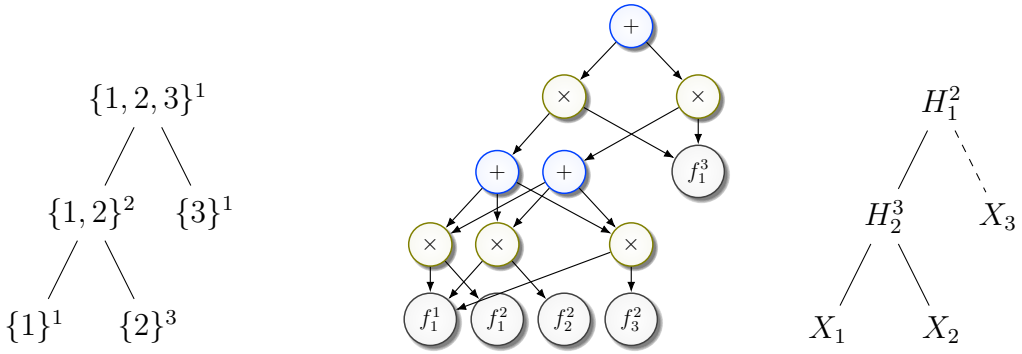


Figure 2.7: Left: A dimension-partition tree in HTF_+. The superscripts indicate the number of bases. Middle: The equivalent $\mathrm{S^3PN}$. The leaf f_j^i is the j-th basis of vector space V_i. Right: An "equivalent LTM." The superscripts indicate the number of values each hidden variable can take. The two densities of X_3 are equal, i.e. $f_1^3 = f_2^3$ (hence X_3 does not actually depend on H_1).

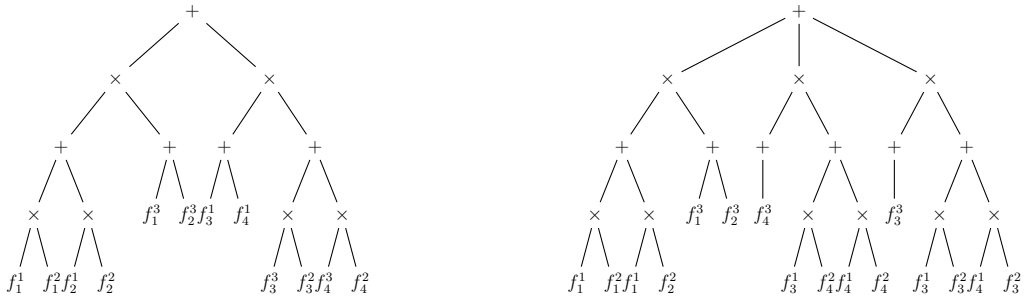


Figure 2.8: Left: An SPN but which is not an $\mathrm{S^3PN}$. The leaf f_j^i is the j-th basis of vector space V_i. Right: The equivalent $\mathrm{S^3PN}$ requires an increase in the size of the network.

for different models, and in the next section we discuss the (huge) size consequence from converting a certain upper representation class to some lower one.

2.4 Depth Separation

In the previous section, we established relationships among different representation schemes for homogeneous density mixtures. In this section, we prove an exponential separation in size when converting one representation to another and extend the results in [Delalleau and Bengio, 2011, Martens and Medabalimi, 2014, Cohen et al., 2016, Sharir et al., 2018].

The key is to exploit the equivalence to HTF, which allows us to bound the model size using linear algebra.

We call a (complete and decomposable) SPN shallow if it has only one sum node, followed by a layer of product nodes. Using the equivalence in Section 2.3, we know a shallow SPN (trivially self-similar) is equivalent to an LCM (a latent tree model with one hidden node taking as many values as the number of product nodes), or an HTF_+ whose DPT has depth 1 (cf. Figure 2.1). Recall that $\mathrm{rank}_+(\mathcal{W})$ denotes the nonnegative rank of a tensor and $\mathsf{nnz}(\mathcal{W})$ is the number of nonzeros. The leaf nodes in SPN (LTM) or the leaf bases in HTF are either from $\mathcal{F}$ (union of *linearly independent* component densities) or $\mathcal{G}$ (the convex hull), see the definitions in Section 2.2.

Our first result characterizes the model capacity of shallow SPNs (LCMs):

Theorem 3. *If a shallow SPN $\mathfrak{T}$, with leaf (input) nodes from $\mathcal{G}$, represents the density mixture $\mathcal{W}$, then $\mathfrak{T}$ has at least $\mathrm{rank}_+(\mathcal{W})$ many product nodes. Conversely, there always exists a shallow SPN that represents $\mathcal{W}$ using $\mathrm{rank}_+(\mathcal{W})$ product nodes and 1 sum node.*

Proof. Suppose the shallow SPN $\mathfrak{T}$ represents the (homogeneous) mixture density $\mathcal{W}$. If the hidden layer is all sum nodes, then the output node must be a product node. The claim trivially holds in this case. If the hidden layer is r product nodes, then the output node is a sum node, with weight z_γ to the γ-th product node. The output of the SPN $\mathfrak{T}$, when expanded at the root, is in the following form:

$$\mathfrak{T}(\mathbf{x}) = \sum_{\gamma=1}^{r} z_\gamma \prod_{i=1}^{d} g_i^\gamma(x_i) = \sum_{\gamma=1}^{r} z_\gamma \langle \mathbf{w}_1^{(\gamma)} \otimes \cdots \otimes \mathbf{w}_d^{(\gamma)}, \vec{f}_1(x_1) \otimes \cdots \otimes \vec{f}_d(x_d) \rangle \tag{2.16}$$

$$= \langle \sum_{\gamma=1}^{r} z_\gamma \mathbf{w}_1^{(\gamma)} \otimes \cdots \otimes \mathbf{w}_d^{(\gamma)}, \vec{f}_1(x_1) \otimes \cdots \otimes \vec{f}_d(x_d) \rangle \tag{2.17}$$

$$= \langle \mathcal{W}, \vec{f}_1(x_1) \otimes \cdots \otimes \vec{f}_d(x_d) \rangle. \tag{2.18}$$

Thus, $\mathcal{W} = \sum_{\gamma=1}^{r} z_\gamma \cdot \mathbf{w}_1^{(\gamma)} \otimes \cdots \otimes \mathbf{w}_d^{(\gamma)}$, i.e., $\mathrm{rank}_+(\mathcal{W}) \le r$.

Conversely, let $r = \mathrm{rank}_+(\mathcal{W})$ with the decomposition $\mathcal{W} = \sum_{\gamma=1}^{r} \mathbf{w}_1^{(\gamma)} \otimes \cdots \otimes \mathbf{w}_d^{(\gamma)}$. Note that each $\mathbf{w}_i^{(\gamma)}$ is nonzero, as otherwise we would be able to reduce the rank. We construct a shallow SPN $\mathfrak{T}$ to represent $\mathcal{W}$ as follows: On the first layer we have r product nodes, with the γ-th one computing $\prod_{i=1}^{d} g_i^{(\gamma)}(x_i)$, where

$$g_i^{(\gamma)}(x_i) = \sum_{j=1}^{k} \bar{w}_{ij}^{(\gamma)} f_{i,j}(x_i), \quad \bar{w}_{ij}^{(\gamma)} = \frac{w_{ij}^{(\gamma)}}{\|\mathbf{w}_i^{(\gamma)}\|_1}. \tag{2.19}$$

Note that $\|\mathbf{w}_i^{(\gamma)}\|_1 := \sum_{j=1}^k w_{ij}^{(\gamma)} > 0$ hence the above is well-defined. Then, we add a sum node on top of all product nodes, with weight $\|\mathbf{w}^{(\gamma)}\|_1 := \prod_{i=1}^d \|\mathbf{w}_i^{(\gamma)}\|_1 > 0$ for the γ-th product node. The output of the constructed shallow SPN is:

$$f(\mathbf{x}) = \sum_{\gamma=1}^r \|\mathbf{w}^{(\gamma)}\|_1 \prod_{i=1}^d g_i^{(\gamma)}(x_i) = \sum_{\gamma=1}^r \prod_{i=1}^d \sum_{j=1}^k w_{ij}^{(\gamma)} f_{i,j}(x_i) \tag{2.20}$$

$$= \sum_{\gamma=1}^r \langle \mathbf{w}_1^{(\gamma)} \otimes \cdots \otimes \mathbf{w}_d^{(\gamma)}, \vec{f}_1(x_1) \otimes \cdots \otimes \vec{f}_d(x_d) \rangle = \langle \mathcal{W}, \vec{f}_1(x_1) \otimes \cdots \otimes \vec{f}_d(x_d) \rangle. \tag{2.21}$$

The proof is now complete. □

In other words, the nonnegative rank characterizes the smallest size of shallow SPNs (LCMs) that represent the density mixture $\mathcal{W}$. Similarly, we can prove the following result when the leaf nodes are from $\mathcal{F}$ instead of the convex hull $\mathcal{G}$.

Theorem 4. *If a shallow SPN $\mathfrak{T}$, with leaf nodes from $\mathcal{F}$, represents the density mixture $\mathcal{W}$, then either $\mathfrak{T}$ has at least $\mathsf{nnz}(\mathcal{W})$ product nodes or $\operatorname{rank}_+(\mathcal{W}) = 1$. Conversely, there always exists a shallow SPN that represents $\mathcal{W}$ using $\mathsf{nnz}(\mathcal{W})$ product nodes and 1 sum node.*

Proof. Suppose $\mathfrak{T}$ has a hidden layer of sum nodes. Because $\mathfrak{T}$ is standard, the product output is then a mixture density of the following form:

$$\mathfrak{T}(\mathbf{x}) = \prod_{i=1}^d \sum_{j=1}^k w_{ij} f_{i,j}(x_i) = \langle \mathbf{w}_1 \otimes \cdots \otimes \mathbf{w}_d, \vec{f}_1 \otimes \cdots \otimes \vec{f}_d \rangle = \langle \mathcal{W}, \vec{f}_1 \otimes \cdots \otimes \vec{f}_d \rangle. \tag{2.22}$$

Thus, $\mathcal{W} = \mathbf{w}_1 \otimes \cdots \otimes \mathbf{w}_d$ has nonnegative rank 1. On the other hand, if $\mathfrak{T}$ has a hidden layer of product nodes, then the output of the standard SPN $\mathfrak{T}$, when expanded at the root sum node, is in the following form:

$$\mathfrak{T}(\mathbf{x}) = \sum_{j_1=1}^k \cdots \sum_{j_d=1}^k z_{j_1,\ldots,j_d} \prod_{i=1}^d f_{i,j_i}(x_i) = \langle \mathcal{W}, \vec{f}_1(x_1) \otimes \cdots \otimes \vec{f}_d(x_d) \rangle. \tag{2.23}$$

Thus, $\mathcal{W} = \mathcal{Z}$, in particular $\mathsf{nnz}(\mathcal{W}) = \mathsf{nnz}(\mathcal{Z})$, but the latter is exactly the number of product nodes in $\mathfrak{T}$.

The converse part follows by reversing the above argument. □

Note that we always have $\text{rank}(\mathcal{W}) \leq \text{rank}_+(\mathcal{W}) \leq \mathsf{nnz}(\mathcal{W})$, thus the lower bound in Theorem 4 is stronger than that in Theorem 3. This is not surprising, because an SPN with leaf nodes from $\mathcal{G}$ is the same as an SPN with leaf nodes from $\mathcal{F}$ and with an additional layer of sum nodes appended at the bottom (to perform the convex hull operation). This difference already indicates that an additional layer of sum nodes at the bottom can strictly increase the expressive power of SPNs. This distinction between leaf nodes from $\mathcal{F}$ or from $\mathcal{G}$, to our best knowledge, has not been noted before.

The significance of Theorem 3 and Theorem 4 is that they give *exact* characterizations of the model size of shallow SPNs, and they pave the way for comparing more interesting models. For convenience, we state our next result in terms of LTMs, but the consequence for dHTFs or SPNs should be clear, thanks to the equivalence in Theorem 2.

Theorem 5. *Let an LTM T have d observed variables $\mathcal{X} = \{X_1, \ldots, X_d\}$ with parents H_i taking r_i values respectively. Assuming the CPTs of T are sampled from a continuous distribution, then almost surely, the tensor representation $\mathcal{W}$ for T has rank at least*

$$\max_{1 \leq m \leq d/2} \quad \max_{\{S_1, \ldots, S_m, \bar{S}_1, \ldots, \bar{S}_m\} \subseteq \mathcal{X}} \quad \prod_{i=1}^{m} \min\{r_i, \bar{r}_i, k_i, \bar{k}_i\}, \tag{2.24}$$

where k_i ($\bar{k}_i$) is the number of (linearly independent) component densities that S_i ($\bar{S}_i$) has, and S_i ($\bar{S}_i$) are non-siblings.

Proof. We present our proof using the equivalence between LTM and dHTF.

Recall that an HTF consists of a dimension-partition tree (DPT) T whose leaf nodes $\{i\}, i \in [d]$ represent d vector spaces with bases $\{\mathbf{v}_1^i, \ldots, \mathbf{v}_{r_i}^i\}$ respectively. At each internal node β with b_β children nodes $\beta_1, \ldots, \beta_{b_\beta}$, we have r_β coefficient tensors $\mathbf{w}^{\beta, \ell[\beta]} \in \mathbb{R}^{r_{\beta_1} \times \cdots \times r_{\beta_{b_\beta}}}, \ell[\beta] \in [r_\beta]$, and r_α denotes the number of bases at node α. Any tensor $\mathcal{W}$ living in the space at the root D of T can thus be represented using r_D coefficients $\{c_{\ell[D]} : \ell[D] \in [r_D]\}$ in the following way (cf. Eq (11.26) of [Hackbusch, 2012] for the special case when the DPT is binary):

$$\mathcal{W} = \sum_{\substack{\ell[i]=1 \\ i \in [d]}}^{r_i} \left[\sum_{\substack{\ell[\alpha]=1 \\ \alpha \in \mathsf{T} \setminus \mathsf{L}}}^{r_\alpha} c_{\ell[D]} \prod_{\beta \in \mathsf{T} \setminus \mathsf{L}} w^{\beta, \ell[\beta]}_{\ell[\beta_1], \ldots, \ell[\beta_{b_\beta}]} \right] \bigotimes_{i=1}^{d} \mathbf{v}^i_{\ell[i]}. \tag{2.25}$$

For a dHTF, the coefficient tensors $w^{\beta, \ell[\beta]}_{\ell[\beta_1], \ldots, \ell[\beta_{b_\beta}]}$ are diagonal, so in the summation above we can only consider sibling nodes once as a group. The key observation is that the right-hand side of (2.25) is a sum of many rank-1 tensors, hence $\mathcal{W}$ is likely to have a large rank.

Let $\{S_i, \bar{S}_i : i = 1, \ldots, m\} \subseteq \mathcal{X} := \{X_1, \ldots, X_d\}$, where S_i's are non-siblings and $\bar{S}_i$'s are also non-siblings. Set $t_i = \min\{r_i \bar{r}_i, k_i, \bar{k}_i\}$. For each S_i, set its parent say H_i's (diagonal) coefficient tensor as follows:

$$w_{\ell[S_i]}^{H_i, \ell[H_i]} = \begin{cases} 1, & \text{if } \ell[H_i] = \ell[S_i] = \ell[\bar{S}_i] \leq t_i \\ 0, & \text{otherwise} \end{cases}. \tag{2.26}$$

Similarly for each $\bar{S}_i$. For any remaining internal node β, set its (diagonal) coefficient tensor as:

$$w_{\ell[\beta_1]}^{\beta, \ell[\beta]} = \begin{cases} 1, & \text{if } \ell[\beta] = 1 \\ 0, & \text{otherwise} \end{cases}. \tag{2.27}$$

Under the above specification, we have

$$\mathcal{W} \propto \sum_{j_1=1}^{t_1} \cdots \sum_{j_m=1}^{t_m} \underbrace{\left[\otimes_{i=1}^m \mathbf{v}_{j_i}^{S_i}\right]}_{\mathbf{a}_{j_1,\ldots,j_m}} \bigotimes \underbrace{\left[\otimes_{i=1}^m \mathbf{v}_{j_i}^{\bar{S}_i}\right]}_{\mathbf{b}_{j_1,\ldots,j_m}}. \tag{2.28}$$

Since $\{\mathbf{a}_{j_1,\ldots,j_m}\}$ and $\{\mathbf{b}_{j_1,\ldots,j_m}\}$ are linearly independent, respectively, through matricization we know $\text{rank}(\mathcal{W}) \geq \prod_i t_i$. This shows that there exist coefficient tensors w so that $\text{rank}(\mathcal{W}) \geq \prod_i t_i$.

To extend our conclusion from a special realization above to the generic case, let us note that the determinant of any submatrix of any matricization of $\mathcal{W}$ is a polynomial function of the coefficient tensors w. We have shown above this polynomial is not always zero, but then it follows immediately that the zero set of this polynomial has measure zero [Caron and Traynor, 2005], i.e., for a generic realization of the coefficient tensors, we have $\text{rank}(\mathcal{W}) \geq \prod_i t_i$. □

We constructed an explicit homogeneous mixture in the above proof whose rank is exponential. A similar construction, in the discrete setting, is given below [Martens and Medabalimi, 2014]:

Example 1. *Let $x_i \in \{0, 1\} \forall i$, $k = 2$ and $d = 2m$. Choose for all $\forall i$ the basis (unnormalized) densities $f_{i,1}(x_i) = 1(x_i = 1)$ and $f_{i,2}(x_i) = 1(x_i = 0)$ (wrt some non-degenerate counting measure on $\{0, 1\}$). Consider the (unnormalized) multivariate density F on $\{0, 1\}^d$ (wrt the product counting measure):*

$$F(\mathbf{x}) = \begin{cases} 1, & \textit{if } x_i = x_{i+m} \textit{ for all } 1 \leq i \leq m \\ 0, & \textit{otherwise} \end{cases}. \tag{2.29}$$

Clearly, F is a density mixture with input nodes from $\mathcal{F}$ and the associated tensor $\mathcal{W}$ satisfies $\operatorname{rank}_+(\mathcal{W}) > 1$. Hence, a standard shallow SPN needs at least $\mathsf{nnz}(\mathcal{W}) = 2^{d/2}$ product nodes to represent F, with input nodes from $\mathcal{F}$. The density mixture F is the so-called EQUAL *function in [Martens and Medabalimi, 2014], whose Theorem 24 follows immediately from our Theorem 4 since $\mathsf{nnz}(\mathcal{W}) \geq \operatorname{rank}(W)$ for any matricization W of $\mathcal{W}$.*

We note that F is also a density mixture with input nodes from $\mathcal{G}$ (elements of which are themselves mixtures of $f_{i,1}$ and $f_{i,2}$), and $\operatorname{rank}_+(\mathcal{W}) \geq \operatorname{rank}(W) = \mathsf{nnz}(\mathcal{W}) = 2^{d/2} \geq \operatorname{rank}_+(\mathcal{W})$. Thus, any standard shallow SPN with input nodes from $\mathcal{G}$ still requires $2^{d/2}$ product nodes to represent the EQUAL *function. In other words, an SPN with an input layer from $\mathcal{F}$, a layer of sum nodes, a layer of product nodes, and a single sum node as output, would still require $2^{d/2}$ product nodes in order to represent the* EQUAL *function. This distinction between input nodes from $\mathcal{F}$ and input nodes from $\mathcal{G}$, to our best knowledge, has not been noted before.*

Corollary 6. *In addition to the setting in Theorem 5, if each observed variable X_i has b sibling observed variables and $r_i \equiv r \leq k \equiv k_i$, then the tensor representation $\mathcal{W}$ has rank at least $r^{\lfloor d/b \rfloor}$.*

Corollary 7. *In addition to the setting in Theorem 5, if each observed variable X_i has no sibling observed variables and $r_i \equiv r \leq k \equiv k_i$, then the tensor representation $\mathcal{W}$ has rank at least $r^{\lfloor d/2 \rfloor}$.*

Combining Corollary 6 with Theorem 4 we conclude that an LTM T with d observed variables X_i where every b of them share the same hidden parent node is equivalent to an LCM T' where the hidden node must take at least $r^{\lfloor d/b \rfloor}$ many values. Note that T has $\Theta(d/b)$ hidden variables, each of them taking r values, thus the total size of the CPTs of T is $\Theta(rd/b)$ while the total size of that of T' is $r^{\lfloor d/b \rfloor}$, an exponential blow-up. By combining Corollary 7 with Theorem 4 a similar conclusion can be made for converting an HMM into a LCM. Of course, interpretation using SPNs is also readily available: Almost all depth-L S^3PNs ($L \geq 2$) with weights sampled from a continuous distribution can be written as a shallow SPN with necessarily exponentially many product nodes.

To our best knowledge, [Delalleau and Bengio, 2011] was the first to construct a polynomial that, while representable by a polynomially-sized depth-$\log d$ SPN, would require exponentially many product nodes if represented by a shallow SPN. However, the deep SPN given in [Delalleau and Bengio, 2011, Figure 1] is not complete. Recently, [Cohen et al., 2016] proved that the existence result of [Delalleau and Bengio, 2011] is in fact *generic*. However, the results of [Cohen et al., 2016] and subsequent work [Sharir et al., 2018] are limited to full binary trees. In contrast, our general Theorem 5 holds for any

tree, and we allow non-sibling nodes to take different number of values. As a result, we are able to handle HMMs, the opposite extreme of TMM. Another important point we want to emphasize is that the exponential separation from a shallow (i.e. depth-1) tree can be achieved by increasing the depth by merely 1, as opposed to the depth-$\log d$ constructions in [Delalleau and Bengio, 2011, Sharir et al., 2018].

We end this section by making another observation about Theorem 5: It also allows us to compare the model size of LTMs T_1 and T_2 where say T_1, after removing its root R, is a subtree of T_2. Indeed, in this case we need only define the children nodes of R as "observed" variables. Then, T_1 becomes an LCM and T_2 serves as T in Theorem 5, with observed variables as the children nodes of R. This essentially extends [Cohen et al., 2016, Theorem 3] from a full binary tree to any tree and allowing non-sibling nodes to take different number of values.

2.5 Approximate Representation

In the previous section, we proved that homogeneous mixtures representable by "deep" architectures (such as SPN or LTM) of polynomial size cannot be *exactly* represented by a shallow one with sub-exponential size. In this section, we address a more intricate and relevant question: What if we are only interested in an *approximate* representation?

To formulate the problem, let g and h be two homogeneous mixtures with tensor representation $\mathcal{W}$ and $\mathcal{Z}$, respectively. We consider the distance $\operatorname{dist}(g, h) := \|\mathcal{W} - \mathcal{Z}\|$ for some norm $\|\cdot\|$ specified later. Using the characterization in Theorem 3 we formulate our approximation problem as follows. Let Δ be a perturbation tensor with $\|\Delta\| \le \epsilon$. What is the minimum value for $\operatorname{rank}_+(\mathcal{W} + \Delta)$, i.e. the size of a shallow SPN? This motivates the following definition adapted from [Alon, 2009]:

$$\epsilon\text{-rank}_+(\mathcal{W}) = \min\Big\{\operatorname{rank}_+(\mathcal{W}+\Delta) : \|\Delta\| \le \epsilon\Big\} = \min\Big\{\operatorname{rank}_+(\mathcal{Z}) : \|\mathcal{Z}-\mathcal{W}\| \le \epsilon\Big\}. \tag{2.30}$$

In other words, ϵ-rank$_+$ is precisely the minimum size of a shallow SPN (LCM) that approximates a specified mixture $\mathcal{W}$ with accuracy ϵ. We can similarly define ϵ-rank, where we replace the nonnegative rank with the usual rank in (2.30). Note that the notion of ϵ-rank depends on the norm $\|\cdot\|$.

ℓ_1-norm First, we fix the norm $\|\cdot\|$ to be the usual ℓ_1 norm in definition (2.30), and we use the notation ϵ-rank$_+^1$ or ϵ-rank^1 to signify this convention. Our next result provides a new lower bound on the ϵ-rank, based on matricization.

Theorem 8. *Fix $\epsilon > 0$ and let $\mathcal{W} \in \mathbb{R}^{k_1 \times \cdots \times k_d}$. Then,*

$$\epsilon\text{-rank}^1_+(\mathcal{W}) \geq \epsilon\text{-rank}^1(\mathcal{W}) \geq \min\{i \geq 0 : \epsilon \geq \|W\|_{\text{tr}} - \sum_{j=1}^{i} \sigma_i(W)\}, \tag{2.31}$$

where W is any matricization of the tensor $\mathcal{W}$, $\|\cdot\|_{\text{tr}}$ is the matrix trace norm (i.e. sum of singular values), and $\sigma_i(W)$ denotes the i-th largest singular value of W.

Proof. Since the nonnegative rank is lower bounded by the rank, which is in turn lower bounded by the rank of any matricization, we clearly have

$$\epsilon\text{-rank}^1_+(\mathcal{W}) \geq \epsilon\text{-rank}^1(\mathcal{W}) \geq \epsilon\text{-rank}^1(W), \tag{2.32}$$

where W is an arbitrary matricization of $\mathcal{W}$ (note that matricization does not change the ℓ_1 norm). Moreover, for matrices, $\|\cdot\|_\infty \leq \|\cdot\|_{\text{sp}}$ (i.e., maximum singular value) hence $\|\cdot\|_1 \geq \|\cdot\|_{\text{tr}}$, thanks to duality. Therefore,

$$\epsilon\text{-rank}^1(W) = \min_{\|\Delta\|_1 \leq \epsilon} \text{rank}(W + \Delta) \geq \min_{\|\Delta\|_{\text{tr}} \leq \epsilon} \text{rank}(W + \Delta) = \min_{\|W - Z\|_{\text{tr}} \leq \epsilon} \text{rank}(Z). \tag{2.33}$$

Using say [Yu and Schuurmans, 2012, Theorem 1], we know that at the minimum we can choose Z to have the same singular vectors as W. It is clear then that Z should match the singular values of W, from the biggest to smallest, until the trace norm difference between Z and W falls under ϵ. □

The only prior work to Theorem 8 that we are aware of is [Martens and Medabalimi, 2014, Lemma 28], which only deals with the identity matrix $W = I$ and is loose by a factor of 2. [Martens and Medabalimi, 2014] went on to construct a density function based on the EQUAL function (where $W = I$, cf. Example 1) that cannot be approximated by a polynomially-sized standard shallow SPN, up to some exponentially small ϵ. This existence result is then strengthened by [Cohen et al., 2016], who showed that under the HR model, for almost every random tensor $\mathcal{W}$, there exists some ϵ (potentially depending on $\mathcal{W}$), such that any polynomially-sized standard shallow SPN cannot approximate $\mathcal{W}$ under ϵ. Based on Theorem 8, we now present a complementary result.

Theorem 9. *Fix any $\epsilon > 0$, and sample each entry of the tensor $\mathcal{W} \in \mathbb{R}^{k_1 \times \cdots \times k_d}$ independently and identically from a standard Gaussian distribution, then with high probability,*

$$\epsilon\text{-rank}^1(\mathcal{W}) \geq O(k^{d/2} - \epsilon k^{d/4}). \tag{2.34}$$

Proof. Consider the reshaped matrix $W \in \mathbb{R}^{k^{d/2} \times k^{d/2}}$ of the tensor $\mathcal{W}$. Clearly, each entry of W is again an iid sample from the standard Gaussian distribution. As shown in [Rudelson and Vershynin, 2008], the smallest singular value of W is $\Theta(k^{-d/4})$, with high probability. Let $r = \epsilon\text{-rank}^1(\mathcal{W})$, then according to Theorem 8, we have

$$\epsilon \geq \sum_{j=r+1}^{k^{d/2}} \sigma_i(W) \geq (k^{d/2} - r)k^{-d/4}. \tag{2.35}$$

Rearranging we obtain the claimed lower bound. □

The failure probability in Theorem 9 depends on d only mildly: up to a small constant it approaches 0 at the rate $c^{k^{d/2}}$ for some constant $0 < c < 1$. Moreover, the standard Gaussian distribution can be replaced with any subgaussian distribution, or more generally any distribution with a bounded 4-th moment. To see the significance of Theorem 9, let us note first that we can fix ϵ beforehand so there is no dependence on the tensor $\mathcal{W}$. Secondly, Theorem 9 implies that with high probability, for any mixture density f, even if we contend with a constant approximation accuracy $\epsilon = O(1)$, a standard shallow SPN $\mathfrak{T}$ would still need $O(k^{d/2})$ many product nodes.

ℓ_∞-norm Let the norm in the definition (2.30) be the usual ℓ_∞ norm, and we signify this choice with the notation $\epsilon\text{-rank}^\infty$. In this setting, we can prove the following nearly-tight bound on the ϵ-rank.

