Combining Sum Product Networks and Variational Autoencoders

Ping Liang Tan

Supervisor: Dr Robert Peharz

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Masters of Philosphy

Hughes Hall August 2018

Declaration

I, Ping Liang Tan of Hughes Hall, being a candidate for the MPhil in Machine Learning, Speech and Language Technology, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Word Count : 14333

Signed :
Date : 17 August 2018

Ping Liang Tan
August 2018

Acknowledgements

First, I would like to thank my supervisor, Dr Robert Peharz. Without his knowledge of Sum Product Networks, or his kind, attentive guidance, this MPhil project would not have been remotely possible. Learning from Dr Peharz has been an enjoyable and intellectually stimulating experience.

Second, a special shout goes out to the MLSALT 2017-18 cohort which has taught me much about working hard and having fun. Amongst my treasured memories are the late nights in the MPhil room, where we often wondered if we can finish our coursework before the deadline, and our trip to Italy, where we continued doing our coursework in the trains, plane, and cafes.

Not to be forgotten are the other people I met here in Cambridge. I have learnt a lot by meeting people from diverse cultural and academic backgrounds. Having meals with familiar faces has also helped me feel at home away from home.

Finally, I would like to thank my parents. Their beliefs in my ability and emphatic sighs when I played too much had helped spur me on to achieve my goals.

Abstract

Sum Product Networks (SPNs) are a class of deep, tractable probabilistic models. Starting from tractable distributions over few random variables, an SPN hierarchically composes distributions together using only sums and products, which maintain the composite distribution's tractability. A Variational Autoencoder (VAE) on the other hand is a flexible generative probabilistic model that is implemented with a neural network and trained with the assistance of another inference network. Although marginal likelihoods are canonically intractable in VAEs, this thesis proposed using VAEs as the starting distributions of an SPN, resulting in a hybrid model referred to as SP-VAE.

SP-VAE maintains its tractability by using only small VAEs with low-dimensional latent variables. Probabilistic queries like exact marginal likelihood and marginalization that are canonically intractable in VAEs are now tractable in SP-VAE. Experiments on three image datasets and in three distribution types show that SP-VAE is competitive against VAE as density estimators on continuous and categorical image datasets but not on binary datasets. More significantly, SP-VAE does not need inference networks for learning, and preserves some of SPN's robustness against low resource datasets and overfitting.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

“Combine SPNs and VAEs? Why not!"

1.1 Motivation

Amongst machine learning techniques capable of modelling probability distributions, Sum Product Networks (SPNs) [32] stands out for its ability to tractably represent complicated distributions, compute any marginal on that distribution and answer inference queries. Its tractability arises from completeness and decomposability constrains, while its expressiveness arises from the use of depth in a directed acyclic graph that can reuse sub-computation.

In its current understanding due to [28], SPNs can be thought of as like LEGO®. It is a composite model whose basic building blocks are simple distributions, such as the Bernoulli distribution for binary variables, categorical distribution for discrete variables, and Gaussian distribution for continuous variables. Using sum nodes and product nodes, many simple distributions can be composed together into a complicated, multivariate distribution that retains many desirable properties of the individual distributions — for example, easy sampling, exact likelihoods, and tractable inference.

This understanding naturally motivates exploring the use of more sophisticated building block distributions. Looking into recent deep learning literature, neural probabilistic models (VAEs[21], PixelRNNs[40], PixelCNNs[41], and MADEs[16]) are some of the most sophisticated probabilistic models available today and are capable of even modelling “distributions over natural images". Using them to increase the complexity of SPN leaves might be an easier way of boosting SPN performance than learning more complicated SPN structures.

1.2 Research Aims and Scope

In this thesis, I explored the use of variational autoencoders (VAEs) as SPN leaves, leading to a hybrid model called *SP-VAE*.

Since SPNs, SP-VAEs, and VAEs are probabilistic models, the natural task on which to evaluate them is the traditional problem of density estimation, which is a task of learning an estimate of a true distribution from a set of samples. The samples used in this case are three academic image datasets — CALTECH101, MNIST handwritten digits, and cropped Street View House Numbers.