Theorem 10. *Fix $\epsilon > 0$ and tensor $\mathcal{W} \in \mathbb{R}^{k_1 \times \cdots \times k_d}$. Then, for some (small) constant $c > 0$,*

$$\epsilon\text{-rank}^\infty(\mathcal{W}) \leq \tfrac{c\|\mathcal{W}\|_{\text{tr}}}{\epsilon^2}, \tag{2.36}$$

where $\|\mathcal{W}\|_{\text{tr}}$ is the tensor trace norm. A similar result holds for $\epsilon\text{-rank}_+^\infty(\mathcal{W})$. The dependence on ϵ is tight up to a log factor.

Proof. Note that the ℓ_∞ norm is dominated by the ℓ_2 norm, so $\epsilon\text{-rank}^2 \geq \epsilon\text{-rank}^\infty$. Thus, given $\mathcal{W}$, we consider the approximation problem:

$$\min_{\|\mathcal{Z}\|_{\text{tr}} \leq \|\mathcal{W}\|_{\text{tr}}} \|\mathcal{Z} - \mathcal{W}\|_2^2. \tag{2.37}$$

Obviously, the minimum is 0. Moreover, if we run the generalized conditional gradient of [Frank and Wolfe, 1956] with initialization $\mathcal{Z}_0 = \mathbf{0}$, then after t iterations, we have

$$\|\mathcal{Z}_t - \mathcal{W}\|_2^2 \leq \frac{c\|\mathcal{W}\|_{\text{tr}}}{t}, \quad \text{rank}(\mathcal{Z}_t) \leq t, \tag{2.38}$$

where c is some small universal constant. Here we are exploiting the property that each iteration of the conditional gradient algorithm only increments the rank by at most 1. Setting $c\|\mathcal{W}\|_{\text{tr}}/t = \epsilon^2$ gives us

$$\|\mathcal{Z}_t - \mathcal{W}\|_\infty \leq \|\mathcal{Z}_t - \mathcal{W}\|_2 \leq \epsilon, \;\; \text{rank}(\mathcal{Z}_t) \leq \frac{c\|\mathcal{W}\|_{\text{tr}}}{\epsilon^2}, \tag{2.39}$$

whence follows $\epsilon\text{-rank}^\infty(\mathcal{W}) \leq O(\|\mathcal{W}\|_{\text{tr}}/\epsilon^2)$.

The proof for the nonnegative rank is completely similar. □

The inverse-square dependence on ϵ in Theorem 10 is almost tight, as shown below:

Theorem 11 ([Alon, 2009, Theorem 2.1])**.** *Let W be a $k^{d/2} \times k^{d/2}$ matrix with $|w_{i,i}| = 1 \; \forall i$ and $|w_{i,j}| \leq \epsilon \; \forall i \neq j$, where $k^{-d/4} \leq \epsilon \leq \frac{1}{2}$. Then, for some absolute positive constant c,*

$$\text{rank}(W) \geq \frac{cd \log k}{\epsilon^2 \log(\frac{1}{\epsilon})} \tag{2.40}$$

The above theorem, through un-matricization, clearly implies that there exist tensors $\mathcal{W}$ with $\epsilon\text{-rank}^\infty(\mathcal{W})$ lower bounded by $\frac{cd \log k}{\epsilon^2 \log(\frac{1}{\epsilon})}$, when ϵ is not too small.

A few remarks with regard to Theorem 10 are in order. We note first that our proof actually gives the same upper bound for the epsilon-rank under any ℓ_p norm where $p \in [2, \infty]$. Using the norm inequality $\|\mathcal{W}\|_1 \leq k^{d/2}\|\mathcal{W}\|_2$, we then immediately have from Theorem 10 that

$$\epsilon\text{-rank}^1(\mathcal{W}) \leq \frac{c\|\mathcal{W}\|_{\text{tr}} k^d}{\epsilon^2}. \tag{2.41}$$

Note however that there is still a factor of $k^{d/4}$ gap between the upper and lower bounds in Theorem 9. It might be possible to optimize the lower bound in Theorem 9 through different matricizations.

There are at least two issues with Theorem 10. First, if we use it to naively bound the L_1 norm difference of the underlying densities, i.e.,

$$\|g - h\|_1 = \int |\langle \mathcal{W} - \mathcal{Z}, \vec{f}_1(x_1) \otimes \cdots \otimes \vec{f}_d(x_d) \rangle| \mathrm{d}\mu(\mathbf{x}) \leq k^d \|\mathcal{W} - \mathcal{Z}\|_\infty, \tag{2.42}$$

the big constant k^d would pop out in the worse case. More disturbingly, in a practical application, we usually do not have access to $\mathcal{W}$ (which is too large to maintain directly anyways), hence it is not clear how to attain the bound in Theorem 10 algorithmically.

Note that the representative tensor $\mathcal{W}$ for a homogeneous density mixture f is nonnegative and sums to 1, in which case $\|\mathcal{W}\|_{\text{tr}} \leq \|\mathcal{W}\|_1 = 1$. Thus, very surprisingly, Theorem 10 confirms that any deep SPN (or any LTM or HTF_+) can be approximated by some shallow SPN with accuracy ϵ under the ℓ_∞ metric and with at most c/ϵ^2 many product nodes. Of course, this does not contradict with the impossibility results in [Cohen et al., 2016] and [Martens and Medabalimi, 2014], because the accuracy ϵ there is exponentially small.

Theorem 10 remains mostly of theoretical interest, though, because (i) a straightforward application of Theorem 10 leads to a disappointing bound on the total variational distance between the two homogeneous mixtures f and g, due to scaling by the big constant $\prod_i k_i$; (ii) in practical applications we do not have access to $\mathcal{W}$ so the constructive algorithm in our proof does not apply.

KL divergence In contrast to the above ℓ_∞ approximation, we now give an efficient algorithm to approximate a homogeneous density mixture h, using a classic idea of [Li and Barron, 2000]. We propose to estimate h by minimizing the KL divergence over the convex hull[4] of a hypothesis class H:

$$\min_{\mathcal{W}_g \in \text{conv}(\mathsf{H})} \mathsf{KL}(h\|g), \tag{2.43}$$

where $\mathsf{KL}(h\|g) := \int h(\mathbf{x}) \log \frac{h(\mathbf{x})}{g(\mathbf{x})} \mathrm{d}\mu(\mathbf{x})$, and $\mathcal{W}_g$ is the representative tensor for the mixture g. Following [Li and Barron, 2000], we apply the conditional gradient algorithm [Frank and Wolfe, 1956] to solve (4.4): Given g_{t-1}, we find

$$(\eta_t, f_t) \leftarrow \arg \min_{\eta \in [0,1], \mathcal{W}_f \in \mathsf{H}} \mathsf{KL}(h\|(1-\eta)g_{t-1} + \eta f), \qquad g_t \leftarrow (1-\eta_t)g_{t-1} + \eta_t f_t. \tag{2.44}$$

One can also simply set $\eta_t = \frac{2}{2+t}$, as is common in practice. Note that (2.44) can be approximately solved based on an iid sample $\mathbf{x}_1, \ldots, \mathbf{x}_n$ hence is practical:

$$\max_{\eta \in [0,1], \mathcal{W}_f \in \mathsf{H}} \sum_{i=1}^n \log[(1-\eta)g_{t-1}(\mathbf{x}_i) + \eta f(\mathbf{x}_i)]. \tag{2.45}$$

Using basically the same argument as in [Li and Barron, 2000], the above algorithm enjoys the following guarantee:

$$\mathsf{KL}(h\|g_t) \leq c_h \delta / t, \tag{2.46}$$

[4] This is similar in spirit to [Meila and Jordan, 2000, Anandkumar et al., 2012] which learn mixture of trees, but the algorithms are quite different.

where $\delta = \sup\{\log \frac{\langle \mathcal{W}, \vec{f}_1 \otimes \cdots \otimes \vec{f}_d \rangle}{\langle \mathcal{Z}, \vec{f}_1 \otimes \cdots \otimes \vec{f}_d \rangle} : \mathcal{W}, \mathcal{Z} \in \mathsf{H}, \mathbf{x} \in \mathbb{X}\}$, and

$$c_h = \min\{p \geq 0 : \mathcal{W}_h = \sum_{i=1}^{p} \lambda_i \mathcal{W}_i, \mathcal{W}_i \in \mathsf{H}, \boldsymbol{\lambda} \geq 0, \mathbf{1}^\top \boldsymbol{\lambda} = 1\} \tag{2.47}$$

is essentially the rank of the mixture h (with tensor representation $\mathcal{W}_h$) w.r.t. the class H.

The important conclusion we draw from the above bound (2.46) is as follows: First, the constant c_h is no larger than $\prod_i k_i$ if H is any of the classes in Theorem 2 (since we only consider *finite* homogeneous mixtures h). Second, if the target density h is a small number of combinations of densities in H, then c_h is small and we can approximate h using the algorithm (2.44) efficiently. Third, c_h can be vastly different for different hypothesis classes H, as shown in Section 2.4. For instance, if h is a generic TMM and H is the shallow class LCM, then c_h is exponential in d, whereas if H is the class TMM, then c_h can be as small as 1. There is a trade-off though, since solving (2.45) for a simpler class (such as LCM) is easier than a deeper one (such as TMM). We will verify this trade-off in our experiments.

2.6 Experiments

We perform experiments on both synthetic and real world data to reinforce our theoretical findings. Firstly, we present experiments on synthetic data to demonstrate the expressive power of an SPN and the algorithm proposed in (2.44)-(2.45) which we call SPN-CG. Next, we present two sets of experiments on real world datasets and present results for image classification under missing data.

2.6.1 Synthetic Data

In the first experiment, we generated 2000 samples from a 4 component and two dimensional GMM with full covariance matrices for each component. We estimate this GMM using SPN-CG. In each iteration, we add an additional SPN from $\mathsf{TMM}_{2,2}$ and learn its parameters and the mixing weights. We use SGD with weight decay to learn the parameters in each iteration. This experiment helps us to demonstrate that an SPN with univariate leaf distributions (thereby resulting in a mixture model with factored distributions) can estimate a GMM with full covariance matrix. Figure 2.9 shows the convergence of the model to the true negative log-likelihood on a held-out test set as a function of number of iterations.

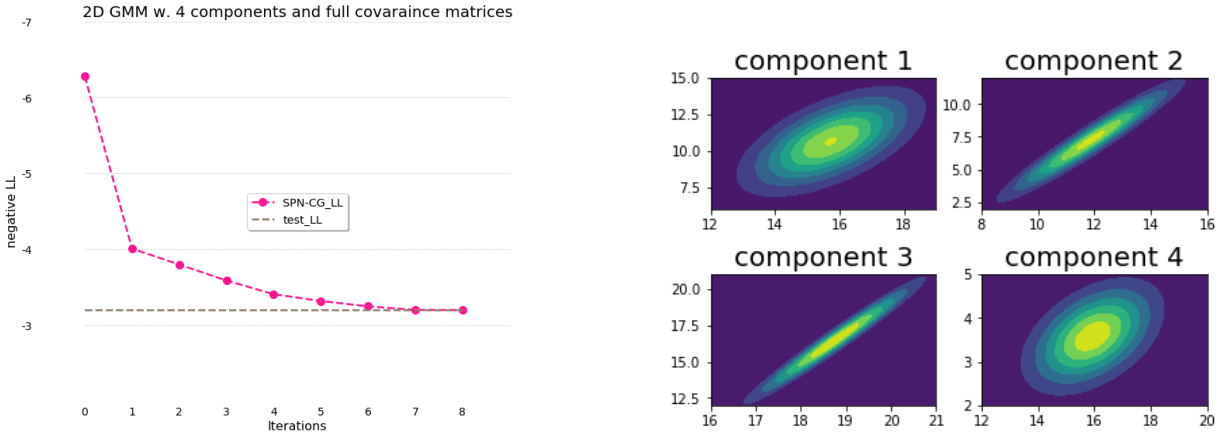


Figure 2.9: (Left) Convergence to true negative log-likelihood using SPN-CG (Right) Surface plots for covariance matrices of the components

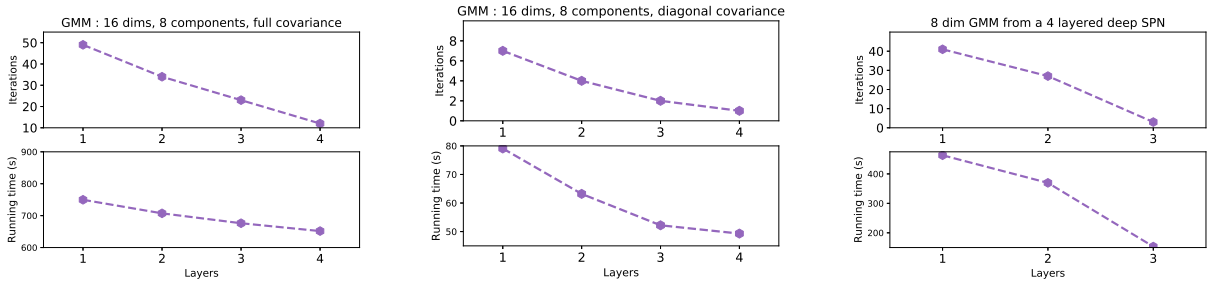


Figure 2.10: Depth efficiency and performance of SPN-CG

In the next set of experiments, we demonstrate the intermediate depth efficiency. We generate 20,000 samples from a 16 dimensional GMM under three different settings - (i) 8 component GMM with full covariance matrices, (ii) 8 component GMM with diagonal covariance matrices and, (iii) GMMs represented by a deep SPN with 4 layers - and estimate each using SPN-CG. We consider layers, $L \in \{1, 2, 3, 4\}$ where $L = 1$ corresponds to a shallow network and $L = 4$ corresponds to a network in TMM (a full binary tree). For each L, at every iteration of SPN-CG we add a network with L layers. In Figure 2.10, we plot the number of iterations and the total running time until convergence w.r.t. the depth for each setting described above. We make the following observations: As the depth (layer) increases, the number of iterations decreases sharply, since adding a deeper network effectively is the same as adding exponentially many shallower networks (confirming Section 2.4). Moreover, although learning a deeper network in each iteration is more expensive

than learning a shallower network, the sharp decrease in iterations full compensates this overhead and leads to a much reduced total running time. The advantage in using deeper networks is more pronounced when the data is indeed generated from a deep model.

2.6.2 Image Classification under Missing Data

A natural setting to test the effectiveness of generative models like deep SPNs is for classification in the regime of missing data. Generative models can cope with missing data naturally through marginalizing the missing values, effectively learning all possible completions for classification. As stated earlier, SPNs are attractive because inference, marginalization and evaluating conditionals is tractable and amounts to one pass through the network. This is in stark contrast with discriminative models that often rely on either data imputation techniques (which result in sub-optimal classification) or by assuming the distribution of missing values is same during train and test time; an assumption that is often not valid in practice.

We perform experiments on MNIST [LeCun et al., 1998] for digit classification and small NORB [LeCun et al., 2004] for 3D object recognition under the MAR (missing at random) regime as described in [Sharir et al., 2018] (Section 3). We experiment with two missing distributions- (i) an i.i.d. mask with a fixed probability of missing each pixel, and (ii) a mask obtained by the union of rectangles of a certain size, each positioned uniformly at random in the image. Concretely, let $P(\mathrm{X}, \mathrm{Y})$ be the joint distribution over the images ($\mathrm{X} \in \mathbb{R}^d$) and labels $\mathrm{Y} \in [M]$. Further, let Z be a random binary vector conditioned on $\mathrm{X} = \mathbf{x}$ with distribution $Q(\mathrm{Z}|\mathrm{X} = \mathbf{x})$. To generate images with missing pixels, we sample $\mathbf{z} \in \{0, 1\}^d$ and consider the vector $\mathbf{x} \odot \mathbf{z}$. A pixel $x_i, i \in [d]$ is considered missing if $z_i = 0$ in which case the corresponding coordinate in $\mathbf{x} \odot \mathbf{z}$ holds $*$ and it holds x_i if $z_i = 1$. In the MAR setting that we consider for our experiments, $Q(\mathrm{Z} = \mathbf{z}|\mathrm{X} = \mathbf{x})$ is a function of both $\mathbf{z}$ and $\mathbf{x}$ but is independent of changes to x_i if $z_i = 0$ i.e. Z is independent of missing pixels. As described in [Sharir et al., 2018], the optimal classification rule in the MAR regime is $h^*(\mathbf{x} \odot \mathbf{z}) = P(\mathrm{Y} = y|w(\mathbf{x}, \mathbf{z}))$ where $w(\mathbf{x}, \mathbf{z})$ is the realization when X coincides with $\mathbf{x}$ on coordinates i for which $z_i = 1$.

Our major goal with these experiments is to test our algorithm SPN-CG for high-dimensional real world settings and show the efficacy of learning SPNs by increasing their expressiveness iteratively. Therefore, we directly adapt the experiments as presented in [Sharir et al., 2018]. Specifically, we adapt the code of HT-TMM for our SPN-CG by following the details in [Sharir et al., 2018]. In each iteration of our algorithm, we add an SPN structure exactly similar to HT-TMM. Therefore, the first iteration of our algorithm (i.e. SPN-CG1) amounts to a structure similar to HT-TMM while additional iterations

increase the network capacity. For each iteration, we train the network using an AdamSGD variant with a base learning rate of 0.03 and momentum parameters $\beta_1 = \beta_2 = 0.9$. For each added network structure, we train the model for 22,000 iterations for MNIST and 40,000 for NORB.

We compare our results to the following methods :

1. **Data Imputation Methods** : Data imputation methods are a common technique to handle missing data using discriminative classifiers. The algorithm proceeds by completing missing values via some heuristic and passing the results to a classifier trained on uncorrupted data. In our approach, we use a ConvNet for prediction. We tested on the following data imputation methods in our manuscript :
    - Zero data imputation : completing all missing values with zeros.
    - Mean data imputation : completing all missing values with the mean value over the dataset
    - Generative Stochastic Networks [Bengio et al., 2014]
    - Non-linear Independent Components Estimation (NICE) [Dinh et al., 2014]
    - Diffusion Probabilistic Models (DPM) [Sohl-Dickstein et al., 2015]
2. **LearnSPN** [Poon and Domingos, 2011]: We used the original code to learn the structure augmented with the class variable and learn the joint probability distribution using CCCP.

For all the algorithms except LearnSPN, we used the modified version of the code as suggested by [Sharir et al., 2018]. Most of the code can be publicly accessed at : https://github.com/HUJI-Deep/Generative-ConvACs. We used the original code as suggested by the authors for LearnSPN. For our algorithm, due to time constraints, we could only perform three iterations for both NORB and MNIST dataset. We present the results for these three iterations denoted in the results as SPN-CG1, SPN-CG2 and, SPN-CG3 (see fig. 2.11a - fig. 2.11d). The results show that SPN-CG performs well in the regime of missing data for both MNIST and NORB. Furthermore, other generative models including SPN with structure learning perform comparably only when a few pixels are missing but perform very poorly as compared to deep mixture models as larger amounts of data is missing suggesting that the structure of deep mixture models is advantageous. These experiments on MNIST and NORB help us conclude that deep mixture models learned using SPN-CG outperform other methods on image classification with missing pixels. Our results compliment the results in [Sharir et al., 2018] where such experiments with state of the art results were presented.

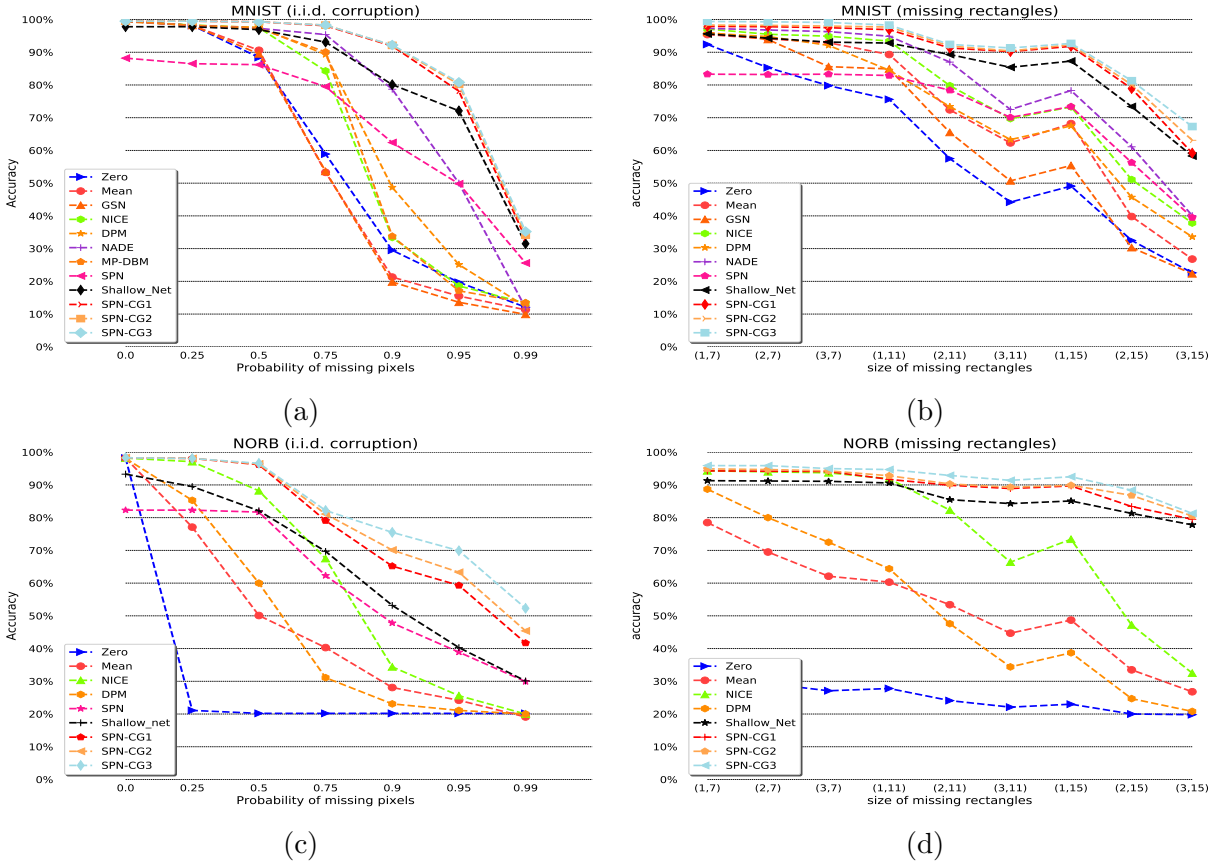


Figure 2.11: Performance of SPN-CG on missing data (a) MNIST data with i.i.d missing pixels (b) MNIST data with rectangles of missing pixels (c) NORB dataset with i.i.d. missing pixels (d) NORB dataset with rectangles of missing pixels

2.7 Connection to Previous Works

The first attempts at rigorously analyzing the effect of depth in a network was in relation to the computational complexity of Boolean circuits. A classical result is due to [Sipser, 1983] who showed that for every integer I, there are Boolean functions computed by a circuit with alternating *AND* and *OR* gates of polynomial size and depth I; but if the depth is reduced to $I-1$, an exponential sized circuit would be required. A similar result was proven later by [Hastad, 1986]. Another body of work in similar spirit was by [Yao, 1985, Ajtai, 1983] proving that solving the k-bit parity task by a circuit of depth 2 requires exponentially many gates. A more recent result is due to [Braverman, 2011] proving that

bounded-depth Boolean circuits cannot distinguish some non-uniform input distributions from the uniform distribution. This work by [Braverman, 2011] solved a longstanding conjecture in the field.

Classical results for analyzing the expressiveness of neural networks involved results on universal approximation by [Cybenko, 1989, Barron, 1993, Hornik et al., 1989], and by [Bartlett, 1998] who studied the networks VC dimension. Although, these early results provided general theoretical insights, they were restricted to shallow networks. Recently, several studies have been undertaken to understand the effect of depth on the expressive capacity of a deep network [Pascanu et al., 2013, Eldan and Shamir, 2016, Telgarsky, 2016, Martens et al., 2013, Lee et al., 2017]. Most of these works provide separation results between the class of functions that can be efficiently represented by a deep network and those by shallow networks. However, one major limitation of these works is they consider pathological hand-coded network weights that exhibit these extremal properties by design. It is not evident if these classes of networks and the hypothesis function class they encode resemble networks and functions used in practice. Therefore, fundamental questions about the expressive power of depth for neural networks used in practice is still not well understood.

Directly relevant to the contributions in this thesis are recent works in analysing the effect of depth in *Arithmetic Circuits* [Darwiche, 2003], *Convolutional Arithmetic Circuits* [Cohen et al., 2016] and particularly in *Sum-Product Networks* [Poon and Domingos, 2011]. The first theoretical results for depth efficiency of SPNs was by [Delalleau and Bengio, 2011]. They constructed two families of functions - $\mathcal{F}$ which formed a full binary tree - and - $\mathcal{G}$ which consisted of n nodes in every layer with each node being connected to $n-1$ nodes in the previous layer - using an SPN with alternating *sum* and *product* layers. Their results establish that any n-dimensional function $f \in \mathcal{F}$ can be computed by an SPN of polynomial size but would require a shallow SPN with $\Omega(2^{\sqrt{n}})$ hidden units to exactly represent f. They further show that for each $d \geq 4$ there exists a function $g_d \in \mathcal{G}$ that can be computed by an SPN of depth d and size $\mathcal{O}(nd)$, but would require a $\Omega(n^d)$ sized shallow SPN to compute. However, this work has several limitations. Firstly, the separation results provided are restricted to depth 3 networks and networks in the family $\mathcal{G}$ and $\mathcal{F}$; it is not clear if a similar separation result holds for intermediate depths. Secondly, as the authors themselves state, it does not comment on any separation results when a deep SPN is only to be *approximated* by a shallow SPN. Thirdly, the specific families of functions $\mathcal{F}$ and $\mathcal{G}$ considered by [Delalleau and Bengio, 2011] are not shown to be a relevant and universal hypothesis class that occurs in practice. Lastly, the SPNs considered in this work are not *valid* SPNs i.e. they do not encode a probability density function. Furthermore, the analysis is limited to only discrete variables with indicator leaf functions.

[Martens and Medabalimi, 2014] extended the work in [Delalleau and Bengio, 2011] by proving that there exist functions that can be efficiently computed by a depth d *valid* SPN but would require super-polynomially size for a depth $d-1$ SPN. In particular, they considered the EQUAL function on an array of Boolean variables $\mathbf{x} = (x_1, x_2, .., x_n)$ defined as follows : let $\mathcal{A} = \{1, 2, 3, .., n/2\}$ and $\mathcal{B} = (n/2+1, n/2+2, ..., n)$ be the index partition. Then, EQUAL : $\{0,1\}^n \rightarrow \{0,1\}$ where EQUAL$(\mathbf{x}) = 1$ when $\mathbf{x}_{\mathcal{A}} = \mathbf{x}_{\mathcal{B}}$ (i.e. the first half of the input is equal to the second half) and 0 otherwise. They proved that a *valid* shallow SPN would require $2^{\frac{n}{2}}$ units in the hidden layer to exactly represent EQUAL$(\mathbf{x})$ while an SPN of depth 4 would require $\mathcal{O}(n)$ size. Further, they also proved that a shallow SPN would still require $2^{n/2-2}$ nodes in the hidden layer to approximate EQUAL$(\mathbf{x})$[5]. However, [Martens and Medabalimi, 2014] also restricted their analysis in the paper to only Boolean variables primarily because they used previous works from circuit complexity on arithmetic circuits to derive their results. Further, for the separation results, they constructed an example restricted to Boolean variables and indicator functions in the leaves; the proof does not generalize to *valid* SPNs with arbitrary density functions. Most importantly, they use very specific hand-crafted functions to prove separation results both for exact and approximate representation with no information on how frequently these functions occur in practice. Therefore, it might be the case that except for a few hand-crafted pathological examples, a shallow SPN can efficiently represent all functions derived from a deep SPN.

Recently, [Cohen et al., 2016] proposed a deep network which they called *convolutional arithmetic circuits* that incorporates locality, sharing and pooling. They went on to show that this network corresponded to the Hierarchical Tucker Tensor decomposition [Hackbusch, 2012]. Their main theoretical result showed that except for a negligible set of measure zero, all functions that can be represented by a deep convolutional arithmetic circuits of polynomial size, require an exponential sized shallow network to be realized exactly or approximated. The hypothesis class they considered was universal. However, a major limitation of their main result is that it is an existence result. That is, they say, for any deep convolutional arithmetic circuit, there exists an ϵ such that no shallow network of polynomial size can approximate the deep network within this ϵ distance. However, they do not provide any explicit relation with ϵ for the approximation. Therefore, a natural question to ask is : what is the ϵ-dependency of the size of a shallow network approximating a deep network within some ϵ distance?

[5] We direct the reader to [Martens and Medabalimi, 2014] for further details on the exact definition of approximation used and the proof.

2.8 Summary

In this chapter, we formally established the relationships among some popular unsupervised learning models, such as latent tree graphical models, hierarchical tensor formats and sum-product networks, based on which we further provided a unified treatment of exponential separation in *exact* representation size between deep architectures and shallow ones. Surprisingly, for *approximate* representation, the conditional gradient algorithm can approximate any homogeneous mixture within accuracy ϵ by combining $O(1/\epsilon^2)$ shallow models, where the hidden constant may decrease exponentially w.r.t. the depth. Finally, experiments on both synthetic and real datasets confirmed our theoretical findings.

Chapter 3

Bayesian Moment Matching

We saw in Chapter 2 that finite mixture models are a powerful tool for compact, interpretable and flexible representation of density functions in unsupervised learning that have been used successfully in several applications. However, with huge amounts of data being generated, there is a need for parameter estimation techniques for these mixture models that can handle streaming data and distribute the computation over several processors. Bayes' theorem provides a natural framework for updating model parameters in an online manner and compute partial posteriors over several processors [Broderick et al., 2013] that can be combined into a single exact posterior. In Chapter 1, we saw briefly that Bayesian learning is a paradigm to update one's belief about the parameters of interest given a prior belief system and observed data through density functions using Bayes' rule. However, a major challenge in Bayesian learning is that the updated belief (henceforth posterior distribution) becomes complex with the availability of new observations. This necessitates the deployment of algorithms that can approximate the true posterior by minimizing some distance metric between the approximation and the true posterior. This motivates us to develop online and distributed Bayesian techniques for finite mixture models.

In this chapter, we will first propose an online and distributed parameter estimation algorithm for Gaussian mixture models called the Bayesian Moment Matching algorithm (BMM) that approximates the true posterior by matching a set of moments of the true posterior with the approximate posterior. Subsequently, we will extend this algorithm for online Bayesian transfer learning for sequential data modeling with applications to diverse problems like activity recognition, network flow size prediction and sleep stage classification for predicting neurological disorders.

The results in this chapter appeared in [Jaini and Poupart, 2017] and [Jaini et al.,

2017].

3.1 Bayesian Moment Matching for Gaussian Mixture Models

Gaussian Mixture models (GMMs) [Murphy, 2012] are simple, yet expressive distributions that are often used for soft clustering and data modeling. Traditionally, the parameters of GMMs are estimated by batch Expectation Maximization (EM) [Dempster et al., 1977]. However, as datasets get larger and do not fit in memory or are continuously streaming, several online variants of EM have been proposed [Titterington, 1984, Neal and Hinton, 1998, Cappé and Moulines, 2009, Liang and Klein, 2009]. They process the data in one sweep by updating a sufficient statistics in constant time after each observation, however this update is approximate and stochastic, which slows down the learning rate. Furthermore it is not clear how to distribute the computation over several processors given the sequential nature of those updates.

As pointed out by [Broderick et al., 2013], Bayes' theorem can be applied after each observation to update the posterior in an online fashion and a dataset can be partitioned into subsets that are each processed by different processors to compute partial posteriors that can be combined into a single exact posterior that corresponds to the product of the partial posteriors divided by their respective priors.

The main issue with parameter estimation using Bayesian learning for Gaussian mixture models is that the posterior is intractable to compute and represent exactly. Let $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\} \subseteq \mathbb{R}^d$ be a set of n data points sampled i.i.d. from a Gaussian mixture model with K components i.e.

$$\mathbf{x}_i \sim \sum_{j=1}^{K} w_j \mathcal{N}(\boldsymbol{\mu}_j, \Sigma_j), \qquad \forall i \in [n] \tag{3.1}$$

where $\forall j \in [K]$, $w_j > 0$ is the weight, $\boldsymbol{\mu}_j \in \mathbb{R}^d$ is the mean and $\Sigma_j \in \mathbb{R}^{d \times d}$ is the covariance matrix of the j^{th} component in the mixture model. Let $\Theta = \{\theta_1, \theta_2, ..., \theta_M\}$ where $\theta_i = (w_i, \boldsymbol{\mu}_i, \Sigma_i)$, $\forall i \in [K]$. Given $\mathcal{D}$, we wish to estimate Θ in an online fashion. In

a Bayesian learning framework, this can be formulated as follows:

$$\begin{aligned} p_n(\Theta) &= p(\Theta|\mathcal{D}) \\ &\propto p_{n-1}(\Theta) \cdot p(\mathbf{x}_n|\Theta) = p(\Theta|\mathcal{D} \setminus \mathbf{x}_n) \cdot p(\mathbf{x}_n|\Theta) \\ &\propto p(\Theta|\mathcal{D} \setminus \mathbf{x}_n) \cdot \sum_{j=1}^{K} w_j \mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}_j, \Sigma_j) \\ &\propto p(\Theta) \cdot \prod_{i=1}^{n} \sum_{j=1}^{K} w_j \mathcal{N}(\mathbf{x}_i; \boldsymbol{\mu}_j, \Sigma_j) \end{aligned}$$

Subsequently, a point estimate of Θ can be obtained by $\hat{\Theta} = \mathbb{E}_{p_n}[\Theta]$. However, notice that with each new data point $\mathbf{x}_j$, the number of terms in the posterior increases by a factor K due to the summation over the number of components. Hence, after n data points, the posterior will consist of a mixture of K^n terms, which is intractable.

Algorithm 1 Generic BMM [Omar, 2016, Section 4.2]

$\mathcal{F}(\Phi)$:= a class of probability densities with hyperparameters Φ
Input: Dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$
Output: Estimated parameters $\hat{\Theta}$
Initialize: prior $p_0(\Theta) \in \mathcal{F}(\Phi)$
for i = 1 : n **do**
  Compute $p_n(\Theta)$ using $p_{n-1}(\Theta)$ and $Pr(\mathbf{x}_n|\Theta)$
  $\forall\ f \in S(\mathcal{F})$, evaluate $\mathbb{E}_{p_n}[f]$
  Compute $\hat{\Phi}$ using $\mathbb{E}_{p_n}[f]$, $f \in S(\mathcal{F})$
  Approximate $p_n(\Theta)$ with $\tilde{p}_n(\Theta) \in \mathcal{F}(\hat{\Phi})$
Return : $\hat{\Theta} = \mathbb{E}[\tilde{p}_n]$

The Bayesian Moment Matching (BMM) algorithm approximates the posterior obtained after each iteration in a manner that prevents the exponential growth of mixture terms. This is achieved by approximating the posterior distribution $p_n(\Theta)$ by another distribution $q_n(\Theta)$ which is in the same family of distributions $\mathcal{F}(\Phi)$ as the prior by matching a set of sufficient moments S of $p_n(\Theta)$ with $q_n(\Theta)$. Algorithm 1 describes a generic procedure to approximate the posterior p_n after each observation with a simpler distribution q_n by moment matching. More precisely, a set of moments sufficient to define q_n are matched with the moments of the exact posterior p_n. For every iteration, we first calculate the posterior distribution $p_n(\Theta|\mathcal{D})$ using the approximate posterior $q_{n-1}(\Theta)$ from the previous

step acting as the prior. We then compute the set of moments $S(\mathcal{F})$ that are sufficient to define a distribution in the family $\mathcal{F}(\Phi)$. Next, we compute the parameter vector Φ based on the set of sufficient moments. This determines a specific distribution q_n in the family $\mathcal{F}$ that we use to approximate p_n.

We now illustrate the BMM algorithm for estimating parameters of a multivariate Gaussian mixture model.

We choose the prior p_0 over the parameters Θ as a product of a Dirichlet distribution over the weights $\mathbf{w} = \{w_1, w_2, \cdots, w_K\}$ and K Normal-Wishart distributions corresponding to the parameters $(\boldsymbol{\mu}_j, \Lambda_j^{-1}),\ \forall\ j \in [K]$ of each Gaussian component i.e. $p_0(\Theta) = Dir(\mathbf{w}|\boldsymbol{\alpha}) \prod_{i=1}^{K} \mathcal{NW}(\boldsymbol{\mu}_i, \Lambda_i|\boldsymbol{\delta}_i, \kappa_i, \mathbf{W}_i, \nu_i)$ where $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, ..., \alpha_K)$, $\mathbf{W}$ is a symmetric positive definite matrix, $\boldsymbol{\delta} \in \mathbb{R}^d$ and $\kappa > 0$ and $\nu > d - 1$ are real. This prior distribution forms a conjugate probability pair of the likelihood function.

The posterior $p_1(\Theta|\mathbf{x}_1)$ after observing the first data point $\mathbf{x}_1$ is given by

$$p_1(\Theta|\mathbf{x}_1) \propto Dir(\mathbf{w}|\boldsymbol{\alpha}) \cdot \prod_{i=1}^{K} \mathcal{NW}(\boldsymbol{\mu}_i, \Lambda_i|\boldsymbol{\delta}_i, \kappa_i, \mathbf{W}_i, \nu_i) \cdot \sum_{j=1}^{K} w_j \mathcal{N}\big(\mathbf{x}_1; \boldsymbol{\mu}_j, \Lambda_j^{-1}\big) \tag{3.2}$$

$$\propto \sum_{j=1}^{K} \Big(w_j \cdot Dir(\mathbf{w}|\boldsymbol{\alpha})\Big) \Big(\prod_{i=1}^{K} \mathcal{NW}(\boldsymbol{\mu}_i, \Lambda_i|\boldsymbol{\delta}_i, \kappa_i, \mathbf{W}_i, \nu_i) \cdot \mathcal{N}\big(\mathbf{x}_1; \boldsymbol{\mu}_j, \Lambda_j^{-1}\big)\Big) \tag{3.3}$$

For simplification, we evalaute the two expressions in the brackets individually below:

$$\begin{aligned} w_j Dir(\mathbf{w}, \boldsymbol{\alpha}) &= \frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)} w_j \prod_i w_i^{\alpha_i} \\ &= \frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)} w_j^{\alpha_j + 1} \prod_{i \neq j} w_i^{\alpha_i} \\ &= \frac{\alpha_j}{\sum_i \alpha_i} Dir(\mathbf{w}; \hat{\boldsymbol{\alpha}}) \end{aligned}$$

where

$$\hat{\alpha}_i = \begin{cases} \alpha_i & \text{if } i \neq j \\ \alpha_i + 1 & \text{if } i = j \end{cases}$$

For the second term we note that a product of a Normal-Wishart distribution and a Gaussian distribution with the same mean and precision matrix is a Normal-Wishart

distribution since they form a conjugate pair. Formally,

$$\mathcal{N}\big(\mathbf{x};\boldsymbol{\mu},(\kappa\boldsymbol{\Lambda})^{-1}\big)\mathcal{NW}(\boldsymbol{\mu},\boldsymbol{\Lambda};\boldsymbol{\mu}_0,\kappa,\mathbf{W},\nu)=c\,\mathcal{NW}(\boldsymbol{\mu},\boldsymbol{\Lambda};\boldsymbol{\mu}_0^*,\kappa^*,\mathbf{W}^*,\nu^*)$$

where

$$\begin{aligned}\boldsymbol{\mu}_0^* &= \frac{\kappa\boldsymbol{\mu}_0+\mathbf{x}}{\kappa+1}\\ \kappa^* &= 1+\kappa\\ \nu^* &= \nu+1\\ \mathbf{W}^* &= \mathbf{W}+\frac{\kappa}{\kappa+1}(\boldsymbol{\mu}_0-\mathbf{x})(\boldsymbol{\mu}_0-\mathbf{x})^T\end{aligned}$$

Therefore, $p_1(\Theta|\mathbf{x}_1)$ can now be written as:

$$p_1(\Theta|\mathbf{x}_1)=\frac{1}{Z}\sum_{j=1}^{M}\left(c_j Dir(\mathbf{w}|\hat{\boldsymbol{\alpha}}_j)\mathcal{NW}(\boldsymbol{\mu}_j,\Lambda_j|\hat{\boldsymbol{\delta}}_j,\hat{\kappa}_j,\hat{\mathbf{W}}_j,\hat{\nu}_j)\prod_{i\neq j}^{M}\mathcal{NW}(\boldsymbol{\mu}_i,\Lambda_i|\boldsymbol{\delta}_i,\kappa_i,\mathbf{W}_i,\nu_i)\right)$$

where $\hat{\boldsymbol{\alpha}}_j=(\alpha_1,\alpha_2,..,\hat{\alpha}_j,..,\alpha_M)$ and Z is the normalization constant. The equation above suggests that the posterior is a mixture of product of distributions where each product component in the summation has the same form as that of the family of distributions of the prior $p_0(\Theta)$. It is evident that the terms in the posterior grow by a factor of M for each iteration, which is problematic. The Bayesian moment matching algorithm approximates this mixture $p_1(\Theta)$ with a single product of Dirichlet and Normal-Wishart distributions $\tilde{p_1}(\Theta)$ by matching all the sufficient moments of p_1 with $\tilde{p_1}$ which belongs to the same family of distributions as the prior:

$$\tilde{p_1}(\Theta)=Dir(\mathbf{w}|\boldsymbol{\alpha}^1)\prod_{i=1}^{M}\mathcal{NW}(\boldsymbol{\mu}_i,\Lambda_i|\boldsymbol{\delta}_i^1,\kappa_i^1,\mathbf{W}_i^1,\nu_i^1)$$

We evaluate the parameters $\boldsymbol{\alpha}^1,\boldsymbol{\delta}_i^1,\kappa_i^1,\mathbf{W}_i^1,\nu_i^1\ \forall i\in\{1,2,..,M\}$ by matching a set of sufficient moments of $\tilde{P}_1(\Theta)$ with $P_1(\Theta)$. The set of sufficient moments in this case is $S=\{\boldsymbol{\mu}_j,\boldsymbol{\mu}_j\boldsymbol{\mu}_j^T,\boldsymbol{\Lambda}_j,\Lambda_{j_{km}}^2,w_j,w_j^2\}\ \forall j\in 1,2,...,M$ where $\Lambda_{j_{km}}^2$ is the $(k,m)^{th}$ element of the matrix $\boldsymbol{\Lambda}_j$. The expressions for sufficient moments are given by $\mathbb{E}[g]=\int_\Theta gP_1(\Theta)d(\Theta)$ leading to the following equations:

$$\begin{aligned}\mathbb{E}[w_i]=\frac{\alpha_i}{\sum_j\alpha_j};\qquad & \mathbb{E}[w_i^2]=\frac{(\alpha_i)(\alpha_i+1)}{\Big(\sum_j\alpha_j\Big)\Big(1+\sum_j\alpha_j\Big)}\\ \mathbb{E}[\boldsymbol{\Lambda}]=\nu\mathbf{W};\qquad & Var(\boldsymbol{\Lambda}_{ij})=\nu(\mathbf{W}_{ij}^2+\mathbf{W}_{ii}\mathbf{W}_{jj})\\ \mathbb{E}[\boldsymbol{\mu}]=\boldsymbol{\delta};\qquad & \mathbb{E}[(\boldsymbol{\mu}-\boldsymbol{\delta})(\boldsymbol{\mu}-\boldsymbol{\delta})^T]=\frac{\kappa+1}{\kappa(\nu-d-1)}\mathbf{W}^{-1}\end{aligned}$$

The set of equations for evaluating the parameters of the approximate posterior $\tilde{p_1}$ from the estimated moments are as follows:

$$\alpha_i = \mathbb{E}[w_i]\frac{\mathbb{E}[w_i] - \mathbb{E}[w_i^2]}{\mathbb{E}[w_i^2] - \mathbb{E}[w_i]^2}$$
$$\boldsymbol{\delta} = \mathbb{E}[\boldsymbol{\mu}]$$
$$\mathbf{W}_{ii} = \frac{Var(\boldsymbol{\Lambda}_{ii})}{\mathbb{E}[\boldsymbol{\Lambda}_{ii}]}$$
$$\mathbf{W}_{ij} = \frac{Var(\boldsymbol{\Lambda}_{ij})}{\mathbb{E}[\boldsymbol{\Lambda}_{ij}]}$$
$$\nu = \frac{\mathbb{E}[\boldsymbol{\Lambda}_{ij}]}{\mathbf{W}_{ij}}$$
$$\kappa = 1 - \Big((\nu - d - 1)\mathbb{E}[(\boldsymbol{\mu} - \boldsymbol{\delta})(\boldsymbol{\mu} - \boldsymbol{\delta})^T])\mathbf{W}\Big)^{-1}$$

A major advantage of Bayes' theorem is that the computation of the posterior can be distributed over several machines, each of which processes a subset of the data. It is also possible to compute the posterior in a distributed manner using Bayesian moment matching algorithm. For example, let us assume that we have T machines and a data set with TN data points. Each machine $t \in [T]$ can compute the approximate posterior $p_t(\boldsymbol{\Theta}|\mathbf{x}^{(t-1)N+1:tN})$ where $t \in [T]$ over N data points. These partial posteriors $\{P_t\}_{t=1}^T$ can be combined to obtain a posterior over the entire data set $\mathbf{x}^{1:TN}$ [Broderick et al., 2013] according to the following equation:

$$P(\boldsymbol{\Theta}|\mathbf{x}^{1:TN}) = P(\boldsymbol{\Theta})\prod_{t=1}^{T}\frac{P_t(\boldsymbol{\Theta}|\mathbf{x}^{(t-1)N+1:tN})}{P(\boldsymbol{\Theta})} \tag{3.4}$$

Subsequently, the estimate $\hat{\boldsymbol{\Theta}} = \mathbb{E}[P(\boldsymbol{\Theta}|\mathbf{x}^{1:TN})]$ is obtained over the whole data set.

3.1.1 Experiments

We compared the performance of online Bayesian Moment Matching algorithm (oBMM) with the online Expectation Maximization algorithm (oEM) described in [Cappé and Moulines, 2009] and online Variational Bayes (oVB) [Hoffman et al., 2013, Beal, 2003] by performing experiments on 2 sets of real datasets - 10 moderate-to-small size datasets

and 4 large datasets available publicly online at the UCI machine learning repository [Dua and Graff, 2017] and Function Approximation repository [Guvenir and Uysal, 2000] spanning diverse domains.

We measure both - the quality of the algorithms in terms of average log-likelihood scores on the held-out test datasets and their scalability in terms of running time. We use the Wilcoxon signed ranked test [Wilcoxon, 1950] to compute the p-value and report statistical significance with p-value less than 0.05, to test the statistical significance of the results. We computed the parameters for each algorithm over a range of components varying from 2 to 10 and report the best performing model. For oEM the step size for the stochastic approximation in the E-Step was set to $(n+3)^{-\alpha}$ where $0.5 \leq \alpha \leq 1$ [Liang and Klein, 2009] where n is the number of observations. We evaluated the performance of Distributed Moment Matching (DMM) by dividing the training datasets in to 5 smaller data sets, and processing each of these small datasets on a different machine. The output from each machine is collected and combined to give a single estimate for the parameters of the model learned.

Table 3.1: Log-likelihood scores on 10 data sets. The best results among oBMM and oEM are highlighted in bold font. ↑(or ↓) indicates that the method has significantly better (or worse) log-likelihoods than Online Bayesian Moment Matching (oBMM) under Wilcoxon signed rank test with pvalue < 0.05.

Data set	Instances	oVB	oEM	oBMM
Abalone	4177	2.18↓	-2.65 ↓	**-1.82**
Banknote	1372	9.89 ↓	-9.74 ↓	**-9.65**
Airfoil	1503	16.71 ↓	**-15.86** ↑	-16.53
Arabic	8800	15.42↓	-15.83 ↓	**-14.99**
Transfusion	748	13.31 ↓	-13.26 ↓	**-13.09**
CCPP	9568	16.87↓	-16.53 ↓	**-16.51**
Comp. Activity	8192	-121.55↓	-132.04 ↓	**-118.82**
Kinematics	8192	10.51 ↓	-10.37 ↓	**-10.32**
Northridge	2929	19.03↓	-18.31 ↓	**-17.97**
Plastic	1650	9.39 ↓	-9.47 ↓	**-9.01**

Table 3.1 shows the average log-likelihood on test sets for oVB, oEM and oBMM. oBMM outperforms oEM on 9 of the 10 datasets and it outperformed oVB on all datasets. The results show that for some datasets, oBMM has significantly better log-likelihoods than both oEM and oVB. Table 3.2 shows the log-likelihood scores and running times of

each algorithm on large datasets. In terms of log-likelihood scores, oBMM outperforms oEM, oVB and oDMM on all 4 datasets. While, the performance of oDMM is expected to be worse than oBMM, its performance was not significantly worse. This is encouraging in light of the huge gains in terms of running time of oDMM over oVB, oEM and oBMM. Table 3.2 shows the performance of each algorithm in terms of running times.

Table 3.2: Log-likelihood scores and Avg. running time on 4 large data sets. The best results among oBMM, oDMM and oEM are highlighted in bold font. The results for oDMM are only for a single run to demonstrate the savings in running time.

Data		Avg. Log-Likelihood				Avg. Running Time (sec)			
Data (Attributes)	Instances	oVB	oEM	oBMM	oDMM	oVB	oEM	oBMM	oDMM
Heterogeneity (16)	3930257	-175.3↓	-176.2↓	**-174.3**	-180.7	87.3	77.3	81.7	**17.5**
Magic 04 (10)	19000	-32.9↓	-33.4↓	**-32.1**	-35.4	8.1	7.3	6.8	**1.4**
Year MSD (91)	515345	-514.6↓	-513.7↓	**-506.5**	-513.8	473.7	336.5	108.2	**21.2**
MiniBooNe (50)	130064	-58.6↓	-58.1↓	**-54.7**	-60.3	57.6	48.6	12.1	**2.3**

3.2 Online Bayesian Transfer Learning for Sequential Data Modeling

In the previous section, we introduced a Bayesian Moment Matching algorithm for Gaussian mixture models. We will now extend the algorithm for domains that generate a sequence of observations. In several application domains, data instances are produced by a population of individuals that exhibit a variety of different characteristics. For instance, in activity recognition, different individuals might walk or run with different gait patterns. Similarly, in sleep studies, different individuals might exhibit different patterns for the same sleep stages. In telecommunication networks, software applications might generate packet flows between two servers according to different patterns. In such scenarios, it is tempting to treat the population as a homogeneous source of data and to learn a single average model for the population. However, this average model will perform poorly in recognition tasks for individuals that differ significantly from the average. Hence, there is a need for transfer learning techniques that take into account the variations between individuals within a population.

Indeed, there is a large literature on transfer learning [Pan and Yang, 2010, Taylor and Stone, 2009, Shao et al., 2015, Cook et al., 2013]. Depending on the problem, the input features, the output labels or the distribution over the features and the labels may be different for the source and target domains. In this work, we assume that the same input features are measured and the same output labels are inferred in the source and target domains. The main problem that we consider is *subject variability* within a population of individuals, which means that different individuals exhibit different distributions over the features and the labels. The problem of subject variability has been studied in several papers. [Chieu et al., 2006] describe how to augment conditional random fields with a subject hidden variable to obtain a mixture of conditional random fields that can naturally infer a distribution over the closest subjects in a training population when inferring the activities of a new individual based on physiological data. [Rashidi and Cook, 2009] proposed a data mining technique with a similarity measure to facilitate the transfer of activity recognition across different people. [Chattopadhyay et al., 2011] describe a similarity measure with an intrinsic manifold that preserve the topology of surface electromyography (SEMG) while mitigating distributional differences among individuals. [Zhao et al., 2011] proposed a transfer learning technique that starts by training a decision tree to recognize the activities of a user based on smartphone accelerometry. The decision tree is gradually adjusted to a new user by a clustering technique that successively re-weights the training data based on the unlabeled data of the new individual. These approaches mitigate subject variability by various *offline* transfer learning techniques. In contrast, we propose an *online* transfer learning technique aimed at applications which exhibit sequences of observations that arrive in a streaming fashion and therefore require an online technique that can infer the hidden state of each observation as it arrives.

We first propose an online Bayesian moment matching technique (§3.1) to estimate the parameters of a hidden Markov model (HMM) with observation distributions represented by Gaussian mixture models (GMMs). This approach allows us to learn separate basis models for different individuals based on streaming data. The second contribution is an unsupervised online technique that infers a probability distribution over the basis models that best models the sequence of observations of a new individual. The approach learns different transition and emission models for each individual in the training population. Those models are then treated as basis models to speed up the online learning process for new individuals. More specifically, a weighted combination of the basis models is learned for each new individual. Furthermore, since the basis models are fixed at classification time and we only learn the weight of each model, good classification accuracy can be obtained more quickly as the stream of observations of the new individual are processed. This idea is related to boosting techniques for transfer learning [Dai et al., 2007, Yao and Doretto,

2010, Al-Stouhi and Reddy, 2011] that estimate a weighted combination of base classifiers. However, we focus on sequence modeling problems where the classes of consecutive data points are correlated while transfer learning by boosting assumes that the data points are identically and independently distributed. Finally, we demonstrate the efficacy of this approach across different real-world applications, which include activity recognition, sleep classification and the prediction of packet flow direction in telecommunication networks.

3.2.1 Problem Setup

We motivate our problem through an example of activity recognition in which different individuals will have different gait patterns despite the fact that they are performing the same activity (e.g. walking, running, standing, etc.). It is therefore problematic to make predictions in such domains by considering the population to be homogeneous. However, every population will have individuals resembling each other in some characteristics. This suggests that we can use individuals in a population to make predictions about similar individuals by identifying those individuals who closely resemble each other. However, identifying individuals with similar traits is not straightforward. Alternatively, weights can be assigned to each individual in a population based on a target individual (individual on whom predictions are to be made). All those individuals who closely resemble the target individual will receive higher weights than those with dissimilar traits. Subsequently, predictions about the behavior of the target individual can be based mostly on the observed behavior of the similar individuals.

Formally, let $\mathbf{X}_{1:T} = \{\mathbf{X}_1, \mathbf{X}_2, \cdots, \mathbf{X}_T\} \subseteq \mathbb{R}^d$ be a sequence of observations for T time-steps with corresponding hidden state labels given by $Y_{1:T} = (Y_1, Y_2, \cdots, Y_T)$ such that $\forall\ t \in [T],\ Y_t \in [N]$. We are given access to a training set of observations from a population of K different individuals (henceforth referred to as different source domains) where each source domain consists of a labeled sequence data given by the pair $\{(\mathbf{X}_t^k, Y_t^k)\}_{t=1}^T$[1]. Given a new individual (called target domain) not present in the source domain, we aim to make predictions given its observations based on the source domain in an online manner. Our transfer learning algorithm addresses precisely these issues. It consists of three steps:

Step 1 - Source domain training: We first learn a hidden Markov model (i.e. transition and emission distributions) for each source domain that best explain the observations

[1]For simplicity in notations we assume that each source domain has the same number of observations but this is not a required assumption for the algorithm.

of that source domain. In a hidden Markov model (HMM), each observation $\mathbf{X}_t$ is associated with a hidden state Y_t. The Markov property states that the current state depends only on the previous state. HMMs have been widely used in domains involving sequential data like speech recognition, activity recognition, natural language processing etc. An HMM is represented by two distributions:

- **Transition distribution**: The transition distribution models the change in the value of the hidden state over time. The distribution over the current state Y_t given that the previous state is $Y_{t-1} = j$ is denoted by $\theta_j = \Pr(Y_t|Y_{t-1} = j)$ where $\theta_j = \{\theta_{1j}, ..., \theta_{Nj}\}$, N is the total number of states and $\theta_{ij} = \Pr(Y_t = i|Y_{t-1} = j)$.
- **Emission distribution**: The emission distribution models the effect of the hidden state on the observation X_t at any given time t and is given by $\Pr(X_t|Y_t)$.

We model the emission distribution as a mixture of Gaussians with M components, i.e., $\Pr(X_t|Y_t = j) = \sum_{m=1}^{M} w_{j,m}\mathcal{N}\big(\mathbf{X}_t; \boldsymbol{\mu}_{j,m}, \Sigma_{j,m}\big)$. The training step therefore involves estimating the parameters of the transition distributions and the parameters of the Gaussian mixture model for the emission distributions. We estimate these parameters using Bayesian learning which is done by calculating the posterior over the parameters given a prior distribution.

$$\Pr\Big(\Theta, \Phi, Y_t = j|X_t, Y_{t-1} = i\Big) \propto \overbrace{\Pr(X_t|Y_t = j)}^{\text{Emission distribution}} \overbrace{\Pr(Y_t = j|Y_{t-1} = i)}^{\text{Transition Probability}} \overbrace{\Pr(\Theta, \Phi, Y_{t-1} = i|X_{1:t-1})}^{\text{Prior for } t-1}$$

$\forall j \in [N]$ where Θ and Φ parametrize the transition and emission distributions respectively.

Step 2 - Target domain learning: In the next step, we learn a model over the target domain by modeling it as a mixture of the learned source domain models i.e. we model the hidden Markov model for the target domain as a mixture of transition distributions and a mixture of emission distributions of the source domain models. At each time step, we update the target model by updating the weights of these mixture distributions. The updated weights identify those individuals in the source domain that closely resemble the target individual. A higher weight for a source domain implies that the corresponding individual resembles the target individual more closely.