The central aim of this thesis then is to **evaluate how SP-VAE compares against traditional SPNs, and vanilla VAE on image density estimation problems**.

In addition to comparing SP-VAE, this thesis provides working code of SP-VAE, explanations of tradeoffs inherent in SP-VAE, inference algorithms for SP-VAE, and possible applications of SP-VAE.

1.3 Overview

In the next two sections, the current literature on SPNs and VAEs is reviewed with occassional introduction of original insights. In section 4, SP-VAE is defined, and learning and inference algorithms are proposed for it. Section 5 documents the processing of dataset and considerations made to ensure fair experimentation. Then in section 6, relevant considerations made about implementing my experiments in Tensorflow are summarized. All these sections build up to section 7, where SP-VAE is shown to be competitive against VAEs, better than traditional SPNs, and possesses hybrid properties between SPNs and VAEs. Research ideas that were encountered but not prioritized during this project are collected in section 8. Finally, section 9 summarizes the insights from experiments, and note their significance to current research trends.

1.4 Notation

In this thesis, I will use upper case letters to refer to a random variable, and lower case letters to refer to instantiations of the random variables. Whether the variable is a scalar, vector or a set will be clear in the context of discussion.

θ	: Parameters of a variational autoencoder
w	: Weights of an SPN's sum node(s)
X	: Input random variables
Y	: Discrete latent variables of SPN's sum nodes
Z	: Continuous latent variables of a VAE
ρ_C	: Categorical distribution associated with sum node C.

Chapter 2

Sum Product Networks

“An SPN is a probabilistic model designed for efficient inference."

In this chapter, I will summarize the theoretical developments of SPNs up to its current understanding, and illustrate its fundamental algorithms. I will also describe a generic SPN structure for image data which will be needed for the rest of this thesis, and note its equivalence to tensorial decomposition.

2.1 Historical Development

Sum Product Networks (SPNs) were originally proposed [9] by Darwiche under the name of *arithmetic circuit* (AC). AC originally refers to a tool in computational complexity theory for measuring the complexity of evaluating polynomials. But Darwiche introduced [8] it into artificial intelligence literature as a compact graphical representation of a discrete variable Bayesian Network (BN).

This was done by first expressing the discrete variable BN as a (possibly intractable) multilinear polynomial of *indicator variables*, also referred to by Darwiche as a *network polynomial*. Indicator variables are like the indicator function in that they can only take values of 0 or 1. But algebraically, they are treated like continuous variables that can be differentiated.

Then through BN’s classical jointree methods, the BN can be compiled into a compact AC. Standard inference queries on the BN, like calculating the likelihood of evidence, was shown to be equivalent to evaluating (forward pass) and differentiating (backpropagation) that AC. All tractable BNs can thus be represented as ACs. But ACs are also more general than BNs because in addition to *conditional independence* (CI), ACs can represent *conditional specific independence* (CSI).

Poon and Domingos [32] built upon Darwiche’s work on ACs. First, they relaxed the condition that ACs need to be *decomposable* to just being *consistent*, and analyzed the effect of this relaxation. For example, a *complete* but *inconsistent* AC overestimates the BN’s probability, while an *incomplete* but *consistent* AC underestimates. Poon and Domingos argued that if ACs with unnormalized weights are *complete* and *consistent*, then they are more general than traditional ACs. This relaxed AC is referred to in the seminal paper [32] as an SPN.

In the same paper, Poon and Domingos further generalized SPNs to handle continuous variables, and treated SPN as a probability model on its own rights instead of just a compilation target for BNs. They provided structure and parameter learning algorithms, and also described a missing data inference algorithm. Experiments showed that SPNs over continuous variables significantly outperformed traditional neural probability models, like deep belief networks and deep Boltzmann machines, on a difficult inference task of image inpainting. SPN's status as a tractable probabilistic model was thus established.

Since then, theoretical research has clarified the correctness of SPNs' core algorithms. One key development was by Peharz [28] which showed that the MPE inference algorithm proposed by [32] is correct only for *selective* SPNs. Peharz proposed [28] an augmentation algorithm to make an SPN's latent variables explicit as indicator variables, thereby allowing the SPN to be selective. Since inferring latent variables is also a subroutine of the EM algorithm, Peharz derived [28] a correct version of EM algorithm for SPNs.

Additionally, theoretical research was interested in the expressiveness of SPN. Multiple authors [30, 44, 7], showed showed that the relaxation of decomposability into consistency does not allow for exponentially more compact SPNs. A deeper SPN can, however, be exponentially more compact than a shallow SPN [10].

On the application front, SPNs have been applied to new problem domains like language modeling [6], and speech processing [29].

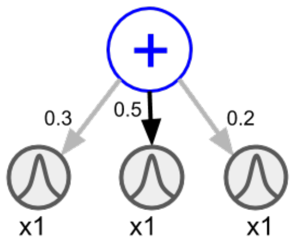
In practice, one of the most tricky factors in using SPNs is learning the structure from data. To that end, many heuristic-based approaches have been proposed and studied. [26, 27, 1, 12, 35]

2.1.1 Definitions

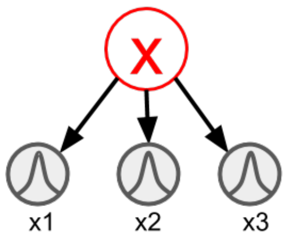
Now, having mentioned some italicized concepts used in classifying SPNs, I will define them:

The **scope** of a sum or product node is the set of random variable which appears in its leaves.

- **decomposable**: the children of product nodes have disjoint scopes
- **consistent**: sampling the AC/SPN will never produce conflicting instantiation of any random variable
- **selective**: no sum node in an evaluation (upwards pass) has more than one child with nonzero weighted activation
- **complete**: all the children of a sum node have the same scope



(a) Sum node



(b) Product node

Fig. 2.1 Fundamental composition operations of a sum product network

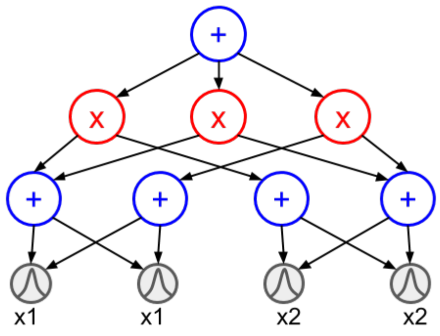


Fig. 2.2 Naive Bayes SPN

2.2 Generalized SPN

In its current understanding, a sum product network is a large, complicated multivariate probability distribution hierarchically composed together from many tiny (often univariate) probability distributions. There are two composition operations — a sum and a product. A sum operation creates a mixture distribution from any number of distributions over the same scope. (See Figure 2.1a.) The result is a more complicated distribution over the same scope. A product operation creates an independent joint probability distribution from any number of distributions of disjoint scope. (See Figure 2.1b.) The result is a distribution over a greater number of random variables.

The starting distributions before any operation are represented graphically as leaf nodes. The distributions resulting from the sum and product compositions are represented as sum and product nodes. Arrows point from the sum or product nodes to their input distributions.

By hierarchically composing distributions, a deep acyclic directed graph of sum and product nodes can be built on top of the starting collection of leaf nodes. The top of the graph is usually a single node representing the overall composite distribution. An example is shown in Figure 2.2.

The composition rules defined above ensures decomposability, and if all leaf distributions are used, completeness is guaranteed. Rich SPN structures wtih fine-grained details (like CSI) can arise just from these compositions rules. To give some organization within what may sometimes seem like an highly detailed SPN structure, nodes with the same scope may be mentally grouped into layers.

Leaf nodes of the same random variables form a leaf layer, while product layers interweave with sum layers.