Step 3 - Prediction: Finally, we make predictions about the hidden states for each observation in the target domain by using the models learned in the source domain and

the updated basis weights that are given to each transition and emission distribution of the source domains.

We now explain each of these steps in more detail.

3.2.2 Source Domain - Training

The first step involves learning a model for each source domain in the training data given a labeled sequence of data for K different source domains:

$$Y_t^k = \text{hidden state label at time step } t \text{ for source domain } k$$
$$\mathbf{X}_t^k = \text{feature vector at time step } t \text{ for source domain } k$$

We define

$$\Theta_{ij}^k = \Pr(Y_t^k = i | Y_{t-1}^k = j) \text{ i.e. the transition probability from state } i \text{ to state } j$$

and denote the transition matrix for the k^{th} source domain with Θ^k. The emission distribution is modeled by a mixture of Gaussian with M components, i.e.

$$\Pr(\mathbf{X}_t^k | Y_t^k = j) = \sum_{m=1}^{M} w_{j,m}^k \mathcal{N}(\mathbf{X}_t^k | \boldsymbol{\mu}_{j,m}^k, \Sigma_{j,m}^k), \quad \forall\ j \in [N], \quad \text{and} \quad m \in [M]$$

Let

$$\Phi^k = \{\phi_1^k, \phi_2^k, ..., \phi_N^k\} \text{ where } \phi_j^k = \{(w_{j,1}^k, \boldsymbol{\mu}_{j,1}^k, \Sigma_{j,1}^k), ..., (w_{j,M}^k, \boldsymbol{\mu}_{j,M}^k, \Sigma_{j,M}^k)\}, \quad j \in [N],\ m \in [M]$$

We want to learn the parameters Θ^k for the transition distribution and Φ^k for the emission distribution for each source domain $k \in [K]$. Since, we use a hidden Markov model, the update equation at each time step for a source domain k is

$$Pr\Big(\Theta, \Phi, Y_t^k = j | X_t^k, Y_{t-1}^k = i\Big) \propto \overbrace{\Pr(X_t^k | Y_t^k = j)}^{\text{Emission distribution}} \overbrace{\Pr(Y_t^k = j | Y_{t-1}^k = i)}^{\text{Transition Probability}} \overbrace{\Pr(\Theta^k, \Phi^k, Y_{t-1}^k = i | X_{1:t-1}^k)}^{\text{Prior for } t-1}$$
$$\forall j \in \{1, 2, ..., N\} \quad (3.5)$$

The prior over (Θ^k, Φ^k) is given by

$$\Pr(\Theta^k, \Phi^k) = \Big[\prod_{i=1}^{N} Dir(\theta_i^k | \boldsymbol{\alpha}_i^k)\Big] \Big[\prod_{j=1}^{N} Dir(\mathbf{w}_j^k; \boldsymbol{\beta}_j^k) \prod_{u=1}^{M} \mathcal{NW}(\boldsymbol{\mu}_{j,u}^k, \boldsymbol{\Lambda}_{j,u}^k; \boldsymbol{\delta}_{j,u}^k, \kappa_{j,u}^k, \mathbf{W}_{j,u}^k, v_{j,u}^k)\Big] \quad (3.6)$$

After substituting the relevant terms in Eq (3.5), we get

$$\Pr\Big(\Theta, \Phi, Y_t^k = j | X_t^k, Y_{t-1}^k = i\Big) \propto \sum_{m=1}^{M} w_{j,m}^k \mathcal{N}(X_t^k | \boldsymbol{\mu}_{j,m}^k, \Sigma_{j,m}^k) \theta_{ji}^k \Big[\prod_{i=1}^{N} Dir(\theta_i^k | \boldsymbol{\alpha}_i^k)\Big]$$
$$\Big[\prod_{j=1}^{N} Dir(\mathbf{w}_j^k; \boldsymbol{\beta}_j^k) \prod_{u=1}^{M} \mathcal{NW}(\boldsymbol{\mu}_{j,u}^k, \boldsymbol{\Lambda}_{j,u}^k; \boldsymbol{\delta}_{j,u}^k, \kappa_{j,u}^k, \mathbf{W}_{j,u}^k, v_{j,u}^k)\Big] \quad \forall j \in \{1, 2, ..., N\} \quad (3.7)$$

Further, $\boldsymbol{\Lambda}_{j,u}^k = (\boldsymbol{\Sigma}_{j,u}^k)^{-1}$. The prior in Eq (3.6) can be understood as having the following components

- **Transition Distribution** : Each column of the $N \times N$ transition matrix specifies the probability of making a transition from that column index to another state given by the row index. We define a Dirichlet distribution as a prior over each column of the transition matrix. Hence, $\prod_{i=1}^{N} Dir(\theta_i^k | \boldsymbol{\alpha}_i^k)$ is the prior over Θ^k.
- **Emission Distribution** : $Dir(\mathbf{w}_j^k; \boldsymbol{\beta}_j^k) \prod_{u=1}^{M} \mathcal{NW}(\boldsymbol{\mu}_{j,u}^k, \boldsymbol{\Lambda}_{j,u}^k; \boldsymbol{\delta}_{j,u}^k, \kappa_{j,u}^k, \mathbf{W}_{j,u}^k, v_{j,u}^k)$ defines a prior over a mixture of Gaussians for hidden state j with M components where the Dirichlet distribution is the prior over the mixture weights and the Normal-Wishart distribution is the prior over the mean and precision matrix of the mixture components. We take a product over j to obtain a prior over all emission distributions.

The posterior distribution (Eq (3.7)) after each observation is a mixture of products of distributions where each component has the same form as the prior distribution since $\Pr(X_t^k | Y_t^k = j)$ is a mixture of Gaussians. Therefore, the number of terms in the posterior increases exponentially if we perform exact Bayesian learning. To circumvent this, we use BMM for Gaussian Mixture Models as described in Section 3.1.

The update equation at each time step for a source domain k is

$$Pr\Big(\Theta, \Phi, Y_t^k = j | X_t^k, Y_{t-1}^k = i\Big) \propto \overbrace{\Pr(X_t^k | Y_t^k = j)}^{\text{Emission distribution}} \overbrace{\Pr(Y_t^k = j | Y_{t-1}^k = i)}^{\text{Transition Probability}} \overbrace{\Pr(\Theta^k, \Phi^k, Y_{t-1}^k = i | X_{1:t-1}^k)}^{\text{Prior for } t-1}$$
$$\forall j \in \{1, 2, ..., N\}$$

The posterior after inserting all the relevant terms can be written as -

$$Pr\Big(\Theta, \Phi, Y_t^k = j | X_t^k, Y_{t-1}^k = i\Big) \propto \sum_{m=1}^{M} w_{j,m}^k \mathcal{N}(X_t^k | \boldsymbol{\mu}_{j,m}^k, \Sigma_{j,m}^k) \theta_{ji}^k \Big[\prod_{i=1}^{N} Dir(\theta_i^k | \boldsymbol{\alpha}_i^k)\Big]$$
$$\Big[\prod_{j=1}^{N} Dir(\mathbf{w}_j^k; \boldsymbol{\beta}_j^k) \prod_{u=1}^{M} \mathcal{NW}(\boldsymbol{\mu}_{j,u}^k, \boldsymbol{\Lambda}_{j,u}^k; \boldsymbol{\delta}_{j,u}^k, \kappa_{j,u}^k, \mathbf{W}_{j,u}^k, v_{j,u}^k)\Big] \quad \forall j \in \{1, 2, ..., N\}$$

We can re-write this as

$$Pr\Big(\Theta, \Phi, Y_t^k = j | X_t^k, Y_{t-1}^k = i\Big) = \frac{1}{Z} \sum_{m=1}^{M} \prod_{u \neq i}^{N} \prod_{u \neq m}^{M} \prod_{i \neq j}^{N} C(i, j, k, m) \Big[Dir(\theta_i^k | \hat{\boldsymbol{\alpha}}_i^k) Dir(\theta_u^k | \boldsymbol{\alpha}_u^k)\Big]$$
$$\Big[Dir(\mathbf{w}_j^k; \hat{\boldsymbol{\beta}}_j^k) Dir(\mathbf{w}_i^k; \boldsymbol{\beta}_i^k)\Big] \Big[\mathcal{NW}(\boldsymbol{\mu}_{j,m}^k, \boldsymbol{\Lambda}_{j,m}^k; \hat{\boldsymbol{\delta}}_m^k, \hat{\kappa}_{j,m}^k, \hat{\mathbf{W}}_{j,m}^k, \hat{v}_{j,m}^k) \mathcal{NW}(\boldsymbol{\mu}_{j,u}^k, \boldsymbol{\Lambda}_{j,u}^k; \boldsymbol{\delta}_{j,u}^k, \kappa_{j,u}^k, \mathbf{W}_{j,u}^k, v_{j,u}^k)\Big] \tag{3.8}$$

where $Z = \sum_{i,j,k,m} C(i, j, k, m)$ is the normalization constant. Eq (3.8) is a mixture of product of distributions where each component belongs to the same family as the prior distribution. The set of sufficient moments in this case would be

$$S = \Big\{\theta_i^k, (\theta_i^k)^2, \mathbf{w}_j^k, (\mathbf{w}_j^k)^2, \mu_{j,m}^k, \mu_{j,m}^k (\mu_{j,m}^k)^T, \boldsymbol{\Lambda}_{j,m}^k, \boldsymbol{\Lambda}_{j,m}^k (\boldsymbol{\Lambda}_{j,m}^k)^T \mid \forall m \in [M]\Big\}$$

Thus, the parameters of the approximate posterior can be evaluated following similar routine as discussed in Section 3.1. The computational complexity for updating the parameters in the source domain learning step for each iteration is $\mathcal{O}(M^2N^2)$ for each scalar parameters and is $\mathcal{O}(M^2N^2d^3)$ for the parameters of the distribution over the precision matrix because that involves a matrix multiplication step where M is the number of components in the mixture model for emission distributions, N is the number of hidden states and d is the number of features in the data.

3.2.3 Target Domain - Learning and Prediction

We next want to predict the hidden states for a target individual as we observe a sequence of observations in a streaming manner. In the previous step, we learned the transition and emission distributions individually for K different sources. The transition and emission distributions learned from the individual sources form a basis for the transition and emission distributions of the target domain. Specifically, let the transition distribution for

the k^{th} source be denoted by $\mathcal{G}(\Theta^k)$ and emission distribution be denoted by $\mathcal{F}(\Phi_j^k)$ for a certain hidden state j. Then, the transition and emission distributions for the target domain is a weighted combination given by

$$\Pr(Y_t = j|Y_{t-1} = i) = \sum_{m=1}^{K} \lambda_m \Pr(Y_t^m = j|Y_{t-1}^m = i) = \sum_{m=1}^{K} \lambda_m \mathcal{G}(\Theta_{ji}^m) \tag{3.9}$$

$$\Pr(X_t|Y_t = j) = \sum_{k=1}^{K} \pi_k \Pr(X_t^k|Y_t^k = j) = \sum_{k=1}^{K} \pi_k \mathcal{F}(\Phi_j^k) \tag{3.10}$$

where $\lambda_m > 0, \forall m \in [K]$, $\pi_k > 0 \ \forall k \in [K]$, $\sum_k \pi_k = 1$ and $\sum_m \lambda_m = 1$. To learn a model over the target domain, we need to compute the basis weights $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, ..., \lambda_K)$ and $\boldsymbol{\pi} = (\pi_1, \pi_2, ..., \pi_K)$. We estimate $(\boldsymbol{\lambda}, \boldsymbol{\pi})$ in an unsupervised manner using BMM. We define a Dirichlet prior over $\boldsymbol{\lambda}$ and $\boldsymbol{\pi}$, i.e. $\Pr(\boldsymbol{\lambda}, \boldsymbol{\pi}) = Dir(\boldsymbol{\lambda}; \boldsymbol{\gamma}) Dir(\boldsymbol{\pi}; \boldsymbol{\nu})$. The posterior after observing a new data point is

$$\Pr\left(\boldsymbol{\lambda}, \boldsymbol{\pi}, Y_t = j|X_t\right) \propto \Pr(X_t|Y_t = j) \sum_{i=1}^{N} \Pr(Y_t = j|Y_{t-1} = i) \Pr(\boldsymbol{\lambda}, \boldsymbol{\pi}, Y_{t-1} = i) \tag{3.11}$$

$$\propto \sum_{k=1}^{K} \pi_k \mathcal{F}(\Phi_j^k) \sum_{i=1}^{N} \sum_{m=1}^{K} \lambda_m \mathcal{G}(\Theta_{ji}^m) Dir(\boldsymbol{\lambda}; \boldsymbol{\gamma}) Dir(\boldsymbol{\pi}; \boldsymbol{\nu}) \tag{3.12}$$

$$\propto \sum_{k,m}^{K} \sum_{i=1}^{N} C(i, j, k, m) \ Dir(\boldsymbol{\pi}; \hat{\boldsymbol{\nu}}) Dir(\boldsymbol{\lambda}; \hat{\boldsymbol{\gamma}}) \tag{3.13}$$

where $\mathcal{F}(\Phi_j^k)\mathcal{G}(\Theta_{ji}^m)$ are known from the source domains, $\pi_k Dir(\boldsymbol{\pi}; \boldsymbol{\nu}) = a_k Dir(\boldsymbol{\pi}; \hat{\boldsymbol{\nu}})$, $\lambda_m Dir(\boldsymbol{\lambda}; \boldsymbol{\gamma}) = b_m Dir(\boldsymbol{\lambda}; \hat{\boldsymbol{\gamma}})$ and $C(i, j, k, m) = a_k b_m \mathcal{F}(\Phi_j^k)\mathcal{G}(\Theta_{ji}^m)$.

The prior over the weights is

$$\Pr(\boldsymbol{\lambda}, \boldsymbol{\pi}) = Dir(\boldsymbol{\lambda}; \boldsymbol{\gamma}) Dir(\boldsymbol{\pi}; \boldsymbol{\nu})$$

where $\boldsymbol{\gamma}$ and $\boldsymbol{\nu}$ are the hyper-parameters for the Dirichlet distribution. The posterior after

each observation is

$$Pr\Big(\boldsymbol{\lambda},\boldsymbol{\pi}, Y_t = j|X_t\Big) \propto \Pr(X_t|Y_t = j)\sum_{i=1}^{N}\Pr(Y_t = j|Y_{t-1} = i)\Pr(\boldsymbol{\lambda},\boldsymbol{\pi}, Y_{t-1}) \tag{3.14}$$

$$\propto \sum_{k=1}^{K}\pi_k\sum_{u=1}^{M}\mathcal{N}(\mu_{j,u}^k,\Sigma_{j,u}^k)\sum_{i=1}^{N}\sum_{m=1}^{K}\lambda_m\theta_{ij}^m Dir(\boldsymbol{\lambda};\boldsymbol{\gamma})Dir(\boldsymbol{\pi};\boldsymbol{\nu}) \tag{3.15}$$

$$\propto \sum_{k,m}^{K}\sum_{i=1}^{N}\underbrace{\pi_k Dir(\boldsymbol{\pi};\boldsymbol{\nu})}_{\text{combines}}\underbrace{\lambda_m Dir(\boldsymbol{\lambda};\boldsymbol{\gamma})}_{\text{combines}}\underbrace{\sum_{u=1}^{M}\mathcal{N}(\mu_{j,u}^k,\Sigma_{j,u}^k)\theta_{ij}^m}_{\text{known}} \tag{3.16}$$

$$= \frac{1}{Z}\sum_{k,m}^{K}\sum_{i=1}^{N}C(j,k,m)\ Dir(\boldsymbol{\pi};\hat{\boldsymbol{\nu}})Dir(\boldsymbol{\lambda};\hat{\boldsymbol{\gamma}}) \tag{3.17}$$

where $Z = \sum_{i,j,k,m} C(i,j,k,m)$ is the normalization constant. The exact computation leading to Equation (3.17) is as follows:

$$\lambda_m Dir(\boldsymbol{\lambda},\boldsymbol{\gamma}) = \frac{\gamma_m}{\sum_i \gamma_i}Dir(\boldsymbol{\lambda};\hat{\boldsymbol{\gamma}}) \tag{3.18}$$

where

$$\hat{\gamma}_i = \begin{cases}\gamma_i & \text{if } i \neq m\\ \gamma_i + 1 & \text{if } i = m\end{cases}$$

Therefore *C(i,j,k,m)* in Eq (3.13) is

$$C(i,j,k,m) = \left(\frac{\gamma_m}{\sum_i \gamma_i}\right)\left(\frac{\pi_k}{\sum_i \pi_i}\right)\sum_{u=1}^{M}\mathcal{N}(\mu_{j,u}^k,\Sigma_{j,u}^k)\theta_{ij}^m \tag{3.19}$$

For the moment matching step, the set of sufficient moments is given by

$$S = \{\lambda_i, \lambda_i^2, \pi_i, \pi_i^2 \mid \forall i \in \{1,2,..,K\}\}$$

$$\mathbb{E}[\lambda_n] = \frac{1}{Z}\sum_{k,m}^{K}\sum_{i=1}^{N}\int \lambda_n C(i,j,k,m) Dir(\boldsymbol{\pi};\hat{\boldsymbol{\nu}}) Dir(\boldsymbol{\lambda};\hat{\boldsymbol{\gamma}}) d(\boldsymbol{\lambda}) d(\boldsymbol{\pi}) \tag{3.20}$$

$$= \frac{1}{Z}\sum_{k,m}^{K}\sum_{i=1}^{N}\int \lambda_n C(i,j,k,m) Dir(\boldsymbol{\lambda};\hat{\boldsymbol{\gamma}}) d(\boldsymbol{\lambda}) \tag{3.21}$$

$$= \frac{1}{Z}\sum_{k,m}^{K}\sum_{i=1}^{N}\left(\frac{\hat{\lambda}_n}{\sum_u \hat{\lambda}_u}\right) C(i,j,k,m) \tag{3.22}$$

Similarly, the second moment can be evaluated as

$$\mathbb{E}[\lambda_n^2] = \frac{1}{Z}\sum_{k,m}^{K}\sum_{i=1}^{N}\int \lambda_n^2 C(i,j,k,m) Dir(\boldsymbol{\pi};\hat{\boldsymbol{\nu}}) Dir(\boldsymbol{\lambda};\hat{\boldsymbol{\gamma}}) d(\boldsymbol{\lambda}) d(\boldsymbol{\pi}) \tag{3.23}$$

$$= \frac{1}{Z}\sum_{k,m}^{K}\sum_{i=1}^{N}\left(\frac{\hat{\lambda}_n(\hat{\lambda}_n+1)}{(\sum_u \hat{\lambda}_u)(1+\sum_u \hat{\lambda}_u)}\right) C(i,j,k,m) \tag{3.24}$$

Thus, we have approximated the posterior in Eq (3.13) by projecting it onto a tractable family of distributions with the same set of sufficient moments as the posterior using the Bayesian Moment Matching approach. The estimate of $(\boldsymbol{\lambda}, \boldsymbol{\pi})$ is the expected value of the final posterior. which completes the learning stage for the target domain and we proceed to making predictions based on the observations of the target domain.

Predictions can be made in two different manners

- **Online** - initialize the prior over $\boldsymbol{\lambda}$ and $\boldsymbol{\pi}$ to be uniform. As each new data point is observed in a sequence, a prediction is made based on the mean of the current posterior over $\boldsymbol{\lambda}$ and $\boldsymbol{\pi}$ and subsequently the posterior is updated based on Eq (3.13).
- **Offline** - compute the posterior of $\boldsymbol{\lambda}$ and $\boldsymbol{\pi}$ based on Eq (3.13) by using the entire sequence of observations of the target individual. Once, the posterior is computed, predict the hidden states for each observation in the sequence based on the mean estimates of the posterior.

The target domain step involved two routines :

- **Learning step** - In this step, the hyper-parameters $(\boldsymbol{\gamma}, \boldsymbol{\nu})$ over the weights $(\boldsymbol{\lambda}, \boldsymbol{\pi})$ are updated. The main computation in this step is the calculation of the set of sufficient moments from the updated Bayesian posterior given in Eq. (3.13). Hence, the computational complexity of the update step in the target domain for each observation is $\mathcal{O}(K^2N^2)$ where K is the number of source domains and N is the number of hidden states.

- **Prediction step** - In the prediction step, a hidden label is assigned to the observation based on the model obtained from the update step. The main computation is calculation of the likelihood of each hidden state for the observation. The computational complexity of the prediction step is hence $\mathcal{O}(MKN)$ where M is the number of components in the mixture model, K is the total number of source domains and N is the number of hidden states.

In Fig. 3.1, we show the schematic for the proposed online transfer learning algorithm. The figure shows the learning phase for each source domain where the emission and transition distributions are learned using Bayesian Moment Matching technique and Algorithm 2 gives the complete algorithm for transfer learning by Bayesian Moment Matching.

3.2.4 Experiments and Results

This section describes experiments on three tasks from different domains - activity recognition, sleep cycle prediction among healthy individuals and patients suffering from Parkinson's disease and packet flow prediction in telecommunication networks.

Experimental Setup

For each task, we compare our online transfer learning algorithm to EM (trained by maximum likelihood) and a baseline algorithm (that uses Bayesian moment matching) that both learn a single HMM with mixtures of Gaussians as emissions by treating the population as homogeneous. Furthermore, we conduct experiments using recurrent neural networks (RNNs) due to their popularity in sequence learning.

The baseline algorithm uses Bayesian Moment Matching to learn the parameters of the HMM. Concretely, we have data collected from several individuals (or sources) in a population for each task. For transfer learning, we train an HMM with mixture of Gaussian emission distributions for each source (or individual) except the target individual. For

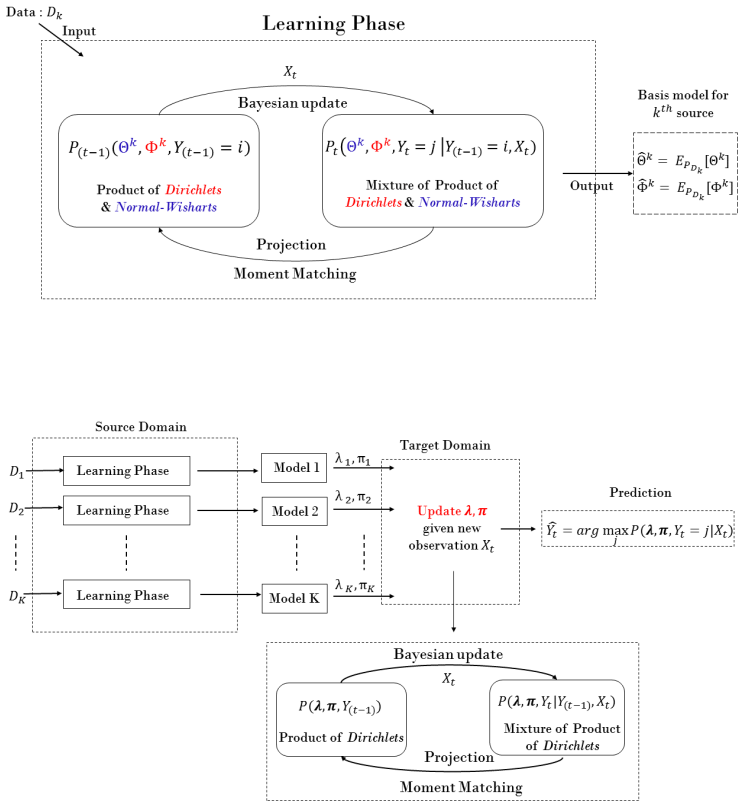


Figure 3.1: Transfer Learning architecture

the target individual, we estimate a posterior over the basis weights in an online and unsupervised fashion and make online predictions about the hidden states. We compare the performance of our transfer learning algorithm against the EM and baseline algorithms that treat the population as homogeneous, i.e., we train an HMM by combining the data from all the sources except the target individual. Then, using this model, we make online predictions about the hidden states of the target individual.

We report the results based on leave-one-out cross validation where the data of a different individual is left out in each round. For each task, we treat every individual as a target individual once. For a fair comparison, the HMM model learned for both the baseline algorithm and the EM algorithm has the same number of components as the HMM model learned by the online transfer learning algorithm.

Regarding RNNs, we used architectures with as many input nodes as the number of

Algorithm 2 Online Transfer Learning by Bayesian Moment Matching

Input (Learning): labeled sequence data from multiple domains (individuals)
Input (Prediction): unlabeled sequence data from individuals
Output: labels for hidden states

Source Domain - learning transition and emission distribution

Input: labeled sequence data from K domains
specify # of hidden states : nClass
specify # of components in GMM : nComponents
procedure LEARNSOURCEHMM(data, nClass, nComponents)
    for k $= 1 :$ K **do**
        Let $f(\boldsymbol{\Theta}, \boldsymbol{\Phi})$ be a family of probability distributions with parameters $\boldsymbol{\gamma}$
        Initialize a prior $P_0^k(\boldsymbol{\Theta}, \boldsymbol{\Phi})$ from f over transition and emission parameters respectively
        for n $= 1 :$ D_k **do** ▷ D_k : size of data for k^{th} source domain
            Compute $P_n(\boldsymbol{\Theta}, \boldsymbol{\Phi})$ from $P_{n-1}(\boldsymbol{\Theta}, \boldsymbol{\Phi})$ using Eq. 3.7
            Using BMM approximate P_n with $\tilde{P}_n(\boldsymbol{\Theta}, \boldsymbol{\Phi}) = f(\boldsymbol{\Theta}, \boldsymbol{\Phi}|\boldsymbol{\gamma})$
        Return : $\hat{\boldsymbol{\Theta}} = \mathbb{E}_{\boldsymbol{\Theta}}[\tilde{P}_n(\boldsymbol{\Theta}, \boldsymbol{\Phi})]$
        Return : $\hat{\boldsymbol{\Phi}} = \mathbb{E}_{\boldsymbol{\Phi}}[\tilde{P}_n(\boldsymbol{\Theta}, \boldsymbol{\Phi})]$
Return : emission and transition distributions for each source

Target Domain - learning basis weights for each source domain & prediction

Input: unlabeled sequence data
procedure PREDICTTARGETDOMAIN(data, sourceDistributions)
    Let $g(\boldsymbol{\lambda}, \boldsymbol{\pi}) = Dir(\boldsymbol{\lambda}; \boldsymbol{\gamma}) Dir(\boldsymbol{\pi}; \boldsymbol{\nu})$ be a family of probability distributions
    Initialize a prior $P_0(\boldsymbol{\lambda}, \boldsymbol{\pi})$ from g with equal weights to each source distribution
    for n $= 1 :$ D **do** ▷ D : size of data for target domain
        Compute $P_n(\boldsymbol{\Theta}, \boldsymbol{\Phi})$ from $P_{n-1}(\boldsymbol{\Theta}, \boldsymbol{\Phi})$ using Eq. 3.13
        Using BMM approximate P_n with $\tilde{P}_n(\boldsymbol{\lambda}, \boldsymbol{\pi}) = g(\boldsymbol{\lambda}, \boldsymbol{\pi})$
        Predict : $\hat{Y}_n = \text{argmax}_j Pr\left(\boldsymbol{\lambda}, \boldsymbol{\pi}, Y_n = j | \mathbf{X}_n\right)$ using Eq (3.13)
        Return : $\hat{\boldsymbol{\lambda}} = \mathbb{E}_{\boldsymbol{\lambda}}[\tilde{P}_n(\boldsymbol{\lambda}, \boldsymbol{\pi})]$
        Return : $\hat{\boldsymbol{\pi}} = \mathbb{E}_{\boldsymbol{\pi}}[\tilde{P}_n(\boldsymbol{\lambda}, \boldsymbol{\pi})]$
        Return : prediction $\hat{Y}_n$

attributes, one hidden layer consisting of long short term memory (LSTM) units [Hochreiter and Schmidhuber, 1997] and one softmax output layer with as many nodes as the number

of classes. We use the categorical cross-entropy loss as the cost function. We select LSTM units instead of sigmoid or hyperbolic tangent units due to their popularity and success in sequence learning [Sutskever et al., 2014].

We perform grid search to select the best hyper-parameters for each setting. For the training method, we either use Nesterov's accelerated gradient descent [Nesterov, 1983, Sutskever et al., 2013] with learning rates [0.001,0.01,0.1,0.2] and momentum values [0,0.2,0.4,0.6,0.8,0.9], or rmsprop [Tieleman and Hinton, 2012] having $\varepsilon = 10^{-4}$ and decay factor 0.9 (standard values) with learning rates [0.00005,0.0001,0.0002,0.001] and momentum values [0,0.2,0.4,0.6,0.8,0.9]. The weight decay takes values from [0.001,0.01,0.1], whereas the number of LSTM units in the hidden layer takes the possible values [2,4,6,9,12].