2.2.1 Formal Definition

More formally, we may define an SPN $\mathscr{S} = (\mathscr{G}, \rho)$ over random variables X as a tuple of a rooted acyclic directed graph $\mathscr{G} = (V, E)$ and a set of categorical distributions $\rho = \{\rho_C\}$. The set of graph nodes V contains three types of nodes C:

sum node : C_s is a node over other nodes C' of the same scope

$$C_s = \sum_{C' \in \mathbf{child}(C_s)} \rho_{C_s}(C_s \to C') \times C'$$

Its scope is the same as any of its children.

$$\mathbf{scope}(C_s) = \mathbf{scope}(C') \quad \forall \quad C' \in \mathbf{child}(C_s)$$

product nodes : C_p is a node over other nodes C' of disjoint scopes

$$C_p = \prod_{C' \in \mathbf{child}(C_p)} C'$$

Its scope is the union of its children's scope.

$$\mathbf{scope}(C_p) = \bigcup_{C' \in \mathbf{child}(C_p)} \mathbf{scope}(C')$$

leaf nodes : C_l is a tractable distribution over some subset of random variables $Y \subseteq X$.
Its scope is Y.

2.3 SPN Operations

A simple but deep composition of elementary distributions preserves the ease of performing many probabilistic manipulation and calculations. If the leaf distribution can perform a query in $O(1)$ time, often we find that an SPN of those leaves can perform the same query in $O(N)$ time where N is the number of nodes in the network. I will define six probabilistic queries for which SPNs are efficient, and intuitively describe SPN's algorithm to these queries. More details and proofs can be found in [25].

2.3.1 Sampling

Definition: Draw a sample x from $p(X)$

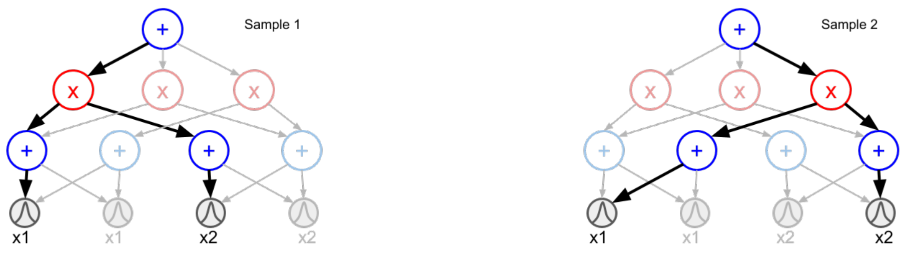


Fig. 2.3 How to sample from an SPN.

Algorithm: Ancestral sampling. We start from the root of the SPN and traverse down to the leaves. If we encounter a sum node, we stochastically descend down to one of its children according to its categorical probability. If we encounter a product node, we descend down all of its children. The recursive process then continues in ever more parallel threads until we encounter a set of leaf nodes. At that point, we independently sample observation from the leaf distributions.

Discussion: If the SPN is decomposable, it can be shown that the scopes of the leaves $\{X_l\}$ are partitions of the overall scope X. In other words, no random variable will be sample more than once. If the SPN is consistent, a random variable may be sampled more than once but it will never produce conflicting instantiations.

2.3.2 Complete observation

Definition: Compute $p(X = x_{obs})$

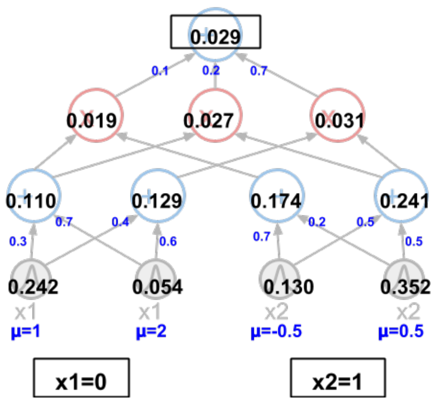
Algorithm: We start from the leaves of the SPN. All leaf distributions compute the likelihood of their scopes. Like in a computation graph, these likelihoods propagate up the SPN to the root. A sum node's activation is a weighted sum of its children's activation. A product node's activation is a product of its children's activations. See Figure 2.4a

2.3.3 Partial observation

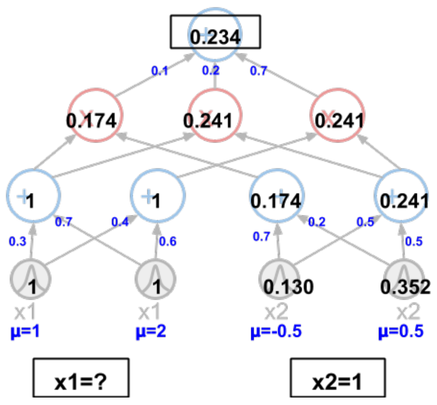
Definition: Only some random variables $X_{obs} \subseteq X$ are observed. Let's partition X into (X_{obs}, X_m).
Compute $p(X_{obs} = x_{obs}) = \int p(X_{obs} = x_{obs}, X_m = x_m) dx_m$

Algorithm: Same evaluate (upwards pass) as when computing complete observations, except now leaf nodes without observations simply set their activations to 1.

Discussion: Peharz offered a proof of this algorithm in [25]. In a nutshell, an integral of a sum is a sum of integrals, while an integral of a product of disjoint scopes is a product of integrals. The work



(a) $p(X = x)$



(b) $p(X_{obs} = x_{obs})$

Fig. 2.4 SPN operations illustrated through an example

needed to marginalize the distribution at the root of the SPN is thus reduced in polynomial time to marginalization of the leaves.

Full marginalization of a leaf, $\int p(X = x)dx$, produces 1 — the entire probability mass. Partial marginalization, $\int_a^b p(X = x)dx$, is also possible and it produces some probability mass smaller than 1.

Note that after marginalization, the marginalized leaf's activation have units of probability mass while the other (continuous) leaves' activations have units of densities. An SPN can compute mixed probability mass and densities.

2.3.4 Marginalizing

Definition: Only some random variables $X_{obs} \subseteq X$ are observed. Let's partition X into (X_{obs}, X_m).
Create a new SPN $p(X_{obs}) = \int p(X_{obs}, X_m = x_m)dx_m$

Algorithm: Prune all sum nodes and leaf nodes whose scopes are purely in X_m. Renormalize sum node weights, and remove redundant sum nodes and product nodes.

Discussion: Marginalizing before computing complete evidence is entirely equivalent to computing the partial evidence. Yet I considered this to be a separate probabilistic query to highlight SPNs' unique ability to compile a simpler version of itself.

2.3.5 Conditioning

Definition: Only some random variables $X_{obs} \subseteq X$ are observed. Let's partition X into (X_{obs}, X_m)
Create a new SPN of the posterior: $p(X_m|X_{obs} = x_{obs})$.

Algorithm: Calculate the likelihood of partial observation but this time, record the incoming activations to every sum node. On downwards pass, renormalize the weighted activations at every sum node such that they sum to 1, and use the activations as the new weights. Prune the observed variables away, then remove redundant nodes from the SPN.

Discussion: Again, computing the posterior $p(X_m = x_m | X_{obs} = x_{obs})$ is entirely equivalent to computing two forward passes on the original SPN $p(X_m, X_{obs})$ and using Baye's rule:

$$p(X_m | X_{obs}) = \frac{p(X_m, X_{obs})}{p(X_{obs})} \tag{2.1}$$

Yet I considered this as a separate probabilistic query to highlight SPN's unique ability to compile a simpler version of itself.

2.3.6 Argmax

Definition: Compute $\arg\max_X p(X,Y)$,
where Y are the latent variables associated with the SPN's sum nodes.

Algorithm: Greedy Algorithm. Convert every sum node to a max node. We will start from the root of the Max Product Network and descend to the leaves. If we encounter a sum node, we descend down the child with the highest weight. If we encounter a product node, we descend all children. At a leaf node, we instantiate the random variable at the mode of its distribution.

Discussion: This inference scenario is called Most Probable Explanation (MPE) because we optimize over X and Y but discard the answer to Y. This is in contrast to an alternative inference scenario called MAP, where we marginalize the uninteresting variables away before performing optimization.

$$\text{(MPE inference)} \qquad \arg\max_X \quad p(X,Y) \tag{2.2}$$

$$\text{(MAP inference)} \qquad \arg\max_X \quad p(X) \quad = \quad \arg\max_X \quad \sum_y p(X, Y = y) \tag{2.3}$$

This MPE algorithm is only correct for *selective* SPN [28]. Unless the leaves perfectly partition the support of any random variable, generalized SPNs are not *selective*. And unfortunately, in general, MPE and MAP inference in generalized SPN is NP-hard.