We experimented with various architectures before we ended up with the aforementioned values; in particular, architectures with a single hidden layer consistently performed better than multiple layers, possibly because our datasets are not very complex. We train the network by backpropagation through time (bptt) truncated to 20 time steps [Williams and Peng, 1990]. The RNNs are trained for a maximum number of 150 epochs, or until convergence is reached. Our implementation is based on the Theano library [Theano Development Team, 2016] in Python.

For each task, we run experiments 10 times with each individual taken as target and the rest acting as source domains for training. We report the average percentage accuracy and use the Wilcoxon signed rank test [Wilcoxon, 1950] to compute a p-value and report statistical significance when the p-value is less than 0.05. In the following sections, we discuss the results for each task in detail.

Activity Recognition

As part of an on-going study to promote physical activity, we collected smartphone data with 19 participants and tested our transfer learning algorithm to recognize 5 different kinds of activities: sitting, standing, walking, running and in-a-moving-vehicle. While APIs already exist to automatically recognize walking, running and in-a-moving-vehicle by Android and Apple smartphones, sitting and standing are not available in the standard APIs. Furthermore, our long term goal is to obtain robust recognition algorithms for older adults and individuals with perturbed gait (e.g., due to a stroke). Labeled data was obtained by instructing the 19 participants to walk at varying speeds for 4 min, run for 2 min, stand for 2 min, sit for 2 min and ride a moving vehicle to a destination of their choice. The data collected was segmented in epocs of 1 second where 48 features (means and standard deviations of the 3D accelerometry in each epoch) were computed by the

smartphone. The online transfer learning algorithm learned an HMM over 18 individuals which acted as basis models for prediction on the 19^{th} individual. In this manner, we ran experiments for each individual 10 times to get a statistical measure of the results.

Table 3.3: Average percentage accuracy of prediction for activity recognition on 19 different individuals. The best results among the Baseline, the EM algorithm, RNN and Transfer Learning algorithm are highlighted in bold font. ↑(or ↓) indicates that Transfer Learning has significantly better (or worse) accuracy than the the best algorithm among the baseline, EM and RNN under the Wilcoxon signed rank test with p-value < 0.05.

Target Domain	Baseline	EM	RNN	Transfer Learning
Person 1	**91.29**	83.57	71.15	88.36 ↓
Person 2	81.37	79.87	79.58	**87.65**↑
Person 3	74.68	75.91	69.56	**93.15**↑
Person 4	73.39	68.29	74.25	**84.70**↑
Person 5	95.94	89.59	95.36	**99.75**↑
Person 6	73.98	69.77	61.71	**96.43**↑
Person 7	57.62	55.15	69.22	**70.75**↑
Person 8	91.72	86.05	74.49	**97.80**↑
Person 9	81.19	78.88	78.72	**88.75**↑
Person 10	**99.12**	93.60	92.00	97.35↓
Person 11	76.59	74.67	84.75	**88.75**↑
Person 12	55.36	59.71	53.63	**95.05**↑
Person 13	79.66	73.46	65.54	**97.60**↑
Person 14	92.06	89.11	63.59	**93.12**↑
Person 15	79.25	72.24	91.08	**94.20**↑
Person 16	**84.08**	79.23	74.74	83.51↓
Person 17	93.95	91.03	81.25	**97.60**↑
Person 18	82.84	74.88	79.45	**87.20**↑
Person 19	**95.97**	89.06	95.88	95.06↓

Table (3.3) compares the average percentage accuracy of prediction for activity recognition with 19 different individuals. It demonstrates that the transfer learning algorithm performed better than the baseline on 15 individuals and in other cases its accuracy was close to the baseline. Furthermore, it is also worth noting that in most cases, the confusion in the algorithm's prediction was between the following pairs of classes: *In a Moving Vehicle—Standing* and *In a Moving Vehicle—Sitting*. This is expected because in most cases the person was either standing/sitting in a bus or sitting in a car. Table (3.3) also

demonstrates the superior performance of online transfer learning algorithm as compared to the EM algorithm. Finally, note the poor performance of RNNs despite the fact that we fine-tuned the architecture to get the best results. RNNs are in theory very expressive. However, they are also notoriously difficult to train and fine-tune due to their non-convexity and vanishing/exploding gradient issues that arise in backpropagation through time. Indeed, in several cases they even underperform all other methods.

Sleep Stage Classification

Sleep disruption can lead to various health issues. Understanding and analyzing sleep patterns, therefore, has great potential to significantly improve the quality of life for both patients and healthy individuals. In both clinical and research settings, the standard tool for quantifying sleep architecture and physiology is polysomnography (PSG), which is the measurement of electroencephalography (EEG), electrooculography (EOG), electromyography (EMG), electrocardiography (ECG), and respiratory function of an individual during sleep. The analysis of sleep architecture is of relevance for the diagnosis of several neurological disorders, e.g., Parkinson's disease [Peeraully et al., 2012], because neurological anomalies often also reflect in variations of a patient's sleep patterns.

Typically, PSG data is divided into 30-second *epochs* and classified into 5 stages of sleep — wake (W), rapid eye movement sleep (REM) or one of 3 non-REM sleep stages (N1, N2, and N3) — based on the visual identification of specific signal features on the EEG, EOG, and EMG channels. Epochs that cannot be distinctly sorted into one of the 5 stages are labeled as *Unknown*. While it is a valuable clinical and research tool, visual classification of EEG data remains time consuming, requiring up to 2 hours for a highly trained technologist to classify all the epochs within a typical 7-hour PSG recording. Beyond that, inter-scorer agreement rates remain low around 80 [Rosenberg and Van Hout, 2013]. High annotation costs and low inter-scorer agreement rates have motivated efforts to develop fully automated approaches for sleep stage classification [Anderer et al., 2005, Jensen et al., 2010, Mal, 2013, Punjabi et al., 2015]. However, many of these methods result in generic cross-patient classifiers that fail to reach levels of accuracy and reliability high enough to be adopted in real-world medical settings.

The polysomnograms (PSGs) we used for our evaluation were obtained at a clinical neurophysiology laboratory in Toronto (name anonymized) according to the American Academy of Sleep Medicine guidelines using a Grael HD PSG amplifier (Compumedics, Victoria, Australia). We selected recordings from 142 patients obtained between 2009 and 2015. Out of these 142 recordings, 91 were from healthy subjects and 51 were from patients with Parkinson's disease.

Each recording was manually scored by a single registered PSG technologist. Recordings were first segmented into fixed-sized windows of 30 second epochs. To reduce complexity and processing time in the feature extraction and manual labeling step, we only retained EEG channel C4-A1, which is deemed especially important for sleep stage classification [Sil, 2007]. Channel selection and segmentation resulted in a ground truth data set where each instance was represented by a single-channel time series of 7680 floating point numbers corresponding to 30 seconds of C4-A1, sampled at 256 Hz. A vector of 26 scalar features was extracted from each epoch. [Bao et al., 2011] and [Motamedi-Fakhr et al., 2014] give a detailed listing and explanation of all 26 features.

The online transfer learning algorithm learned an HMM over 50 individuals chosen at random which acted as basis models for prediction on the target individual. We did not use all 140 individuals for the basis models because it resulted in sources getting sparse weights diluting the effect of heterogeneity. We completed the experiments for each individual 10 times in this manner to get a statistical measure of the results.

Fig. (3.2) shows the scatter plots of accuracy for our online transfer learning technique and the three baseline algorithms - BMM, EM (maximum likelihood) and RNNs - which treat the data as homogeneous for the sleep stage classification dataset. For each plot, a point above the dotted line indicates higher accuracy of online transfer learning technique as compared to the corresponding baseline algorithm for the target patient. The plots show consistent superior performance of our online transfer learning technique as compared to both baseline algorithms - BMM and EM for all target patients. The online transfer learning technique also performs better on a majority of patients (102 out of 142) as compared to an optimized RNN.

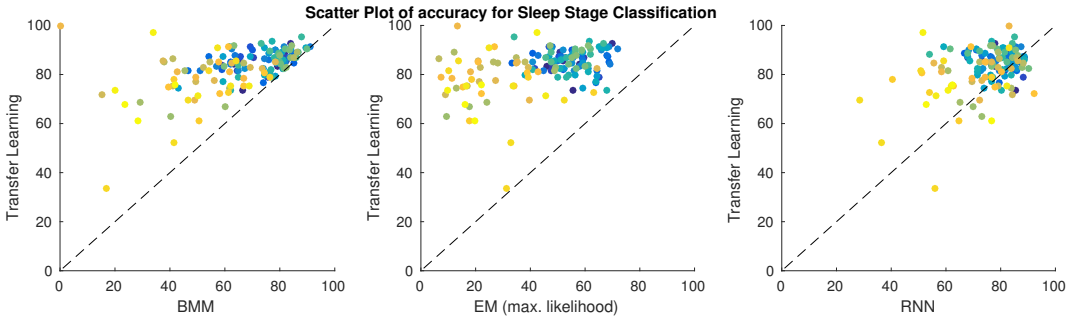


Figure 3.2: Performance comparison of online transfer learning algorithm with three different baseline algorithms - BMM, EM (max. likelihood) and RNNs on Sleep Stage Classification data using scatter plots of accuracy.

All the results are statistically significant under the Wilcoxon signed rank test with

p-value < 0.05.

Flow Direction Prediction

Accurate prediction of future traffic plays an important role in proactive network control. Proactive network control means that if we know the future traffic (including directions and traffic volume), then we have more time to find a better policy for the network routing, priority scheduling as well as rate control in order to maximize network throughput while minimizing transmission delay, packet loss rate, etc.

Better understanding the behavior of TCP connections in certain applications can provide important input to automatic application type detection, especially in those scenarios where network traffic is encrypted and DPI (Deep Packet Inspection) is nearly impossible. Different applications can be distinguished by the distinct behavior of their TCP connections, which are well described by the corresponding HMMs.

We performed our experiments with a publicly available dataset of real traffic from academic buildings. The dataset consists of packet traces with TCP flows. For our experiments, we only consider three packet sizes and flow size as the features. The hidden labels are the source of generation of the packet, i.e., *Server* or *Client*. We divided the dataset into 9 domains with each domain consisting of a number of observation sequences. For the online transfer learning algorithm, we learned an HMM for each of 8 sources that acted as basis models for prediction on the 9^{th} source. We compared the performance of the online transfer learning algorithm with EM and the baseline algorithm which treat the data as homogeneous. Table 3.4 reports the average (of 10 experimental runs) percentage accuracy for each source. The online transfer learning algorithm performs better than both the baseline and the EM algorithm. The results are statistically significant under the Wilcoxon signed rank test with p-value < 0.05. Furthermore, we compare our method to RNNs. It turns out that RNNs perform well for the task of traffic direction prediction unlike for instance the activity recognition dataset. The better performance this time may be due to the simpler structure of the data that consists of a single attribute and a binary class. This is in sharp contrast to the activity recognition dataset whose instances contain 48 attributes and can belong to 5 classes, and is thus harder to train.

Table 3.4: Average percentage accuracy of prediction for flow direction prediction for 9 different domains. The best results among the Baseline, the EM algorithm, RNN and the Transfer Learning algorithm are highlighted in bold font. ↑(or ↓) indicates that transfer learning has significantly better (or worse) accuracy than the best technique among the baseline algorithm, EM and RNN under Wilcoxon signed rank test with pvalue < 0.05.

Target Domain	Baseline	EM	RNN	Transfer Learning
Source 1	72.00	54.90	**80.00**	71.02 ↓
Source 2	85.33	**89.10**	65.30	86.50↓
Source 3	80.33	81.90	**86.50**	83.33↑
Source 4	86.50	75.80	86.60	**87.17**↑
Source 5	**87.33**	82.80	81.70	86.00↓
Source 6	93.33	78.20	88.90	**93.50**↑
Source 7	95.17	90.70	93.50	**95.33**↑
Source 8	89.83	91.14	91.00	**91.63**↑
Source 9	76.67	75.68	**81.98**	78.83↑

3.3 Summary

In this chapter, we investigated online algorithms to learn the parameters of Gaussian Mixture models and their applications to sequential data modeling. We proposed an online Bayesian Moment Matching algorithm for parameter learning and demonstrated its use in a distributed manner for offline settings. We showed through empirical analysis on several real-world datasets that the online Bayesian Moment Matching algorithm outperforms online EM and online VB. We also demonstrated that distributing the algorithm over several machines results in faster running times without significantly compromising accuracy, which is particularly advantageous when running time is a major bottleneck.

Next, we considered domains in which data is produced by a population of individuals that exhibit a certain degree of variability. Traditionally, machine learning techniques ignore this variability and train a single model under the assumption that the population is homogeneous. We described the first online transfer learning technique (to our knowledge) that incrementally determines which source models best explain a streaming sequence of observations while predicting the corresponding hidden states by adapting the online Bayesian moment matching algorithm originally developed for mixture models to hidden Markov models. We confirmed the effectiveness of the approach on three real-world applications: activity recognition, sleep stage recognition and flow direction prediction.

Chapter 4

Neural Density Estimation

We have so far focused on density estimation using finite mixture models. In this chapter, we will focus on another framework for parametric density estimation called neural density estimation by constructing flexible mappings that can transform a simple probability density into any desired target density by using the notion of *triangular maps*. Triangular map is a recent construct in probability theory that allows one to transform any source probability density function to any target density function. Based on triangular maps, we propose a general framework for high-dimensional density estimation, by specifying one-dimensional transformations (or equivalently conditional densities) and appropriate conditioner networks. We will show in detail that this framework (a) reveals the commonalities and differences of existing autoregressive and flow based methods, (b) allows a unified understanding of the limitations and representation power of these recent approaches and, (c) motivates us to uncover a new Sum-of-Squares (SOS) flow that is interpretable, universal, and easy to train. We further demonstrate the benefits (and short-comings) of such transformations through several synthetic experiments on various density geometries. Finally, we will compare our method – SOS flows– to other neural density estimation techniques on a suite of density estimation tasks.

The results in this chapter appeared in [Jaini et al., 2019].

4.1 Introduction

Neural density estimation methods are gaining popularity for the task of multivariate density estimation in machine learning [Kingma et al., 2016, Dinh et al., 2015, Dinh et al.,

2017, Papamakarios et al., 2017, Uria et al., 2016, Huang et al., 2018]. These generative models provide a tractable way to evaluate the exact density, unlike generative adversarial nets [Goodfellow et al., 2014] or variational autoencoders [Kingma and Welling, 2014, Rezende et al., 2014]. Popular methods for neural density estimation are *autoregressive models* [Neal, 1992b, Bengio and Bengio, 1999, Larochelle and Murray, 2011, Uria et al., 2016] and *normalizing flows* [Rezende and Mohamed, 2015, Tabak and Vanden-Eijnden, 2010, Tabak and Turner, 2013]. These models aim to learn an invertible, bijective and increasing transformation $\mathbf{T}$ that pushes forward a (simple) source probability density (or measure, in general) to a target density such that computing the inverse $\mathbf{T}^{-1}$ and the Jacobian $|\mathbf{T}'|$ is *easy*.

In probability theory, it has been rigorously proven that increasing *triangular* maps [Bogachev et al., 2005] are universal, i.e. any source density can be transformed into a target density using an increasing triangular map. Indeed, the Knothe-Rosenblatt transformation [Villani, 2008, Ch.1,] gives a (heuristic) construction of such a map, which is unique up to null sets [Bogachev et al., 2005]. Furthermore, by definition the inverse and the Jacobian of a triangular map can be very efficiently computed through univariate operations. However, for multivariate densities computing the exact Knothe-Rosenblatt transform itself is not possible in practice. Thus, a natural question is: Given a pair of densities, how can we efficiently estimate this unique increasing triangular map?

This chapter is devoted to studying these increasing, bijective, and monotonic triangular maps, in particular how to estimate them in practice. In §4.2, we precisely formulate the density estimation problem and propose a general maximum likelihood framework for estimating densities using triangular maps. We also explore the properties of the triangular map required to push a source density to a target density.

Subsequently, in §4.3, we trace back the origins of the triangular map and connect it to many recent works on generative modelling. We relate our study of increasing, bijective, triangular maps to works on iterative Gaussianization [Chen and Gopinath, 2001, Laparra et al., 2011] and normalizing flows [Tabak and Vanden-Eijnden, 2010, Tabak and Turner, 2013, Rezende and Mohamed, 2015]. We show that a triangular map can be decomposed into compositions of one dimensional transformations or equivalently univariate conditional densities, allowing us to demonstrate that all *autoregressive models* and *normalizing flows* are subsumed in our general density estimation framework. As a by-product, this framework also reveals that *autoregressive models* and *normalizing flows* are in fact equivalent. Using this unified framework, we study the commonalities and differences of the various aforementioned models. Most importantly, this framework allows us to study the universality in a much cleaner and more streamlined way. We present a unified understanding of the limitations and representation power of these approaches, summarized concisely in

Table 4.1 below.

In §4.4, by understanding the pivotal properties of triangular maps and using our proposed framework, we uncover a new neural density estimation procedure called the Sum-of-Squares polynomial flows (SOS flows). We show that SOS flows are akin to higher order approximation of $\mathbf{T}$ depending on the degree of the polynomials used. Subsequently, we show that SOS flows are universal, i.e. given enough model complexity, they can approximate any target density. We further show that (a) SOS flows are a strict generalization of the inverse autoregressive flow (IAF) of [Kingma et al., 2016], (b) they are interpretable; its coefficients directly control the higher order moments of the target density and, (c) SOS flows are easy to train; unlike NAFs [Huang et al., 2018] which require non-negative weights, there are no constraints on the parameters of SOS.

In §4.5, we report our empirical analysis. We performed holistic synthetic experiments to gain intuitive understanding of triangular maps and SOS flows in particular. Additionally, we compare SOS flows to previous neural density estimation methods on real-world datasets where it achieved competitive performance.

We summarize the main contributions in this chapter below:

- We study and propose a rigorous framework for using triangular maps for density estimation
- Using this framework, we study the similarities and differences of existing flow based and autoregressive models
- We provide a unified understanding of the limitations and representational power of these methods
- We propose SOS flows that are universal, interpretable, and easy to train.
- We perform several synthetic and real-world experiments to demonstrate the efficacy of SOS flows.

4.2 Density estimation through triangular map

In this section we set up our main problem, introduce key definitions and notations, and formulate the general approach to estimate density functions using triangular maps.

Let p, q be two probability density[1] functions (w.r.t. the Lebesgue measure) over the source domain $\mathsf{Z} \subseteq \mathbb{R}^d$ and the target domain $\mathsf{X} \subseteq \mathbb{R}^d$, respectively. Our main goal is to find a *deterministic* transformation $\mathbf{T} : \mathsf{Z} \to \mathsf{X}$ such that for all (measurable) set $B \subseteq \mathsf{X}$,

$$\int_B q(\mathbf{x})\mathrm{d}\mathbf{x} \approx \int_{\mathbf{T}^{-1}(B)} p(\mathbf{z})\mathrm{d}\mathbf{z}. \tag{4.1}$$

In particular, when $\mathbf{T}$ is bijective and differentiable [Rudin, 1987], we have the change-of-variable formula $\mathbf{x} = \mathbf{T}(\mathbf{z})$ such that

$$\begin{aligned} q(\mathbf{x}) &= p(\mathbf{z})/|\mathbf{T}'(\mathbf{z})| && (4.2)\\ &= p(\mathbf{T}^{-1}\mathbf{x})/|\mathbf{T}'(\mathbf{T}^{-1}\mathbf{x})| =: \mathbf{T}_{\#}p, && (4.3) \end{aligned}$$

where $|\mathbf{T}'(\mathbf{z})|$ is the (absolute value) of the Jacobian (determinant of the derivative) of $\mathbf{T}$. In other words, by pushing the source random variable $\mathbf{z} \sim p$ through the map $\mathbf{T}$ we can obtain a new random variable $\mathbf{x} \sim q$. This "push-forward" idea has played an important role in optimal transport theory [Villani, 2008] and in recent Monte carlo simulations [Marzouk et al., 2016, Parno and Marzouk, 2018, Peherstorfer and Marzouk, 2018].

Here, our interest is to learn the target density q through the map $\mathbf{T}$. Let $\mathcal{F}$ be a class of mappings and use the KL divergence[2] to measure closeness between densities. We can formulate the density estimation problem as:

$$\min_{\mathbf{T}\in\mathcal{F}} \ \mathsf{KL}(q\|\mathbf{T}_{\#}p) \equiv -\int q(\mathbf{x}) \log \frac{p(\mathbf{T}^{-1}\mathbf{x})}{|\mathbf{T}'(\mathbf{T}^{-1}\mathbf{x})|}\mathrm{d}\mathbf{x}. \tag{4.4}$$

When we only have access to an i.i.d. sample $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \sim q$, we can replace the integral above with empirical averages, which amounts to maximum likelihood estimation:

$$\max_{\mathbf{T}\in\mathcal{F}} \ \frac{1}{n}\sum_{i=1}^{n} \Big[-\log|\mathbf{T}'(\mathbf{T}^{-1}\mathbf{x}_i)| + \log p(\mathbf{T}^{-1}\mathbf{x}_i)\Big]. \tag{4.5}$$

Conveniently, we can choose *any* source density p to facilitate estimation. Typical choices include the standard normal density on $\mathsf{Z} = \mathbb{R}^d$ (with zero mean and identity covariance) and uniform density over the cube $\mathsf{Z} = [0,1]^d$.

Computationally, being able to solve (4.5) efficiently relies on choosing a map $\mathbf{T}$ whose

[1] All of our results can be extended to two probability measures satisfying mild regularity conditions. For simplicity and concreteness we restrict to probability densities here.

[2] Other statistical divergences can be used as well.

- inverse $\mathbf{T}^{-1}$ is "cheap" to compute;
- Jacobian $|\mathbf{T}'|$ is "cheap" to compute.

Fortunately, this is always possible. Following [Bogachev et al., 2005] we call a (vector-valued) mapping $\mathbf{T} : \mathbb{R}^d \to \mathbb{R}^d$ triangular if for all j, its j-th component T_j only depends on the first j variables $x_1, \ldots, x_j$. The name "triangular" is derived from the fact that the derivative of $\mathbf{T}$ is a triangular matrix function[3]. We call $\mathbf{T}$ (strictly) increasing if for all $j \in [d]$, T_j is (strictly) increasing w.r.t. the j-th variable x_j when other variables are fixed.

Theorem 12 ([Bogachev et al., 2005])**.** *For any two densities p and q over $\mathsf{Z} = \mathsf{X} = \mathbb{R}^d$, there exists a unique (up to null sets of p)* increasing *triangular map $\mathbf{T} : \mathsf{Z} \to \mathsf{X}$ so that $q = \mathbf{T}_{\#}p$. The same[4] holds over $\mathsf{Z} = \mathsf{X} = [0,1]^d$.*

Conveniently, to compute the Jacobian of an increasing triangular map we need only multiply d *univariate* partial derivatives $|\mathbf{T}'(\mathbf{x})| = \prod_{j=1}^d \frac{\partial T_j}{\partial x_j}$. Similarly, inverting an increasing triangular map requires inverting d *univariate* functions sequentially, each of which can be efficiently done through say bisection. [Bogachev et al., 2005] further proved that the change-of-variable formula (4.3) holds for any increasing triangular map $\mathbf{T}$ (without any additional assumption but using the right-side derivative).

Thus, triangular mappings form a very appealing function class for us to learn a target density as formulated in (4.4) and (4.5). Indeed, [Moselhy and Marzouk, 2012] already promoted a similar idea for Bayesian posterior inference and [Spantini et al., 2018] related the sparsity of a triangular map with (conditional) independencies of the target density. Moreover, many recent generative models in machine learning are precisely special cases of this approach. Before we discuss these connections, let us give some examples to help understand Theorem 12.

Example 2. *Consider two probability densities p and q on the real line $\mathbb{R}$, with distribution function F and G, respectively. Then, we can define the increasing map $T = G^{-1} \circ F$ such that $q = \mathbf{T}_{\#}p$, where $G^{-1} : [0,1] \to \mathbb{R}$ is the quantile function of q:*

$$G^{-1}(u) := \inf\{t : G(t) \geq u\}. \tag{4.6}$$

[3]The converse is clearly also true if our domain is connected.

[4]More generally on any open or closed subset of $\mathbb{R}^d$ if we interpret the monotonicity of $\mathbf{T}$ appropriately [Alexandrova, 2006].

Indeed, it is well-known that $F(Z) \sim$ uniform *if* $Z \sim p$ *and* $G^{-1}(U) \sim q$ *if* $U \sim$ uniform. *Theorem 12 is a rigorous iteration of this univariate argument by repeatedly conditioning. Note that the increasing property is essential for claiming the uniqueness of* $\mathbf{T}$. *Indeed, for instance, let* p *be standard normal, then both* $\mathbf{T} = id$ *and* $\mathbf{T} = -id$ *push* p *to the same target normal density.*

Example 3 (Pushing uniform to normal)**.** *Let* p *be uniform over* $[0,1]$ *and* $q \sim \mathcal{N}(\mu, \sigma^2)$ *be normal distributed. The unique increasing transformation*

$$T(z) = G^{-1} \circ F = \mu + \sqrt{2}\sigma \cdot \operatorname{erf}^{-1}(2z-1) \tag{4.7}$$

$$= \mu + \sqrt{2}\sigma \cdot \sum_{k=0}^{\infty} \frac{\pi^{k+1/2} c_k}{2k+1} (z - \tfrac{1}{2})^{2k+1}, \tag{4.8}$$

where $\operatorname{erf}(t) = \frac{2}{\sqrt{\pi}} \int_0^t e^{-s^2} \mathrm{d}s$ *is the error function, which was Taylor expanded in the last equality. The coefficients* $c_0 = 1$ *and* $c_k = \sum_{m=0}^{k-1} \frac{c_m c_{k-1-m}}{(m+1)(2m+1)}$. *We observe that the derivative of* T *is an infinite sum of squares of polynomials. In particular, if we truncate at* $k = 0$, *we obtain*

$$T(z) = \mu + \sqrt{2\pi}\sigma(z - \tfrac{1}{2}) + O(z^3). \tag{4.9}$$

Example 4 (Pushing normal to uniform)**.** *Similar as above but we now find a map* S *that pushes* q *to* p*:*

$$S(x) = F^{-1} \circ G = \Phi(\tfrac{x-\mu}{\sigma}) \tag{4.10}$$

$$= \frac{1}{2} + \frac{1}{\sqrt{\pi}} \sum_{k=0}^{\infty} \frac{(-1)^k}{k!(2k+1)} \left(\frac{x-\mu}{\sqrt{2}\sigma} \right)^{2k+1}, \tag{4.11}$$

where Φ *is the cdf of standard normal. As shown by [Medvedev, 2008],* S *must be the inverse of the map* T *in Example 3. We observe that the derivative of* S *is no longer a sum of squares of polynomials, but we prove later that it is approximately so. If we truncate at* $k = 0$, *we obtain*

$$S(x) = \frac{1}{2} + \frac{1}{\sqrt{2\pi}\sigma}(x - \mu) + O(x^3), \tag{4.12}$$

where the leading term is also the inverse of the leading term of T *in* (4.9).

We end this section with two important remarks.

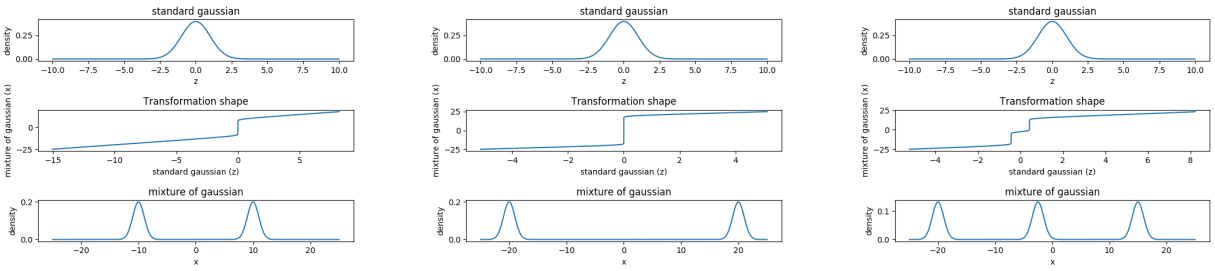


Figure 4.1: Transformation curves from standard Gaussian to mixture of Gaussians.

Remark 1. *So far we have employed the (increasing) triangular map* $\mathbf{T}$ explicitly *to represent our estimate of the target density. This is advantageous since it allows us to easily draw samples from the estimated density, and, if needed, it results in the estimated density formula* (4.3) *immediately. An alternative would be to parameterize the estimated density directly and explicitly, such as in mixture models, probabilistic graphic models and sigmoid belief networks. The two approaches are conceptually* equivalent*: Thanks to Theorem 12, we know choosing a family of triangular maps fixes a family of target densities that we can represent, and conversely, choosing a family of target densities fixes a family of triangular maps that we can* implicitly *learn. The advantage of the former approach is that given a sample from the target density, we can infer the "pre-image" in the source domain while this information is lost in the second approach.*

In Chapters 2 and 3, we studied in detail density estimation via mixture models, particularly Gaussian mixture models. Here, we briefly make a precise connection between mixture models and neural density estimation by characterizing the transformation required to transform a standard normal distribution to mixture of normal distributions in the univariate case. Example 2 gives the analytic form of the increasing transformation T that maps any source density to any target density. Using this analytic form, in Figure 4.1 we show three columns of plots: In the leftmost column, the top plot is the source distribution ($\mathsf{Z} \sim \text{normal}(0, 1)$) which is standard normal. The bottom plot is the target distribution for the random variable X which is a Gaussian mixture model with two components. The means are -10 and 10, the variance is 1 and weights are 0.5 for each component respectively. The middle plot shows the transformation T required to push forward a standard normal distribution to the target. In the second column of plots, we now transform a standard normal distribution to a mixture distribution but with means as -20 and 20, i.e. the components are more separated than the first plot. Finally, in the plots given in the rightmost column, we transform a standard normal distribution to a mixture of three Gaussians with means -20, -5, and 15. The variances are 1 and weights

are $\frac{1}{3}$ respectively.

We analyse the transformation depicted in these plots using the slope $T'(z)$ of T at any point z which is given by:

$$T'(z) = \frac{p(z)}{q \circ T(z)}, \quad \text{where} \quad x = T(z) \tag{4.13}$$

$$= \frac{p \circ F^{-1}(u)}{q \circ G^{-1}(u)}, \quad \text{where} \quad u = F(z) \tag{4.14}$$

i.e. the slope $T'(z)$ is the ratio of probability density quantiles (pdQs) of the source random variable and the target random variable.

We make the following observations: In all three plots, we notice that the transformation admits jumps (close to being vertical) i.e. the slope at these points is large and close to infinity. This is expected since the regions where the target has almost zero mass but the source has finite mass would lead to a slope with such behavior. In the plots, this is the region in between the components where the mass of the target density approaches zero. Furthermore, the larger this area, the longer is the height of this jump (see plots on column one and column two). With densities that have two such areas, the transformation as expected has two jumps (plots on column three). The slope of T on the extremes is a constant and is equal to the standard deviation of the component on that extreme. This is because:

$$\lim_{z \to \pm\infty} T'(z) = \lim_{z \to \pm\infty} \frac{p(z)}{q \circ T(z)} \tag{4.15}$$

As $z \to \infty$, q is approximately equal to the component on the positive extreme of the x-axis. Therefore, $\lim_{z\to\infty} T'(z) = \sigma_+$ where σ_+ is the standard deviation of the component on the positive extreme of the x-axis; similarly, we get $\lim_{z\to-\infty} T'(z) = \sigma_-$ i.e. $T'(z)$ is a constant in almost all the region of zero mass on the left of the component on the negative extreme and on the right of the positive extreme (verified in Figure 4.1). Therefore, any T that transforms a standard normal distribution to a mixture of Gaussians will be approximately piece-wise linear with jumps. The number of linear pieces in this transformation will be equal to the number of components in the mixture. The slopes of these linear pieces will be a function of the standard deviations of the mixture components. Additionally, the height of the jump will be a function of the mixing weights and standard deviation of the mixture components. The summary of these observations lead to the following remark:

Remark 2. *If the target density has disjoint support e.g. mixture of Gaussians (MoGs) with well-separated components, then the resulting transformation will need to admit sharp*

jumps for areas of near zero mass. This follows by analyzing the transformation $T(z) = G^{-1} \circ F$. The slope $T'(z)$ of $T(z)$ is the ratio of the quantile pdfs of the source density and the target density. Therefore, in regions of near zero mass for target density, the transformation will have near infinite slope. We demonstrated this phenomena (see fig. 4.1) specifically for well-separated MoGs and show that a piece-wise linear function transforms a standard Gaussian to MoGs. This also opens the possibility to use the number of jumps of an estimated transformation as the indication of the number of components in the data density.

4.3 Connection to existing works

The results in Section 4.2 suggest using (4.5) with $\mathcal{F}$ being a class of triangular maps for estimating a probability density q. In this section we put this general approach into historical perspective, and connect it to the many recent works on generative modelling. Due to space constraint, we limit our discussion to works that are directly relevant to ours.

Origins of triangular map: [Rosenblatt, 1952], among his contemporary peers, used the triangular map to transform a continuous multivariate distribution into the uniform distribution over the cube. Independently, [Knothe, 1957] devised the triangular map to transform uniform distributions over convex bodies and to prove generalizations of the Brunn-Minkowski inequality. [Talagrand, 1996], unaware of the previous two results and in the process of proving some sharp Gaussian concentration inequality, effectively discovered the triangular map that transforms the Gaussian distribution into any continuous distribution. The work of [Bogachev et al., 2005] rigorously established the existence and uniqueness of the triangular map and systematically studied some of its key properties. [Carlier et al., 2010] showed surprisingly that the triangular map is the limit of solutions to a class of Monge-Kantorovich mass transportation problems under quadratic costs with diminishing weights. None of these pioneering works considered using triangular maps for density estimation.

Iterative Gaussianization and Normalizing Flow: In his seminal work, [Huber, 1985] developed the important notion of non-Gaussianality to explain the projection pursuit algorithm of [Friedman et al., 1984]. Later, [Chen and Gopinath, 2001], based on a heuristic argument, discovered the triangular map approach for density estimation but deemed it impractical because of the seemingly impossible task of estimating too many conditional densities. Instead, [Chen and Gopinath, 2001] proposed the iterative Gaussian-

Table 4.1: Various auto-regressive and flow-based methods expressed under a unified framework. All the conditioners can take inputs $\mathbf{x}$ instead of $\mathbf{z}$. The symbol 🔗 is used for weight sharing, 🐱 for use of masks for efficient implementation, 🏛 for universality of the method and, Δ if the method learns a triangular transformation explicitly (E) or implicitly (I). ? implies that universality of these methods has neither been proved or disproved although it can now be analyzed with ease using our framework. $S_j(z_j; \boldsymbol{\theta}_j)$ is defined in eq. (4.19) and $\mathfrak{P}_{2r+1}(z_j; \mathbf{a}_j)$ is defined in eq. (4.22).

Model	conditioner C_j output	$T_j(z_j\ ; C_j(z_1, \ldots, z_{j-1}))$	🔗	🐱	🏛	Δ
Mixture [McLachlan and Peel, 2004b]	$\boldsymbol{\theta}_j$	$S_j(z_j; \boldsymbol{\theta}_j)$	✗	✗	✓	I
[Bengio and Bengio, 1999]	$\boldsymbol{\theta}_j(z_{<j})$	$S_j(z_j; \boldsymbol{\theta}_j)$	✗	✗	?	I
MADE [Germain et al., 2015]	$\boldsymbol{\theta}_j(z_{<j})$	$S_j(z_j; \boldsymbol{\theta}_j)$	✓	✓	?	I
NICE [Dinh et al., 2015]	$\mu_j(z_{<l})$	$z_j + \mu_j \cdot \mathbf{1}_{j \notin [l]}$	✗	✗	?	E
NADE [Uria et al., 2016]	$\boldsymbol{\theta}_j(z_{<j})$	$S_j(z_j; \boldsymbol{\theta}_j)$	✓	✗	?	I
IAF [Kingma et al., 2016]	$\sigma_j(z_{<j}),\ \mu_j(z_{<j})$	$\sigma_j z_j + (1 - \sigma_j)\mu_j$	✓	✓	?	E
MAF [Papamakarios et al., 2017]	$\alpha_j(z_{<j}),\ \mu_j(z_{<j})$	$z_j \exp(\alpha_j) + \mu_j$	✓	✓	?	E
Real-NVP [Dinh et al., 2017]	$\alpha_j(z_{<l}),\ \mu_j(z_{<l})$	$\exp(\alpha_j \cdot \mathbf{1}_{j \notin [l]}) \cdot z_j + \mu_j \cdot \mathbf{1}_{j \notin [l]}$	✗	✗	?	E
NAF [Huang et al., 2018]	$\mathbf{w}_j(z_{<j})$	$\mathrm{DNN}(z_j\ ; \mathbf{w}_j)$	✓	✓	✓	E
SOS	$\mathbf{a}_j(z_{<j})$	$\mathfrak{P}_{2r+1}(z_j; \mathbf{a}_j)$	✓	✓	✓	E

ization technique, which essentially decomposes[5] the triangular map into the composition of a sequence of alternating diagonal maps $\mathbf{D}_t$ and linear maps $\mathbf{L}_t$. The diagonal maps are estimated using the univariate transform in Example 2 where G is standard normal and F is a mixture of standard normals. Later, [Laparra et al., 2011] simplified the linear map into random rotations. Both approaches, however, suffer cubic complexity w.r.t. dimension due to generating or evaluating the linear map. The recent work of [Tabak and Vanden-Eijnden, 2010, Tabak and Turner, 2013] coined the name *normalizing flow* and further exploited the straightforward but crucial observation that we can approximate the triangular map through a sequence of "simple" maps such as radial basis functions or rotations composed with diagonal maps. Similar simple maps have also been explored in [Ballé et al., 2016]. [Rezende and Mohamed, 2015] designed a "rank-1" (or radial) normalizing flow and applied it to variational inference, largely popularizing the idea in generative modelling. These approaches are not estimating a triangular map *per se*, but the main ideas are nevertheless similar.

[5]This can be made precise, much in the same way as decomposing a triangular matrix into the product of two rotation matrices and a diagonal matrix, i.e. the so-called Schur decomposition.

(*Bona fide*) **Triangular Approach**: [Deco and Brauer, 1995] (see also [Redlich, 1993]), to our best knowledge, is among the first to mention the name "triangular" *explicitly* in tasks (nonlinear independent component analysis) related to density estimation. More recently, [Dinh et al., 2015] recognized the promise of even simple triangular maps in density estimation. The (increasing) triangular map in [Dinh et al., 2015] consists of two simple (block) components: $T_1(\mathbf{x}_1) = \mathbf{x}_1$ and $T_2(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_2 + m(\mathbf{x}_1)$, where $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$ is a two-block partition and m is a map parameterized by a neural net. The advantage of this triangular map is its computational convenience: its Jacobian is trivially 1 and its inversion only requires evaluating m. [Dinh et al., 2015] applied different partitions of variables, iteratively composed several such simple triangular maps and combined with a diagonal linear map[6]. However, these triangular maps appear to be too simple and it is not clear if through composition they can approximate any increasing triangular map. In subsequent work, [Dinh et al., 2017] proposed the extension where $T_1(\mathbf{x}_1) = \mathbf{x}_1$ but $T_2(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_2 \odot \exp(s(\mathbf{x}_1)) + m(\mathbf{x}_1)$, where $\odot$ denotes the element-wise product. This map is again increasing triangular. [Moselhy and Marzouk, 2012] employed triangular maps for Bayesian posterior inference, which was further extended in [Marzouk et al., 2016] for sampling from an (unknown) target density. One of their formulations is essentially the same as our eq. (4.4).

Autoregressive Neural Models: A joint probability density function can be factorized into the product of marginal and conditionals:

$$q(x_1, \ldots, x_d) = q(x_1) \prod_{j=2}^{d} q(x_j | x_{j-1}, \ldots, x_1). \tag{4.16}$$

In his seminal work, [Neal, 1992b] proposed to model each (discrete) conditional density by a simple linear logistic function (with the conditioned variables as inputs). This was later extended by [Bengio and Bengio, 1999] using a two-layer nonlinear neural net. The recent work of [Uria et al., 2016] proposed to decouple the hidden layers in [Bengio and Bengio, 1999] and to introduce heavy weight sharing to reduce overfitting and computational complexity. Already in [Bengio and Bengio, 1999], univariate mixture models were mentioned as a possibility to model each conditional density, which was further substantiated in [Uria et al., 2016]. More precisely, they model the j-th conditional density as:

$$q(x_j | x_{j-1}, \ldots, x_1) = \sum_{\kappa=1}^{k} w_{j,\kappa} \, \mathcal{N}(x_j; \mu_{j,\kappa}, \sigma_{j,\kappa}) \tag{4.17}$$

$$\boldsymbol{\theta}_j := (w_{j,\kappa}, \mu_{j,\kappa}, \sigma_{j,\kappa})_{\kappa=1}^{k} = C_j(x_{j-1}, \ldots, x_1), \tag{4.18}$$

[6] They also considered a more general coupling that may no longer be triangular.

where C_j is the so-called conditioner network that outputs the parameters for the (univariate) mixture distribution in (4.17). According to Example 2 there exists a unique increasing map $S_j(\cdot\,;\boldsymbol{\theta}_j)$ that maps a univariate standard normal random variable z_j into x_j that follows (4.17). In other words,

$$x_j = S_j(z_j;\boldsymbol{\theta}_j) =: T_j(z_1,\ldots,z_{j-1},z_j), \tag{4.19}$$

where the last equality follows from induction, using the fact that $\boldsymbol{\theta}_j = C_j(x_{j-1},\ldots,x_1)$. Thus, as already pointed out in Remark 1, specifying a family of conditional densities as in (4.17) is **equivalent** as (implicitly) specifying a family of triangular maps. In particular, if we use a nonparametric family such as mixture of normals, then the induced triangular maps can approximate any increasing triangular map. The special case, when $k = 1$ in (4.17), was essentially dealt with by [Kingma et al., 2016]: for $k = 1$ the map $S_j(\boldsymbol{\theta}_j) = \mu_j + \sigma_j z_j$ hence the triangular map

$$T_j(z_1,\ldots,z_{j-1},z_j) = \mu_j(z_{<j}) + \sigma_j(z_{<j})\cdot z_j. \tag{4.20}$$

Obviously, not every triangular map can be written in the form (4.20), which is affine in z_j when $z_{<j}$ are fixed. To address this issue, [Kingma et al., 2016] composed several triangular maps in the form of (4.20), hoping this suffices to approximate a generic triangular map. In contrast, [Huang et al., 2018] proposed to replace the affine form in (4.20) with a univariate neural net (with z_j as input and μ_j and σ_j serve as weights). Lastly, based on binary masks, [Germain et al., 2015] and [Papamakarios et al., 2017] proposed efficient implementations of the above that compute all parameters in a single pass of the conditioner network. It should be clear now that (a) autoregressive models implement exactly a triangular map; (b) specifying the conditional densities directly is equivalent as specifying a triangular map explicitly.

Other Variants. Recurrent nets have also been used in autoregressive models (effectively triangular maps). For instance, [Oord et al., 2016] used LSTMs to directly specify the conditional densities while [MacKay et al., 2018] chose to explicitly specify the triangular maps. The two approaches, as alluded above, are equivalent, although one may be more efficient in certain applications than the other. [Oliva et al., 2018] tried to combine both while [Kingma and Dhariwal, 2018] used an invertible 1×1 convolution. We note that the work of [Ostrovski et al., 2018] models the conditional quantile function, which is equivalent to but can sometimes be more convenient than the conditional density.

Non-Triangular Flows. Sylvester Normalizing Flows (SNF) [Berg et al., 2018] and FFJORD [Grathwohl et al., 2019] are examples of normalizing flows that employ non-triangular maps. They both propose efficient methods to compute the Jacobian for change

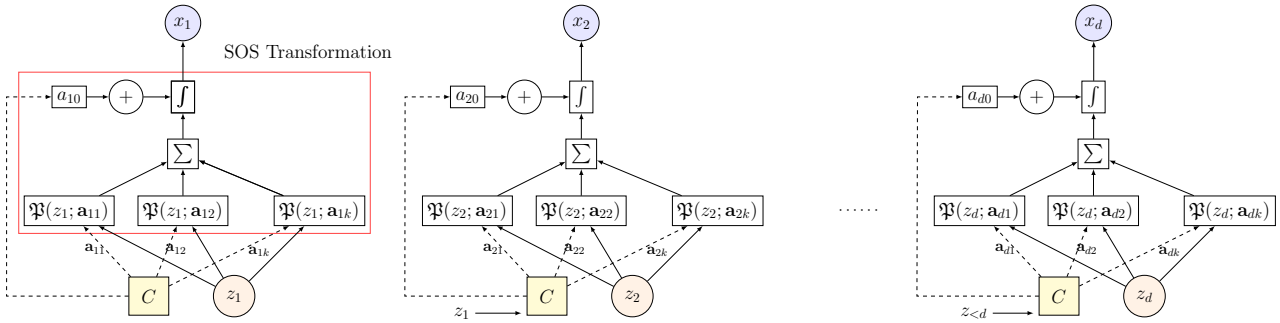


Figure 4.2: Schematic of SOS flows depicting the conditioner network and relevant transformations. We provide the algorithm in Algorithm 3 and Figure 4.3 shows the schematic for SOS Flows by stacking multiple blocks of SOS transformation.

of variables. SNF utilizes Sylvester's determinant theorem for that purpose. FFJORD, on the other hand, defines a generative model based on continuous-time normalizing flows proposed by [Chen et al., 2018] and evaluates the log-density efficiently using Hutchinson's trace estimator.

4.4 Sum-of-Squares Polynomial Flow

In Section 4.2 we developed a general framework for density estimation using triangular maps, and in Section 4.3 we showed the many recent generative models are all trying to estimate a triangular map in one way or another. In this section we give a surprisingly simple way to parameterize triangular maps, which, when plugged into (4.5), leads to a new density estimation algorithm that we call sum-of-squares (SOS) polynomial flow.

Our approach is motivated by some classical results on simulating univariate non-normal distributions. Let z be univariate standard normal. [Fleishman, 1978] proposed to simulate a non-normal distribution by fitting a degree-3 polynomial:

$$x = \mathfrak{P}_3(z; \mathbf{a}) = a_0 + a_1 z + a_2 z^2 + a_3 z^3, \tag{4.21}$$

where the coefficients $\{a_l\}$ are estimated by matching the first 4 moments of x with those of empirical data. This approach was quite popular in practice because it allows researchers to precisely control the moments (such as skewness and kurtosis). However, three difficulties remain: (1) with degree-3 polynomial one can only (approximately) simulate a (very) strict subset of non-normal distributions. This can be addressed by using polynomials of higher degrees and better quantile matching techniques [Headrick, 2009]. (2) The estimated

coefficients $\{a_l\}$ may not guarantee the monotonicity of the polynomial, making inversion and density evaluation difficult, if not impossible. (3) Extension to the multivariate case was done through composing a linear map [Vale and Maurelli, 1983], which can be quite inefficient.

We show that all three difficulties can be overcome using SOS flows. First, let us recall a classic result in algebra:

Theorem 13. *A univariate real polynomial is increasing iff it can be written as:*

$$\mathfrak{P}_{2r+1}(z;\mathbf{a}) = a_0 + \int_0^z \sum_{\kappa=1}^{k} \left(\sum_{l=0}^{r} a_{\kappa l} u^l \right)^2 \mathrm{d}u, \tag{4.22}$$

where $\mathbf{a} \in \mathbb{R}^{1+k(1+r)}$, $r \in \mathbb{N}$, *and* k *can be as small as 2.*

Note that a univariate increasing polynomial is strictly increasing iff it is not a constant. Theorem 13 is obtained by integrating a nonnegative polynomial, which is necessarily a sum-of-squares, see e.g. [Marshall, 2008]. Now, by applying (4.22) to model each conditional density in (4.19) we effectively addressed the last two issues above. Pleasantly, this approach strictly generalizes the affine triangular map (4.20) of [Kingma et al., 2016], which amounts to truncating $r = 0$ in (4.22). However, by using a larger r, we can learn certain densities more faithfully (especially for capturing higher order statistics), without significantly increasing the computational complexity. Additionally, implementing (4.22) in practice is simple: It can be computed *exactly* since it is an integral of univariate polynomials.

Lastly, we prove that as the degree r increases, we can approximate any triangular map. We will require the following Lemma for our proof.

Lemma 1 ([Mulansky and Neamtu, 1998])**.** *Let* S *be a dense subspace of* X *and let* $C \subseteq X$ *be a convex set such that* $\operatorname{int}(C) \neq \emptyset$. *Then* $C \cap S$ *is dense in* C.

Proof. Since the interior $\operatorname{int}(C)$ is open and nonempty, and S is dense, we know $\operatorname{int}(C) \cap S$ is dense in $\operatorname{int}(C)$. (Every open set of $\operatorname{int}(C)$ is also an open set of X, hence intersects the dense set S.) Moreover, since C is convex and $\operatorname{int}(C) \neq \emptyset$, we know $\operatorname{cl}(\operatorname{int}(C)) = \operatorname{cl}(C)$, hence $\operatorname{cl}(\operatorname{int}(C) \cap S) = \operatorname{cl}(C)$, i.e., $\operatorname{int}(C) \cap S$, whence also the "larger" set $C \cap S$, is dense in C. □

Equipped with the Lemma above, we are in a position to state and prove our result for universal approximation as the degree r increases.

Theorem 14. *Let* C *be the space of real univariate continuous functions, equipped with the topology of compact convergence. Then, the set of increasing polynomials is dense in the cone of increasing continuous functions.*

Proof. Let us define P to be the space of polynomials, and I the space of increasing functions. We need only prove on any compact set K, the set of polynomials of the form (4.22), i.e. $\mathsf{I} \cap \mathsf{P}$ thanks to Theorem 13, is dense in $\mathsf{C}(K) \cap \mathsf{I}$. By Weierstrass' theorem we know P is dense in $\mathsf{C}(K)$. Moreover, the convex subset $\mathsf{I} \cap \mathsf{C}(K)$ has nonempty interior (take say a linear function with positive slope). Applying Lemma 1 above completes the proof [7]. □

Since the topology of pointwise convergence is weaker than that of compact convergence (i.e. uniform convergence on every compact set), we immediately know that there exists a sequence of increasing polynomials of the form (4.22) that converges pointwise to any given continuous function. This universal property of increasing polynomials allows us to prove the universality of SOS flows, i.e. the capability of approximating any (continuous) triangular map.

SOS flow consists of two parts: an increasing (univariate) polynomial $\mathfrak{P}_{2r+1}(z_j; \mathbf{a}_j)$ of the form (4.22) for modelling conditional densities and a conditioner network $C_j(z_1, \ldots, z_{j-1})$ for generating the coefficients $\mathbf{a}_j$ of the polynomial $\mathfrak{P}_{2r+1}(z_j; \mathbf{a}_j)$. In other words, the triangular map learned using SOS flows has the following form:

$$\forall j, \quad T_j(z_1, \ldots, z_j) = \mathfrak{P}_{2r+1}\big(z_j; C_j(z_1, \ldots, z_{j-1})\big). \tag{4.23}$$

If we choose a universal conditioner (that can approximate any continuous function), such as a neural net, then combining with Theorem 13 and Theorem 14 we verify that the triangular maps in the form of (4.23) can approximate any increasing continuous triangular map in the pointwise manner. It then follows that the transformed densities will converge weakly to any desired target density (i.e. in distribution). This solves the first issue mentioned before Theorem 13. We remark that our universality proof for SOS flows is significantly shorter and more streamlined than the previous attempt of [Huang et al., 2018], and it can be seemingly extended to analyze other models summarized in Table 4.1.

As pointed out by [Papamakarios et al., 2017] we can also construct conditioner networks C_j that take inputs $x_1, \ldots, x_{j-1}$, instead of $z_1, \ldots, z_{j-1}$. They are equivalent in theory but one can be more convenient than the other, depending on the downstream application. Figure 4.2 illustrates the main components of a single-block SOS flow, where

[7] We would like to thank Csaba Szepesvári for bringing [Mulansky and Neamtu, 1998] to our attention, which allowed us to reduce a lengthy proof of Theorem 14 to the current slim one.

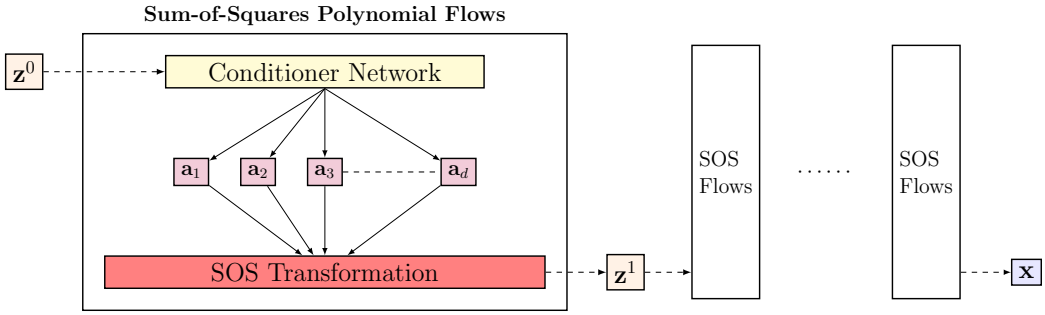


Figure 4.3: Schematic of SOS flows by stacking multiple blocks of SOS transformation.

we implement the conditioner network in the same way as in [Papamakarios et al., 2017]. To get a higher degree approximation, we can either increase r or stack a few single-block SOS flows, as shown in Figure 4.3. The former approach appears to be more general but also more difficult to train due to the larger number of parameters. Indeed, the effective number of parameters for SOS flows obtained by stacking L blocks with k polynomials of degree $2r+1$ is $L(1+k+kr)$ whereas achieving the same representation with a single block wide SOS flow would require $1+k((2r+1)^L+1)/2$ parameters. In Section 4.5.1 we perform simulated experiments to compare deep vs. wide SOS flows.

Algorithm 3 Sum-of-Squares Polynomial Flow

Input: Dataset $\mathcal{D} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$
Output: SOS transformation $\mathbf{T} = (T_1, T_2, \cdots, T_d)$
specify source distribution : $p(\mathbf{z})$
specify the degree in SOS flow: r
specify number of sum terms in SOS flow: k
procedure SOSFLOW($\mathcal{D}, p(\mathbf{z}), r, k$)
    evaluate $\mathbf{a}_j := C(\mathbf{x}_{i,<j}\ ; \mathbf{w}),\ \forall i \in [N]$
    evaluate $\mathbf{z}_i = \mathbf{T}^{-1}(\mathbf{x}_i\ ; \mathbf{a})$
    evaluate $\mathcal{L}_i = \log q(\mathbf{x}_i) = -\log p\big(\mathbf{T}^{-1}(\mathbf{x}_i)\big) + \sum_j \log \partial_j T_j\big(\mathbf{T}^{-1}(\mathbf{x}_i)\big)$
    $\mathbf{T}^* = \arg\min_{\mathbf{T}} \sum_{i=1}^N \mathcal{L}_i$
    Return : $\mathbf{T}^*$

SOS flow is similar to the neural autoregressive flow (NAF) of [Huang et al., 2018] in the sense that both are capable of approximating any (continuous) triangular map hence learning any desired target density. However, SOS flow has the following advantages:

- As mentioned before, SOS flow is a strict generalization of the inverse autoregressive flow (IAF) of [Kingma et al., 2016], which corresponds to setting $r = 0$.

- SOS flow is more interpretable, in the sense that its parameters (i.e. coefficients of the polynomials) directly control the first few moments of the target density.
- SOS flow may be easier to train, as there is no constraint on its parameters $\mathbf{a}$. In contrast, NAF needs to make sure the parameters are nonnegative[8].

4.5 Experiments

We performed several experiments on both synthetic datasets and real-world datasets to demonstrate the performance of SOS flows empirically. Through synthetic experiments, our goal was to study the effect of relative ordering of variables for the conditioner network while building SOS transformations, its ability to capture multi-modal distributions and the representational power of deep and wide SOS flows. For the real-world datasets, we compare SOS flows to other popular autoregressive and flow methods.

4.5.1 Simulated Experiments

We first explore the effect of relative ordering for the conditioner network in building transformations for SOS flows as well as mixture of Gaussians. For this task, we generated two sets of 2D densities given by $p(x_1, x_2) = \mathcal{N}(x_2\ ; 0, 4)\mathcal{N}(x_1\ ; 0.25x_2^2, 1)$ and $p(x_1, x_2) = \mathcal{N}(x_2\ ; 2, 2)\mathcal{N}(x_1\ ; 1/3x_2^3, 1.5)$. However, we trained both SOS flows and GMMs with the reverse order i.e. (x_1, x_2) i.e. although in the true density x_1 is dependent on x_2, for the transformation we reverse this order to have x_2 dependent on x_1. For SOS flows we tested using both deep and wide flows whereas for MoGs we tested with varying number of components for each conditional distributions. We present the plots in Figure 4.4. The best performance here is by a deep SOS flow. Furthermore, while a flat SOS flow is able to achieve almost the same geometrical shape as the target density, its learned density still differs from the true density. For mixture of Gaussians, a large number of components for each conditional improved the performance of the resulting model.

We also tested the representational power of deep and wide SOS flows and the results are given in Figure 4.5. In the first row, the true transformation was simulated by stacking multiple blocks of SOS transformation. Subsequently, we generated the target density using this transformation and estimated it using a deep flow, wide flow, wide-deep flow

[8] A typical remedy is to re-parameterize through an exponential transform, which, however, may lead to overflows or underflows.

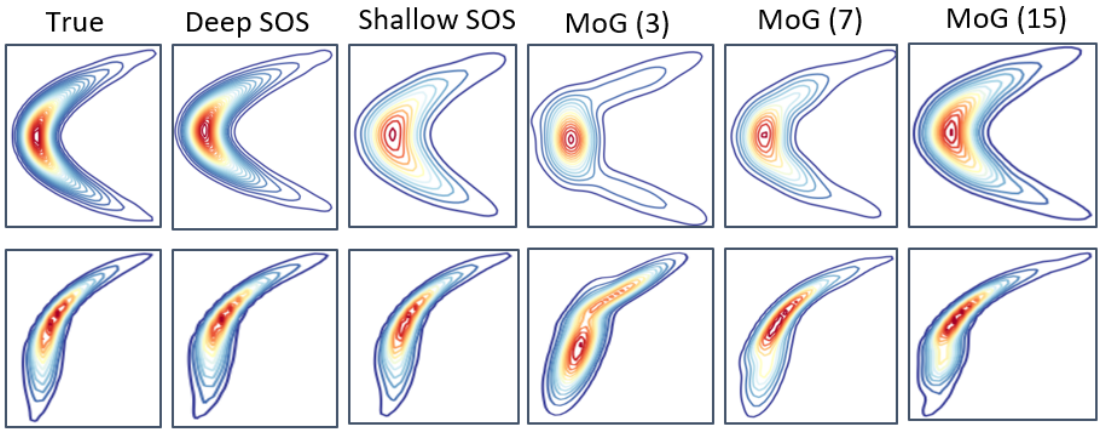


Figure 4.4: **Top:** Leftmost is true density $p(x_1, x_2) = \mathcal{N}(x_2\ ; 0, 4)\mathcal{N}(x_1\ ; 0.25x_2^2, 1)$. The second plot shows the density learnt by SOS flows with 3 blocks and a sum of 2 polynomials with degree 3 with ordering (x_1, x_2). Third plot shows the density learnt by SOS flows with 1 block and a sum of 2 polynomials with degree 4 and ordering (x_1, x_2). The last three plots estimate this density using a Mixture of Gaussian conditionals with varying components given in parenthesis and ordering (x_1, x_2). **Bottom:** Same as Top but with target density given by $p(x_1, x_2) = \mathcal{N}(x_2\ ; 2, 2)\mathcal{N}(x_1\ ; 0.33x_1^3, 1.5)$.

and mixture of Gaussians. In the second row, we simulated the true transformation using a single block SOS transformation and performed the same experiment as before. In both simulations, we tried to break our model by adding random noise to the coefficients of simulated transformation. As the figure shows, however, both deep and wide variants performed equally well in terms of representation. As expected however, the training time for wider flows was significantly longer than that for deeper flows.

Next, in Figure 4.6, we demonstrate the ability of SOS flows to represent transformations that lead to multi-modal densities by generating data from a mixture of Gaussians for two cases - well-connected and disjoint support. The true transformation can be computed exactly following Example 2. We show three transformations learned by SOS flows for each case corresponding to a deep SOS flow, wide SOS flow and wide-deep SOS flow. As is evident, SOS flows were fairly successful in learning the transformations. We further estimated the parameters of these simulated densities using Gaussian mixtures trained using maximum likelihood under three cases - exact (same number of components as target density), under-specified (lesser number of components) and over-specified. Subsequently, we plot the resulting transformation in each case following Example 2. While, GMMs with

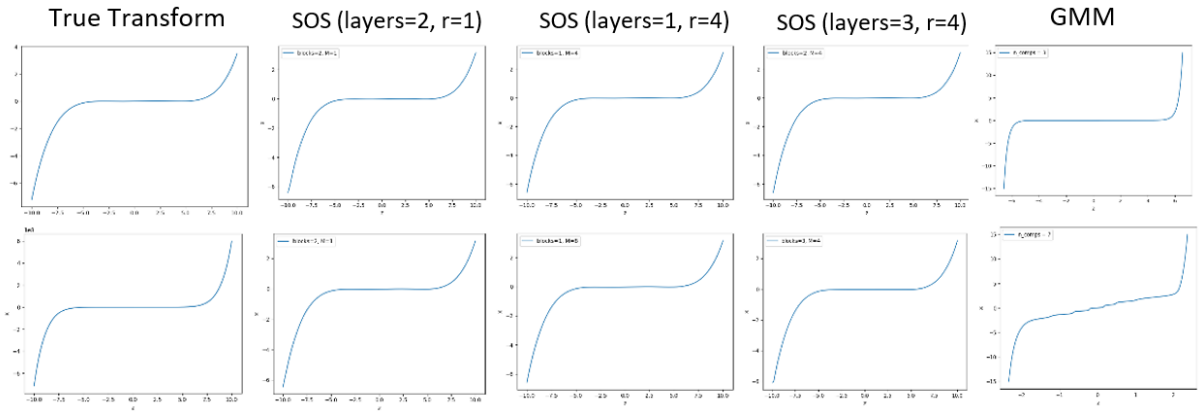


Figure 4.5: **Top Row:** Transformation defined by a deep SOS flow with $r = 1$ and blocks =4. The next three plots show SOS flows learning this transformation with different configurations (deep, wide and, wide-deep). The last plot shows the transformation learned when a Gaussian mixture model learns the density (or transformation). **Bottom Row:** Same as Top Row but the true transformation was derived by a wide and shallow SOS flow with $r = 4$ and blocks=1.

exact components work well as expected, the transformations learned by under-specified and over-specified models are not as good. This experiment also goes on to show that using a parameterized density to model conditionals is equivalent to implicitly learning a transformation.

4.5.2 Real-World Datasets

We also performed density estimation experiments on 5 real world datasets that include four datasets from the UCI repository and BSDS300. These datasets have been previously considered for comparison of flows based methods [Huang et al., 2018].

The SOS transformation was trained using maximum likelihood method with source density as standard normal distribution. We used stochastic gradient descent to train our models with a batch size of 1000, learning rate = 0.001, number of stacked blocks = 8, number of polynomials (k) = 5 and, degree of polynomials (r) = 4 with number of epochs for training = 40. We compare our method to previous works on normalizing flows and autoregressive models which include MADE-MoG [Germain et al., 2015], MAF [Papamakarios et al., 2017], MAF-MoG [Papamakarios et al., 2017], TAN [Oliva et al., 2018] and NAFs [Huang et al., 2018]. In Table 4.3, we report the average log-likelihood

Table 4.2: Negative test log-likelihoods for various density estimation models on image datasets (lower is better). * results/models used multi-scale convolutional architectures.

Method	MNIST	CIFAR10
Real-NVP	1.06*	3.49*
Glow	1.05*	3.35*
FFJORD	0.99*	3.40*
MADE	2.04	5.67
MAF	1.89	4.31
SOS	1.81	4.18

Table 4.3: Average test log-likelihoods and standard deviation for SOS flows over 10 trials (higher is better). The other methods report the average log-likelihood and standard deviation over five trials. The numbers in the parenthesis indicate the number of stacked blocks for the resultant transformation.

Method	Power	Gas	Hepmass	MiniBoone	BSDS300
MADE	0.40 ± 0.01	8.47 ± 0.02	-15.15 ± 0.02	-12.24 ± 0.47	153.71 ± 0.28
MAF affine (5)	0.14 ± 0.01	9.07 ± 0.02	-17.70 ± 0.02	-11.75 ± 0.44	155.69 ± 0.28
MAF affine (10)	0.24 ± 0.01	10.08 ± 0.02	-17.73 ± 0.02	-12.24 ± 0.45	154.93 ± 0.28
MAF MoG (5)	0.30 ± 0.01	9.59 ± 0.02	-17.39 ± 0.02	-11.68 ± 0.44	156.36 ± 0.28
TAN	0.60 ± 0.01	12.06 ± 0.02	-13.78 ± 0.02	-11.01 ± 0.48	159.80 ± 0.07
NAF DDSF (5)	0.62 ± 0.01	11.91 ± 0.13	-15.09 ± 0.40	-8.86 ± 0.15	157.73 ± 0.04
NAF DDSF (10)	0.60 ± 0.02	11.96 ± 0.33	-15.32 ± 0.23	-9.01 ± 0.01	157.43 ± 0.30
SOS (7)	0.60 ± 0.01	11.99 ± 0.41	-15.15 ± 0.10	-8.90 ± 0.11	157.48 ± 0.41

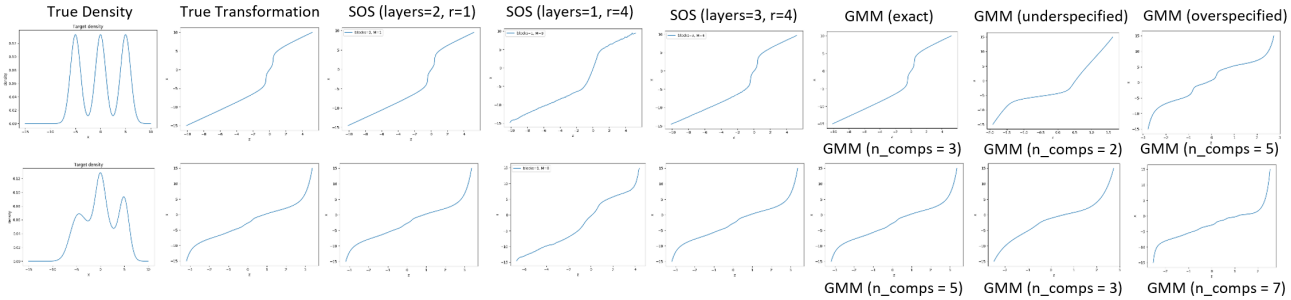


Figure 4.6: **Top Row:** First plot from the left shows the target density, a mixture of three component Gaussians with means = (-5, 0, 5), variances = (1, 1, 1) and, weights = (1/3, 1/3, 1/3). The second plot shows the exact transformation required to transform a standard Gaussian to this mixture. The next three plots shows the transformation learned by SOS flows with different configurations (deep, wide and wide-deep, respectively). The last three plots show the transformation learned by estimating the parameters of the Gaussian mixture using log-likelihood with exact (3), under-specified (2) and over-specified (5) number of components respectively. **Bottom Row:** Same as Top Row but with target density being a mixture of five Gaussians with means = (-5, -2, 0, 2, 5), variances = (1.5, 2, 1, 2, 1) and, weights = 0.2 each.

obtained using 10 fold cross-validation on held-out test sets for SOS flows. The performance reported for other methods are those reported in [Huang et al., 2018]. The results show that SOS flows are able to achieve competitive performance as compared to other methods.

4.6 Summary

We presented a unified framework for estimating complex densities using monotone and bijective triangular maps. The main idea is to specify one-dimensional transformations and then iteratively extend to higher-dimensions using conditioner networks. Under this framework, we analyzed popular autoregressive and flow based methods, revealed their similarities and differences, and provided a unified and streamlined approach for understanding the representation power of these methods. Along the way we uncovered a new sum-of-squares polynomial flow that we show is universal, interpretable and easy to train. We discussed the various advantages of SOS flows for stochastic simulation and density estimation, and we performed various experiments on simulated data to explore the properties of SOS flows. Lastly, SOS flows achieved competitive results on real-world datasets. In the future we plan to carry out the analysis indicated in Table 4.1, and to formally

establish the respective advantages between deep and wide SOS flows.

Chapter 5

Conclusion and Discussion

In this section, I summarize the contributions made in this thesis and discuss possible future directions.

In Chapter 2, we formally established the relationships among some popular unsupervised learning models, such as latent tree graphical models, hierarchical tensor formats and sum-product networks, based on which we further provided a unified treatment of exponential separation in *exact* representation size between deep architectures and shallow ones. Surprisingly, for *approximate* representation, the conditional gradient algorithm can approximate any homogeneous mixture within accuracy ϵ by combining $O(1/\epsilon^2)$ shallow models, where the hidden constant may decrease exponentially wrt the depth. Our experiments on both synthetic and real datasets confirmed our theoretical findings on the benefits of depth in representing deep mixture models. An interesting direction for future work will be to formalize Theorem 2 and explicitly characterize the increase in size from one format to the other. Another direction will be to explore structure learning algorithms for deep homogeneous mixture models by exploiting the compact representation using higher order tensors. The problem of structure learning can then be stated as learning a higher order weight tensor by keeping the leaf distributions fixed in a data driven manner.

We began Chapter 3 by proposing an online Bayesian Moment Matching algorithm to learn the parameters of Gaussian Mixture models and demonstrated its use in a distributed manner. We showed through empirical analysis that the online Bayesian Moment Matching outperforms online EM and online Variational Bayes on real-world datasets and demonstrated that distributing the algorithm over several machines results in faster running times without significantly compromising accuracy, which is particularly advantageous when running time is a major bottleneck. In the future, it will be interesting to undertake

a theoretical analysis of the Bayesian Moment Matching algorithm. In particular, it will be worth exploring the effect of the approximation step and if certain desirable properties of exact Bayesian learning (see [Doob, 1949]) that lead to its consistency are still preserved despite the approximation.

Subsequently, in Chapter 3, we considered problems in domains where data is produced by a population of individuals that exhibit a certain degree of variability. Traditionally, machine learning techniques ignore this variability and train a single model under the assumption that the population is homogeneous. While several offline transfer learning techniques have already been proposed to account for population heterogeneity, we described an online transfer learning technique (the first to our best knowledge) that incrementally determines which source models best explain a streaming sequence of observations while predicting the corresponding hidden states. We achieved this by adapting the online Bayesian moment matching algorithm to hidden Markov models with Gaussian mixture emission distributions. Our experimental results confirmed the effectiveness of the approach in three real-world applications: activity recognition, sleep stage recognition and flow direction prediction.

In the future, this work could be extended in several directions. Since it is not always clear how many basis models should be used and that the observation sequences of target individuals can necessarily be explained by a weighted combination of basis models, it would be interesting to explore techniques that can automatically determine a good number of basis models and can generate new basis models on the fly when existing ones are insufficient. Furthermore, since recurrent neural networks (RNNs) have been shown to outperform HMMs with GMMs emission distributions in some applications such as speech recognition [Graves et al., 2013], it would be interesting to generalize our online transfer learning technique to RNNs.

Finally, in Chapter 4, we turned our focus from mixture models to neural density estimation. We presented a unified framework for estimating complex densities using monotone and bijective triangular maps. The main idea is to specify one-dimensional transformations and then iteratively extend to higher-dimensions using conditioner networks. Under this framework, we analyzed popular autoregressive and flow based methods, revealed their similarities and differences, and provided a unified and streamlined approach for understanding the representation power of these methods. Along the way we uncovered a new sum-of-squares polynomial flow that we show is universal, interpretable and easy to train. We performed various experiments on simulated data to explore the properties of SOS flows. and also showed that SOS flows achieved competitive results on real-world datasets compared to other flow based methods.

This work can be extended in several directions in the future. Firstly, a direct extension of this work is to carry out the analysis indicated in Table 4.1, and to formally establish the respective advantages between deep and wide SOS flows. Secondly, it will be interesting to explore and understand how neural density estimation methods compare to other classical techniques like Variational inference and identify its advantages and disadvantages over other methods. Another interesting direction will be in studying flow based methods and triangular transformations for capturing specific tail behaviour in the target distribution. Particularly, the main aim will be to characterize the change in tails of the source density after applying a single block of triangular transformation. This line of work can shed light on the potential benefits of adapting tails of the source density with potential applications for fat-tailed variational inference. Additionally, a major problem with neural density estimation methods is that they are computationally expensive and are not scalable to very high dimensions. A potentially impactful direction of work will be to construct sparse triangular transformations that can have applications in Bayesian deep learning and Bayesian reinforcement learning. Finally, an open research direction for flow based methods is their efficient extension for discrete random variables.

References

[Sil, 2007] (2007). The Visual Scoring of Sleep in Adults. *Journal of Clinical Sleep Medicine*, 3(2):121–131. (Cited on page 72.)

[Mal, 2013] (2013). Performance of an Automated Polysomnography Scoring System Versus Computer-assisted Manual Scoring. *Sleep*, 36(4):573–582. (Cited on page 71.)

[Ajtai, 1983] Ajtai, M. (1983). $\sum_1^1$-formulae on finite structures. *Annals of pure and applied logic*, 24(1):1–48. (Cited on page 44.)

[Akaike, 1974] Akaike, H. (1974). A new look at the statistical model identification. In *Selected Papers of Hirotugu Akaike*, pages 215–222. Springer. (Cited on page 9.)

[Al-Stouhi and Reddy, 2011] Al-Stouhi, S. and Reddy, C. K. (2011). Adaptive boosting for transfer learning using dynamic updates. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 60–75. Springer. (Cited on page 58.)

[Alexandrova, 2006] Alexandrova, D. (2006). Convergence of Triangular Transformations of Measures. *Theory of Probability & Its Applications*, 50(1):113–118. (Cited on page 79.)

[Alon, 2009] Alon, N. (2009). Perturbed identity matrices have high rank: Proof and applications. *Combinatorics, Probability and Computing*, 18(1-2):3–15. (Cited on pages 35 and 38.)

[Anandkumar et al., 2012] Anandkumar, A., Hsu, D., Huang, F., and Kakade, S. M. (2012). Learning mixtures of tree graphical models. In *Advances in Neural Information Processing Systems*. (Cited on page 39.)

[Anderer et al., 2005] Anderer, P., Gruber, G., Parapatics, S., Woertz, M., Miazhynskaia, T., Klosch, G., Saletu, B., Zeitlhofer, J., Barbanoj, M. J., Danker-Hopfe, H., Himanen,

S.-L., Kemp, B., Penzel, T., Grozinger, M., Kunz, D., Rappelsberger, P., Schlogl, A., and Dorffner, G. (2005). An E-health Solution for Automatic Sleep Classification According to Rechtschaffen and Kales: Validation Study of the Somnolyzer 24 x 7 Utilizing the Siesta Database. *Neuropsychobiology*, 51(3):115–133. (Cited on page 71.)

[Ballé et al., 2016] Ballé, J., Laparra, V., and Simoncelli, E. P. (2016). Density modeling of images using a generalized normalization transformation. In *ICLR*. (Cited on page 84.)

[Bao et al., 2011] Bao, F. S., Liu, X., and Zhang, C. (2011). PyEEG: An Open Source Python Module for EEG/MEG Feature Extraction. *Computational Intelligence and Neuroscience*, 2011:1–7. (Cited on page 72.)

[Barron, 1993] Barron, A. R. (1993). Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945. (Cited on page 45.)

[Bartlett, 1998] Bartlett, P. L. (1998). The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE transactions on Information Theory*, 44(2):525–536. (Cited on page 45.)

[Baum and Petrie, 1966] Baum, L. E. and Petrie, T. (1966). Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, 37(6):1554–1563. (Cited on page 28.)

[Beal, 2003] Beal, M. J. (2003). *Variational algorithms for approximate Bayesian inference*. University of London London. (Cited on page 54.)

[Bengio and Bengio, 1999] Bengio, Y. and Bengio, S. (1999). Modeling High-Dimensional Discrete Data with Multi-Layer Neural Networks. In *NeurIPS*. (Cited on pages 76, 84, and 85.)

[Bengio et al., 2014] Bengio, Y., Laufer, E., Alain, G., and Yosinski, J. (2014). Deep generative stochastic networks trainable by backprop. In *International Conference on Machine Learning*, pages 226–234. (Cited on page 43.)

[Berg et al., 2018] Berg, R. v. d., Hasenclever, L., Tomczak, J. M., and Welling, M. (2018). Sylvester normalizing flows for variational inference. In *UAI*. (Cited on page 86.)

[Bishop, 2006] Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer. (Cited on page 2.)

[Bogachev et al., 2005] Bogachev, V. I., Kolesnikov, A. V., and Medvedev, K. V. (2005). Triangular transformations of measures. *Sbornik: Mathematics*, 196(3):309–335. (Cited on pages 76, 79, and 83.)

[Braverman, 2011] Braverman, M. (2011). Poly-logarithmic independence fools bounded-depth boolean circuits. *Communications of the ACM*, 54(4):108–115. (Cited on pages 44 and 45.)

[Broderick et al., 2013] Broderick, T., Boyd, N., Wibisono, A., Wilson, A. C., and Jordan, M. I. (2013). Streaming variational bayes. In *Advances in Neural Information Processing Systems*, pages 1727–1735. (Cited on pages 49, 50, and 54.)

[Cappé and Moulines, 2009] Cappé, O. and Moulines, E. (2009). On-line expectation–maximization algorithm for latent data models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 71(3):593–613. (Cited on pages 50 and 54.)

[Carlier et al., 2010] Carlier, G., Galichon, A., and Santambrogio, F. (2010). From Knothe's Transport to Brenier's Map and a Continuation Method for Optimal Transport. *SIAM Journal on Mathematical Analysis*, 41(6):2554–2576. (Cited on page 83.)

[Caron and Traynor, 2005] Caron, R. and Traynor, T. (2005). The zero set of a polynomial. Technical report. (Cited on page 33.)

[Chacón and Duong, 2018] Chacón, J. E. and Duong, T. (2018). *Multivariate kernel smoothing and its applications*. Chapman and Hall/CRC. (Cited on page 4.)

[Chattopadhyay et al., 2011] Chattopadhyay, R., Krishnan, N. C., and Panchanathan, S. (2011). Topology preserving domain adaptation for addressing subject based variability in semg signal. In *AAAI Spring Symposium: Computational Physiology*, pages 4–9. (Cited on page 57.)

[Chen and Gopinath, 2001] Chen, S. S. and Gopinath, R. A. (2001). Gaussianization. In *NeurIPS*, pages 423–429. (Cited on pages 76 and 83.)

[Chen et al., 2018] Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018). Neural ordinary differential equations. In *NeurIPS*, pages 6572–6583. (Cited on page 87.)

[Chieu et al., 2006] Chieu, H. L., Lee, W. S., and Kaelbling, L. P. (2006). Activity recognition from physiological data using conditional random fields. (Cited on page 57.)

[Choi et al., 2011] Choi, M. J., Tan, V. Y. F., Anandkumar, A., and S.Willsky, A. (2011). Learning latent tree graphical models. *Journal of Machine Learning Research*, 12:1771–1812. (Cited on page 25.)

[Cohen et al., 2017] Cohen, N., Sharir, O., Levine, Y., Tamari, R., Yakira, D., and Shashua, A. (2017). Analysis and design of convolutional networks via hierarchical tensor decompositions. arXiv:1705.02302v4. (Cited on page 27.)

[Cohen et al., 2016] Cohen, N., Sharir, O., and Shashua, A. (2016). On the expressive power of deep learning: A tensor analysis. In *Conference on Learning Theory*, pages 698–728. (Cited on pages 18, 27, 29, 34, 35, 36, 39, 45, and 46.)

[Cohen and Shashua, 2016] Cohen, N. and Shashua, A. (2016). Convolutional rectifier networks as generalized tensor decompositions. In *ICML*. (Cited on page 18.)

[Cook et al., 2013] Cook, D., Feuz, K. D., and Krishnan, N. C. (2013). Transfer learning for activity recognition: A survey. *Knowledge and information systems*, 36(3):537–556. (Cited on page 57.)

[Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314. (Cited on page 45.)

[Dai et al., 2007] Dai, W., Yang, Q., Xue, G.-R., and Yu, Y. (2007). Boosting for transfer learning. In *Proceedings of the 24th international conference on Machine learning*, pages 193–200. ACM. (Cited on page 58.)

[Darwiche, 2003] Darwiche, A. (2003). A differential approach to inference in bayesian networks. *Journal of the ACM (JACM)*, 50(3):280–305. (Cited on pages 17, 21, 22, and 45.)

[Deco and Brauer, 1995] Deco, G. and Brauer, W. (1995). Nonlinear higher-order statistical decorrelation by volume-conserving neural architectures. *Neural Networks*, 8(4):525–535. (Cited on page 85.)

[Delalleau and Bengio, 2011] Delalleau, O. and Bengio, Y. (2011). Shallow vs. deep sum-product networks. In *Advances in Neural Information Processing Systems*, pages 666–674. (Cited on pages 18, 29, 34, 35, 45, and 46.)

[Dempster et al., 1977] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38. (Cited on page 50.)

[Dinh et al., 2014] Dinh, L., Krueger, D., and Bengio, Y. (2014). Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*. (Cited on page 43.)

[Dinh et al., 2015] Dinh, L., Krueger, D., and Bengio, Y. (2015). NICE: Non-linear independent components estimation. In *ICLR workshop*. (Cited on pages 12, 76, 84, and 85.)

[Dinh et al., 2017] Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2017). Density estimation using Real NVP. In *ICLR*. (Cited on pages 12, 76, 84, and 85.)

[Doob, 1949] Doob, J. L. (1949). Application of the theory of martingales. *Le calcul des probabilites et ses applications*, pages 23–27. (Cited on page 98.)

[Dua and Graff, 2017] Dua, D. and Graff, C. (2017). UCI machine learning repository. (Cited on page 55.)

[Durkan et al., 2019] Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. (2019). Neural spline flows. *arXiv preprint arXiv:1906.04032*. (Cited on page 12.)

[Eldan and Shamir, 2016] Eldan, R. and Shamir, O. (2016). The power of depth for feedforward neural networks. In *Conference on Learning Theory*, pages 907–940. (Cited on page 45.)

[Epanechnikov, 1969] Epanechnikov, V. A. (1969). Non-parametric estimation of a multivariate probability density. *Theory of Probability & Its Applications*, 14(1):153–158. (Cited on page 9.)

[Fisher et al., 1920] Fisher, R. A. et al. (1920). 012: A mathematical examination of the methods of determining the accuracy of an observation by the mean error, and by the mean square error. (Cited on page 11.)

[Fix and Hodges, 1951] Fix, E. and Hodges, J. L. (1951). Discriminatory analysis. nonparametric discrimination: consistency properties. *International Statistical Review/Revue Internationale de Statistique*, 57(3):238–247. (Cited on pages 7 and 8.)

[Fleishman, 1978] Fleishman, A. I. (1978). A method for simulating non-normal distributions. *Psychometrika*, 43(4):521–532. (Cited on page 87.)

[Frank and Wolfe, 1956] Frank, M. and Wolfe, P. (1956). An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110. (Cited on pages 37 and 39.)

[Freedman et al., 1998] Freedman, D., Pisani, R., and Purves, R. (1998). Statistics (3rd edn). (Cited on page 8.)

[Friedman et al., 1984] Friedman, J. H., Stuetzle, W., and Schroeder, A. (1984). Projection Pursuit Density Estimation. *Journal of the American Statistical Association*, 79(387):599–608. (Cited on page 83.)

[Germain et al., 2015] Germain, M., Gregor, K., Murray, I., and Larochelle, H. (2015). MADE: Masked autoencoder for distribution estimation. In *ICML*, pages 881–889. (Cited on pages 84, 86, and 93.)

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *NeurIPS*, pages 2672–2680. (Cited on page 76.)

[Grathwohl et al., 2019] Grathwohl, W., Chen, R. T. Q., Betterncourt, J., Sutskever, I., and Duvenaud, D. (2019). Ffjord: Free-form continuous dynamics for scalable reversible generative models. In *ICLR*. (Cited on page 86.)

[Graves et al., 2013] Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE. (Cited on page 98.)

[Guvenir and Uysal, 2000] Guvenir, H. A. and Uysal, I. (2000). Bilkent university function approximation repository. (Cited on page 55.)

[Hackbusch, 2012] Hackbusch, W. (2012). *Tensor Spaces and Numerical Tensor Calculus*. Springer. (Cited on pages 17, 20, 23, 25, 32, and 46.)

[Hastad, 1986] Hastad, J. (1986). Almost optimal lower bounds for small depth circuits. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 6–20. ACM. (Cited on page 44.)

[Headrick, 2009] Headrick, T. C. (2009). *Statistical Simulation Power Method Polynomials and Other Transformations*. CRC Press. (Cited on page 87.)

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comp*, 9(8):1735–1780. (Cited on page 68.)

[Hoffman et al., 2013] Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. (2013). Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347. (Cited on page 54.)

[Hoogeboom et al., 2019] Hoogeboom, E., Peters, J. W., Berg, R. v. d., and Welling, M. (2019). Integer discrete flows and lossless compression. *arXiv preprint arXiv:1905.07376*. (Cited on page 2.)

[Hornik et al., 1989] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366. (Cited on page 45.)

[Huang et al., 2018] Huang, C.-W., Krueger, D., Lacoste, A., and Courville, A. (2018). Neural Autoregressive Flows. In *ICML*. (Cited on pages 12, 76, 77, 84, 86, 89, 90, 93, and 95.)

[Huber, 1985] Huber, P. J. (1985). Projection Pursuit. *The Annals of Statistics*, 13(2):435–475. (Cited on page 83.)

[Ishteva, 2015] Ishteva, M. (2015). Tensors and latent variable models. In *The 12th International Conference on Latent Variable Analysis and Signal Separation (LVA/ICA)*, pages 49–55. (Cited on page 28.)

[Jaini et al., 2017] Jaini, P., Chen, Z., Carbajal, P., Law, E., Middleton, L., Regan, K., Schaekermann, M., Trimponias, G., Tung, J., and Poupart, P. (2017). Online bayesian transfer learning for sequential data modeling. In *International Conference on Learning Representations (ICLR)*. (Cited on pages 14 and 50.)

[Jaini and Poupart, 2017] Jaini, P. and Poupart, P. (2017). Online and distributed learning of gaussian mixture models by bayesian moment matching. In *Workshop on Advances in Approximate Bayesian Inference, NIPS*. (Cited on pages 14 and 49.)

[Jaini et al., 2018] Jaini, P., Poupart, P., and Yu, Y. (2018). Deep homogeneous mixture models: representation, separation, and approximation. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 7136–7145. (Cited on page 14.)

[Jaini et al., 2016] Jaini, P., Rashwan, A., Zhao, H., Liu, Y., Banijamali, E., Chen, Z., and Poupart, P. (2016). Online algorithms for sum-product networks with continuous variables. In *Conference on Probabilistic Graphical Models*, pages 228–239. (Cited on page 15.)

[Jaini et al., 2019] Jaini, P., Selby, K., and Yu, Y. (2019). Sum-of-squares polynomial flow. In *International Conference on Machine Learning (ICML)*. (Cited on pages 12, 14, and 75.)

[Jensen et al., 2010] Jensen, P. S., Sorensen, H. B. D., Leonthin, H. L., and Jennum, P. (2010). Automatic Sleep Scoring in Normals and in Individuals with Neurodegenerative Disorders According to New International Sleep Scoring Criteria. *Journal of Clinical Neurophysiology: Official Publication of the American Electroencephalographic Society*, 27(4):296–302. (Cited on page 71.)

[Jordan et al., 1999] Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233. (Cited on page 4.)

[Kahn, 1955] Kahn, H. (1955). Use of different monte carlo sampling techniques. (Cited on page 3.)

[Kahn and Harris, 1951] Kahn, H. and Harris, T. E. (1951). Estimation of particle transmission by random sampling. *National Bureau of Standards applied mathematics series*, 12:27–30. (Cited on page 3.)

[Kingma and Dhariwal, 2018] Kingma, D. P. and Dhariwal, P. (2018). Glow: Generative flow with invertible 1x1 convolutions. In *NeurIPS*. (Cited on pages 12 and 86.)

[Kingma et al., 2016] Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. (2016). Improved variational inference with inverse autoregressive flow. In *NeurIPS*, pages 4743–4751. (Cited on pages 12, 76, 77, 84, 86, 88, and 90.)

[Kingma and Welling, 2014] Kingma, D. P. and Welling, M. (2014). Auto-encoding variational Bayes. In *ICLR*. (Cited on pages 4 and 76.)

[Knothe, 1957] Knothe, H. (1957). Contributions to the theory of convex bodies. *The Michigan Mathematical Journal*, 4(1):39–52. (Cited on page 83.)

[Laparra et al., 2011] Laparra, V., Camps-Valls, G., and Malo, J. (2011). Iterative Gaussianization: From ICA to Random Rotations. *IEEE Transactions on Neural Networks*, 22(4):537–549. (Cited on pages 76 and 84.)

[Larochelle and Murray, 2011] Larochelle, H. and Murray, I. (2011). The neural autoregressive distribution estimator. In *AISTATS*, pages 29–37. (Cited on page 76.)

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. (Cited on page 42.)

[LeCun et al., 2004] LeCun, Y., Huang, F. J., and Bottou, L. (2004). Learning methods for generic object recognition with invariance to pose and lighting. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–104. IEEE. (Cited on page 42.)

[Lee et al., 2017] Lee, H., Ge, R., Ma, T., Risteski, A., and Arora, S. (2017). On the ability of neural nets to express distributions. In *Conference on Learning Theory*, pages 1271–1296. (Cited on page 45.)

[Li and Barron, 2000] Li, J. Q. and Barron, A. R. (2000). Mixture density estimation. In *Advances in neural information processing systems*, pages 279–285. (Cited on pages 18 and 39.)

[Liang and Klein, 2009] Liang, P. and Klein, D. (2009). Online em for unsupervised models. In *Proceedings of human language technologies: The 2009 annual conference of the North American chapter of the association for computational linguistics*, pages 611–619. Association for Computational Linguistics. (Cited on pages 50 and 55.)

[MacKay, 2003] MacKay, D. J. (2003). *Information theory, inference and learning algorithms*. Cambridge university press. (Cited on page 2.)

[MacKay et al., 2018] MacKay, M., Vicol, P., Ba, J., and Grosse, R. B. (2018). Reversible Recurrent Neural Networks. In *NeurIPS*, pages 9043–9054. (Cited on page 86.)

[Marshall, 2008] Marshall, M. (2008). *Positive Polynomials and Sums of Squares*. AMS. (Cited on page 88.)

[Martens et al., 2013] Martens, J., Chattopadhya, A., Pitassi, T., and Zemel, R. (2013). On the representational efficiency of restricted boltzmann machines. In *Advances in Neural Information Processing Systems*, pages 2877–2885. (Cited on page 45.)

[Martens and Medabalimi, 2014] Martens, J. and Medabalimi, V. (2014). On the expressive efficiency of sum product networks. *arXiv preprint arXiv:1411.7717*. (Cited on pages 18, 20, 21, 22, 29, 33, 34, 36, 39, and 46.)

[Marzouk et al., 2016] Marzouk, Y., Moselhy, T., Parno, M., and Spantini, A. (2016). Sampling via Measure Transport: An Introduction. In Ghanem, R., Higdon, D., and Owhadi, H., editors, *Handbook of Uncertainty Quantification*, pages 1–41. Springer. (Cited on pages 78 and 85.)

[McLachlan and Peel, 2004a] McLachlan, G. and Peel, D. (2004a). *Finite mixture models*. John Wiley & Sons. (Cited on pages 10, 11, 12, 17, and 20.)

[McLachlan and Peel, 2004b] McLachlan, G. and Peel, D. (2004b). *Finite mixture models*. John Wiley & Sons. (Cited on page 84.)

[Medvedev, 2008] Medvedev, K. V. (2008). Certain properties of triangular transformations of measures. *Theory of Stochastic Processes*, 14(1):95–99. (Cited on page 80.)

[Meila and Jordan, 2000] Meila, M. and Jordan, M. I. (2000). Learning with mixtures of trees. *Journal of Machine Learning Research*, 1:1–48. (Cited on page 39.)

[Metropolis et al., 1953] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092. (Cited on page 1.)

[Moselhy and Marzouk, 2012] Moselhy, T. A. E. and Marzouk, Y. M. (2012). Bayesian inference with optimal maps. *Journal of Computational Physics*, 231(23):7815–7850. (Cited on pages 79 and 85.)

[Motamedi-Fakhr et al., 2014] Motamedi-Fakhr, S., Moshrefi-Torbati, M., Hill, M., Hill, C. M., and White, P. R. (2014). Signal Processing Techniques Applied to Human Sleep EEG Signals - A Review. *Biomedical Signal Processing and Control*, 10:21–33. (Cited on page 72.)

[Mourad et al., 2013] Mourad, R., Sinoquet, C., Zhang, N. L., Liu, T., and Leray, P. (2013). A survey on latent tree models and applications. *Journal of Artificial Intelligence Research*, 47:157–203. (Cited on pages 17, 25, 26, 27, and 119.)

[Mulansky and Neamtu, 1998] Mulansky, B. and Neamtu, M. (1998). Interpolation and Approximation from Convex Sets. *Journal of Approximation Theory*, 92(1):82–100. (Cited on pages 88 and 89.)

[Murphy, 2012] Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press. (Cited on page 50.)

[Nadaraya, 1964] Nadaraya, E. A. (1964). On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142. (Cited on page 9.)

[Neal, 1992a] Neal, R. M. (1992a). Connectionist learning of belief networks. *Artificial intelligence*, 56(1):71–113. (Cited on page 11.)

[Neal, 1992b] Neal, R. M. (1992b). Connectionist learning of belief networks. *Artificial Intelligence*, 56(1):71–113. (Cited on pages 76 and 85.)

[Neal and Hinton, 1998] Neal, R. M. and Hinton, G. E. (1998). A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer. (Cited on page 50.)

[Nesterov, 1983] Nesterov, Y. (1983). A method of solving a convex programming problem with convergence rate O(1/sqr(k)). *Soviet Mathematics Doklady*, 27:372–376. (Cited on page 69.)

[Nguyen and McLachlan, 2016] Nguyen, H. D. and McLachlan, G. J. (2016). On approximations via convolution-defined mixture models. *arXiv preprint arXiv:1611.03974*. (Cited on page 17.)

[Oliva et al., 2018] Oliva, J., Dubey, A., Zaheer, M., Poczos, B., Salakhutdinov, R., Xing, E., and Schneider, J. (2018). Transformation Autoregressive Networks. In *ICML*, pages 3898–3907. (Cited on pages 86 and 93.)

[Omar, 2016] Omar, F. (2016). Online bayesian learning in probabilistic graphical models using moment matching with applications. (Cited on page 51.)

[Oord et al., 2016] Oord, A. V., Kalchbrenner, N., and Kavukcuoglu, K. (2016). Pixel Recurrent Neural Networks. In *ICML*, pages 1747–1756. (Cited on pages 12 and 86.)

[Ostrovski et al., 2018] Ostrovski, G., Dabney, W., and Munos, R. (2018). Autoregressive Quantile Networks for Generative Modeling. In *ICML*, pages 3936–3945. (Cited on page 86.)

[Pan and Yang, 2010] Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359. (Cited on page 57.)

[Papamakarios et al., 2017] Papamakarios, G., Pavlakou, T., and Murray, I. (2017). Masked autoregressive flow for density estimation. In *NeurIPS*, pages 2338–2347. (Cited on pages 12, 76, 84, 86, 89, 90, and 93.)

[Parno and Marzouk, 2018] Parno, M. and Marzouk, Y. (2018). Transport Map Accelerated Markov Chain Monte Carlo. *SIAM/ASA Journal on Uncertainty Quantification*, 6(2):645–682. (Cited on page 78.)

[Parzen, 1962] Parzen, E. (1962). On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076. (Cited on page 8.)

[Pascanu et al., 2013] Pascanu, R., Montufar, G., and Bengio, Y. (2013). On the number of response regions of deep feed forward networks with piece-wise linear activations. *arXiv preprint arXiv:1312.6098*. (Cited on page 45.)

[Pearson, 1894] Pearson, K. (1894). Contributions to the mathematical theory of evolution. *Philosophical Transactions of the Royal Society of London. A*, 185:71–110. (Cited on pages 8 and 10.)

[Pearson, 1902a] Pearson, K. (1902a). On the systematic fitting of curves to observations and measurements. *Biometrika*, 1(3):265–303. (Cited on page 7.)

[Pearson, 1902b] Pearson, K. (1902b). On the systematic fitting of curves to observations and measurments: Part ii. *Biometrika*, 2(1):1–23. (Cited on page 7.)

[Peeraully et al., 2012] Peeraully, T., Yong, M.-H., Chokroverty, S., and Tan, E.-K. (2012). Sleep and Parkinson's disease: A review of case-control polysomnography studies. *Movement Disorders*, 27(14):1729–1737. (Cited on page 71.)

[Peharz et al., 2017] Peharz, R., Gens, R., Pernkopf, F., and Domingos, P. (2017). On the latent variable interpretation in sum-product networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(10):2030–2044. (Cited on page 23.)

[Peherstorfer and Marzouk, 2018] Peherstorfer, B. and Marzouk, Y. (2018). A transport-based multifidelity preconditioner for Markov chain Monte Carlo. (Cited on page 78.)

[Poon and Domingos, 2011] Poon, H. and Domingos, P. (2011). Sum-product networks: A new deep architecture. In *Uncertainty in Artificial Intelligence*. UAI. (Cited on pages 17, 21, 22, 23, 43, and 45.)

[Poupart et al., 2016] Poupart, P., Chen, Z., Jaini, P., Fung, F., Susanto, H., Geng, Y., Chen, L., Chen, K., and Jin, H. (2016). Online flow size prediction for improved network routing. In *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, pages 1–6. IEEE. (Cited on page 15.)

[Punjabi et al., 2015] Punjabi, N. M., Shifa, N., Dorffner, G., Patil, S., Pien, G., and Aurora, R. N. (2015). Computer-Assisted Automated Scoring of Polysomnograms Using the Somnolyzer System. *Sleep*, 38(10):1555–1566. (Cited on page 71.)

[Rabiner, 1989] Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286. (Cited on page 28.)

[Rashidi and Cook, 2009] Rashidi, P. and Cook, D. J. (2009). Transferring learned activities in smart environments. In *Intelligent Environments*, pages 185–192. (Cited on page 57.)

[Redlich, 1993] Redlich, A. N. (1993). Supervised Factorial Learning. *Neural Computation*, 5(5):750–766. (Cited on page 85.)

[Rezende and Mohamed, 2015] Rezende, D. J. and Mohamed, S. (2015). Variational inference with normalizing flows. In *ICML*. (Cited on pages 76 and 84.)

[Rezende et al., 2014] Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *ICML*. (Cited on page 76.)

[Rosenberg and Van Hout, 2013] Rosenberg, R. S. and Van Hout, S. (2013). The American Academy of Sleep Medicine Inter-scorer Reliability Program: Sleep Stage Scoring. *Journal of Clinical Sleep Medicine*. (Cited on page 71.)

[Rosenblatt, 1952] Rosenblatt, M. (1952). Remarks on a Multivariate Transformation. *The Annals of Mathematical Statistics*, 23(3):470–472. (Cited on page 83.)

[Rosenblatt, 1956] Rosenblatt, M. (1956). Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, pages 832–837. (Cited on pages 7 and 8.)

[Rudelson and Vershynin, 2008] Rudelson, M. and Vershynin, R. (2008). The least singular value of a random square matrix is $O(n^{-1/2})$. *C. R. Acad. Sci. Paris, Ser. I*, 345:893–896. (Cited on page 37.)

[Rudin, 1987] Rudin, W. (1987). *Real and Complex Analysis*. McGraw-Hill, 3rd edition. (Cited on page 78.)

[Scott, 2015] Scott, D. W. (2015). *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons. (Cited on page 8.)

[Shannon, 1948] Shannon, C. E. (1948). A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423. (Cited on page 2.)

[Shao et al., 2015] Shao, L., Zhu, F., and Li, X. (2015). Transfer learning for visual categorization: A survey. *IEEE transactions on neural networks and learning systems*, 26(5):1019–1034. (Cited on page 57.)

[Sharir et al., 2018] Sharir, O., Tamari, R., Cohen, N., and Shashua, A. (2018). Tensorial mixture models. arXiv:1610.04167v5. (Cited on pages 17, 18, 19, 21, 27, 28, 29, 34, 35, 42, and 43.)

[Sipser, 1983] Sipser, M. (1983). Borel sets and circuit complexity. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 61–69. ACM. (Cited on page 44.)

[Sohl-Dickstein et al., 2015] Sohl-Dickstein, J., Weiss, E. A., Maheswaranathan, N., and Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. *arXiv preprint arXiv:1503.03585*. (Cited on page 43.)

[Song et al., 2013] Song, L., Park, H., Ishteva, M., Parikh, A., and Xing, E. (2013). Hierarchical tensor decomposition of latent tree graphical models. In *ICML*. (Cited on pages 24 and 25.)

[Spantini et al., 2018] Spantini, A., Bigoni, D., and Marzouk, Y. (2018). Inference via low-dimensional couplings. *Journal of Machine Learning Research*, 19:1–71. (Cited on page 79.)

[Sutskever et al., 2013] Sutskever, I., Martens, J., Dahl, G. E., and Hinton, G. E. (2013). On the importance of initialization and momentum in deep learning. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 1139–1147. (Cited on page 69.)

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. (2014). Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112. (Cited on page 69.)

[Tabak and Turner, 2013] Tabak, E. G. and Turner, C. V. (2013). A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164. (Cited on pages 76 and 84.)

[Tabak and Vanden-Eijnden, 2010] Tabak, E. G. and Vanden-Eijnden, E. (2010). Density estimation by dual ascent of the log-likelihood. *Communications in Mathematical Sciences*, 8(1):217–233. (Cited on pages 76 and 84.)

[Talagrand, 1996] Talagrand, M. (1996). Transportation cost for Gaussian and other product measures. *Geometric & Functional Analysis*, 6(3):587–600. (Cited on page 83.)

[Tarter and Lock, 1893] Tarter, M. E. and Lock, M. D. (1893). *Model-free curve estimation*, volume 56. CRC Press. (Cited on page 10.)

[Taylor and Stone, 2009] Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685. (Cited on page 57.)

[Telgarsky, 2016] Telgarsky, M. (2016). Benefits of depth in neural networks. *COLT*. (Cited on page 45.)

[Theano Development Team, 2016] Theano Development Team (2016). Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688. (Cited on page 69.)

[Tieleman and Hinton, 2012] Tieleman, T. and Hinton, G. (2012). Lecture 6.5 - rmsprop, coursera: Neural networks for machine learning. Technical report. (Cited on page 69.)

[Titterington, 1984] Titterington, D. M. (1984). Recursive parameter estimation using incomplete data. page 46(2):257267. (Cited on page 50.)

[Tsybakov, 2009] Tsybakov, A. B. (2009). *Introduction to Nonparametric Estimation*. Springer. (Cited on pages 1, 9, and 21.)

[Uria et al., 2016] Uria, B., Côté, M.-A., Gregor, K., Murray, I., and Larochelle, H. (2016). Neural autoregressive distribution estimation. *The Journal of Machine Learning Research*, 17:7184–7220. (Cited on pages 76, 84, and 85.)

[Vale and Maurelli, 1983] Vale, C. D. and Maurelli, V. A. (1983). Simulating multivariate nonnormal distributions. *Psychometrika*, 48(3):465–471. (Cited on page 88.)

[Villani, 2008] Villani, C. (2008). *Optimal Transport: Old and New*. Springer. (Cited on pages 76 and 78.)

[Wand and Jones, 1994] Wand, M. P. and Jones, M. C. (1994). *Kernel smoothing*. Chapman and Hall/CRC. (Cited on page 9.)

[Watson, 1964] Watson, G. S. (1964). Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 359–372. (Cited on page 9.)

[Wilcoxon, 1950] Wilcoxon, F. (1950). Some rapid approximate statistical procedures. *Annals of the New York Academy of Sciences*, pages 808–814. (Cited on pages 55 and 69.)

[Williams and Peng, 1990] Williams, R. and Peng, J. (1990). An efficient gradient-based algorithm for online training of recurrent network trajectories. *Neural Computation*, 2(4):490–501. (Cited on page 69.)

[Yao, 1985] Yao, A. C.-C. (1985). Separating the polynomial-time hierarchy by oracles. In *Foundations of Computer Science, 1985., 26th Annual Symposium on*, pages 1–10. IEEE. (Cited on page 44.)

[Yao and Doretto, 2010] Yao, Y. and Doretto, G. (2010). Boosting for transfer learning with multiple sources. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1855–1862. IEEE. (Cited on page 58.)

[Yu and Schuurmans, 2012] Yu, Y.-L. and Schuurmans, D. (2012). Rank/norm regularization with closed-form solutions: Application to subspace clustering. *arXiv preprint arXiv:1202.3772*. (Cited on page 36.)

[Zhao et al., 2015] Zhao, H., Melibari, M., and Poupart, P. (2015). On the relationship between sum-product networks and bayesian networks. In *International Conference on Machine Learning*, pages 116–124. (Cited on page 23.)

[Zhao et al., 2011] Zhao, Z., Chen, Y., Liu, J., Shen, Z., and Liu, M. (2011). Cross-people mobile-phone based activity recognition. In *Twenty-Second International Joint Conference on Artificial Intelligence*. (Cited on page 57.)

APPENDICES

Appendix A

More results on comparing different models

This appendix section provides more details to compliment section 2.4. We provide additional details and examples to support the arguments that we made in section 2.4.

A.1 Converting an LTM to S^{3}PN

Given an LTM, we can build a corresponding S^{3}PN as follows: starting from the root of the LTM, for each hidden variable H that takes k possible values $\{1, \ldots, k\}$ and that has r children nodes $\{V_1, \ldots, V_r\}$, we create a sum node S_H with k children product nodes $\{\mathsf{P}_{H,1}, \ldots \mathsf{P}_{H,k}\}$, each of which has r children sum nodes $\{\mathsf{S}_{V_1}, \ldots, \mathsf{S}_{V_r}\}$. We set the weight from the sum node S_H to its i-th child product node $\mathsf{P}_{H,i}$ as $\Pr(H = i|\pi(H) = j)$, if S_H connects to the j-th child product node of the parent hidden variable $\pi(H)$ (for the root, the parent is empty). If the child V_t is a hidden variable, we continue the construction similarly, while if $V_t = X_i$ is an observed variable, then we replace the sum node S_{V_t} with the density $f_j^i(x_i)$, assuming S_{V_t} is connected to the j-th child product node of the parent hidden variable H. Algorithm 4 summarizes this construction, and Figure 2.1 illustrates the idea using a simple latent class model (LCM) [Mourad et al., 2013].

In Algorithm 4, we describe a procedure to convert a latent tree model (LTM) as described in (2.15) to a self-similar SPN (S^{3}PN). In Figure A.1 we give another example to illustrate Algorithm 4.

Algorithm 4 Converting an LTM into an S^3PN

Input : A latent tree model with L levels and $(\mathbf{X}, \mathbf{H})$
Output : An equivalent S^3PN
for $l \leftarrow L$ to 1 **do**
    $\mathbf{H}_l :=$ {all nodes in current level from left to right order}
    while $\mathbf{H}_l \neq \emptyset$ **do**
        $h = \text{Pop}(\mathbf{H}_l)$
        if $h \in \mathbf{X}$ **then**
            for $j \leftarrow 1$ to $|\pi_h|$ **do**
                create a leaf v_j with distribution $Pr(h|\pi_h = j)$
                add an edge (v_j, p_j^{l-1})
        else
            if $\pi_h \neq \emptyset$ **then**
                create $|\pi_h|$ sum nodes i.e. $S_h^l := \{s_1^l, s_2^l, \cdots s_{|\pi_h|}^l\}$
                for $j \leftarrow 1$ to $|\pi_h|$ **do**
                    add an edge (s_j^l, p_j^{l-1})
                create $|h|$ product nodes i.e. $P_h^l := \{p_1^l, p_2^l, \cdots, p_h^l\}$
                for $i \leftarrow 1$ to $|\pi_h|$ **do**
                    for $j \leftarrow 1$ to $|h|$ **do**
                        create an edge (s_i^l, p_j^l) with weight $w_{i,j} = Pr(h = j|\pi_h = i)$
            else
                create one sum node s_h
                create h product nodes i.e. $P_h := \{p_1, p_2, \cdots, p_h\}$
                for $i \leftarrow 1$ to $|h|$ **do**
                    add an edge (s_h, p_i) with weight $Pr(h = i)$

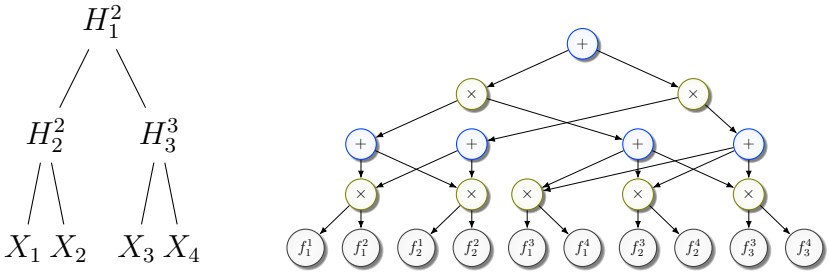


Figure A.1: Left shows a latent tree model with three discrete hidden variables $\mathbf{H} = \{H_1, H_2, H_3\}$ and four observed variables $\mathbf{X} = \{X_1, X_2, X_3, X_4\}$. where H_1, H_2 are binary and H_3 can take three discrete values. The second figure shows the equivalent SPN representing the latent tree.

In Figure A.1, we consider a latent tree graphical model forming a balanced binary tree with three binary hidden variables $\mathbf{H} = \{H_1, H_2, H_3\}$ and four observed variables $\mathbf{X} = \{X_1, X_2, X_3, X_4\}$. The tree has 3 levels and is rooted at H_1. The algorithm proceeds by going through each level one at a time. In the first iteration, it encounters the root node H_1 and creates a corresponding sum node in the SPN. It then creates two (equal to all possible states of H_1) product nodes as children to this sum node. The edge on the left denotes the edge when $H_1 = 0$ and has weight $\Pr(H_1 = 0)$ and the edge on the right denotes edge when $H_1 = 1$ and has weight $\Pr(H_1 = 1)$. In the next iteration, the algorithm proceeds to level 2 which has two hidden variables H_2 and H_3. The algorithm processes these one at a time. First, it takes H_2 and creates two sum nodes corresponding to H_2, one child each for each product node in the previous layer. Next, two product nodes are created and an edge is created between each of these product nodes and each of the sum node created before corresponding to H_2. The same procedure is then repeated for H_3 but with three edges accounting for the fact that it can take three values. Finally, for each observed variable, a leaf distribution $\Pr(X|\pi_X)$ is induced.

A.2 Example for TMM as an LTM and S^{3}PN

In fig. A.2, we give a representation for fig. 2.6 without redundancy. The figure shows that a TMM can be represented by an LTM and hence an S^{3}PN.

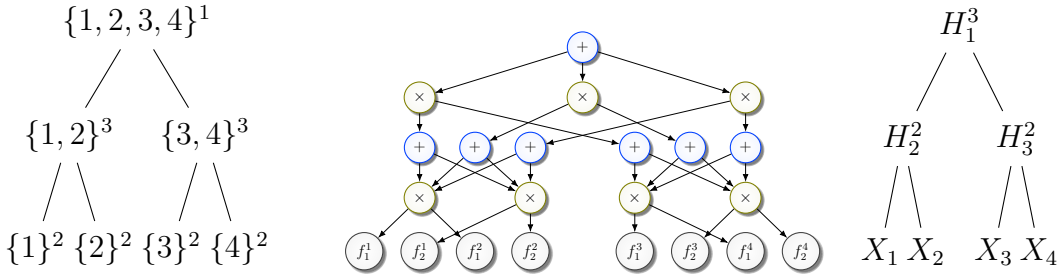


Figure A.2: Left: A dimension-partition tree in HTF. The superscripts indicate the number of bases, which should remain constant on each level. Middle: The equivalent S^3PN. The leaf f_j^i is the j-th basis of vector space V_i. Right: An equivalent TMM. The superscripts indicate the number of values each hidden variable can take (again, remaining constant on each level).

A.3 Example for TMM $\subsetneq$ LTM

In fig. 2.5 we gave an S^3PN that is equivalent to an LTM but not a TMM. It is evident from the figure that the LTM consists of hidden variables at the same level with different number of possible states. This arrangement, however, is not allowed in TMM.

A compact representation of fig. 2.5 without redundancy is given in fig. A.3.

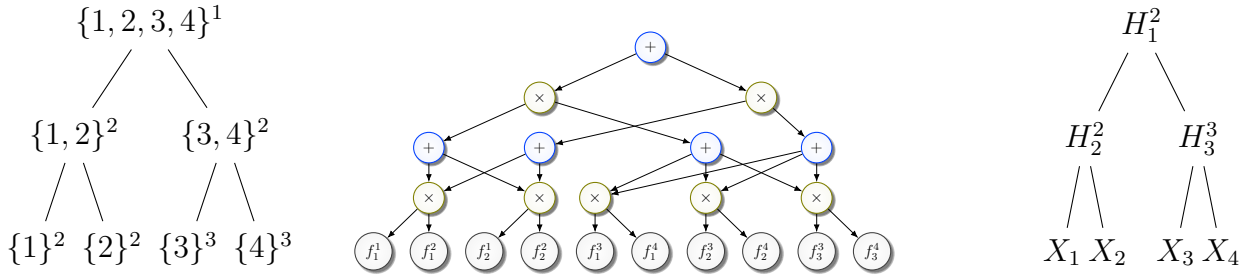


Figure A.3: Left: A dimension-partition tree in HTF_+. The superscripts indicate the number of bases. Middle: The equivalent S^3PN. The leaf f_j^i is the j-th basis of vector space V_i. Right: An equivalent LTM. The superscripts indicate the number of values each hidden variable can take.