Often in literature [32], SPN's inference capabilities are demonstrated on an image completion problem. That is, we want the MAP inference.

$$\arg\max_{X_m} \quad p(X_m | X_{obs}) \quad = \quad \arg\max_{X_m} \quad \sum_y p(X_m, Y = y | X_{obs}) \tag{2.4}$$

This is done by fusing the argmax step and conditioning step (Section 2.3.5) without producing a new intermediate SPN. Interestingly, the combined forward and backwards operation is very similar to the Viterbi algorithm.

Alas, there is no efficient MAP (or even MPE) inference algorithm for generalized SPNs. So many researchers employ this MPE algorithm as an approximation to the true MAP results.

2.4 Learning SPN

2.4.1 Parameters

In the seminal paper, Poon and Domingos proposed [32] two ways to learn sum nodes' probabilities and leaf nodes' parameters. First is gradient-based optimization of the marginal likelihood.

$$\frac{d}{dw}p(X) \tag{2.5}$$

Second is a hard EM algorithm.

$$\text{(hard) E-Step} : Y^t \leftarrow \arg\max_{Y} \quad p(Y|w^{t-1}, X) \tag{2.6}$$

$$\text{M-Step} : w^t \leftarrow \arg\max_{w} \quad p(X|w^{t-1}, Y^t) \tag{2.7}$$

Both methods can be described as *generative* training because we optimize the generative mechanism to fit the observed data. On the other hand, Gens and Domingos proposed [14] a *discriminative* gradient ascent framework for training SPNs. The SPN is optimized to predict a label variable given the input variable X. This label can be the class of the image, or even a hierarchical description of the image.

Both papers noticed difficulties with training SPN via gradient based methods and attributed it to the gradient diffusion problem. Consequently, Gens and Domingos proposed [14] a hard gradient-ascent where the learning signal is transmitted only to a subtree of the SPN. They derive the gradient update formula for both the marginal inference and MPE inference case.

Peharz later discovered [28] that the MPE inference algorithm is only correct for selective SPN, so the original EM algorithm is only correct for selective SPN. A correct soft EM algorithm was derived for generalized SPNs.

2.4.2 Structure

Unlike the structure of neural networks, the structure of the SPN has probabilistic semantics (like CI and CSI). The complexity of the overall SPN distribution is predominantly encoded in the structure instead of the sum node weights. As such, structure learning is important for SPN.

The seminal paper [32] indirectly learnt an SPN structure by first designing a large dense SPN, learning the probabilities with a sparse prior, and finally pruning the structure. This is a simple approach towards structure learning because it relies on reliably known techniques, and keeps the maximum complexity of the structure controlled. More on this technique is described in the Section 2.5 below.

This method, however, is severely limited by the original dense SPN — it cannot learn a structure that is not a sub-network of the dense SPN. As a remedy, Dennis and Ventura first proposed [11] a method to construct an SPN structure purely from data. Starting from a data matrix, it performs clustering on examples to create sum nodes, and clustering on variables to create product nodes. The difficulty with clustering variables, however, is that it is hard to capture correlations.

To represent how the set of problem variables X is decomposed by clustering, Dennis and Ventura introduced [11] the concept of a “region graph". Clustering essentially builds a region graph top-down. As an alternative, Peharz [26] proposed learning a region graph from bottom up. Child regions to be “merged" together are selected by independence tests. The sum node parameters in the merged regions are learned by maximum mutual information. In doing so, the region graph was shown to be a useful abstraction of SPN structure.

At the same time, Gens and Domingos proposed [15] `learnSPN`. Starting from a data matrix, independence tests split variables to create produce nodes, and clustering methods split examples to create sum nodes. This top-down structure learner became the most prominent approach, perhaps because it elegantly reflects the interpretation of sum and product nodes as co-clustering of a data matrix.

Since then, many other structure learners have been proposed. Another prominent one is by Rooshenas and Lowd which proposed [35] `ID-SPN`. It generalized `learnSPN` by not decomposing the data matrix to single variables, but instead to groups of variables whose structures are then learnt by ACMN [22], which is a method to learn the structures and parameters of a tractable discrete variable Markov network.

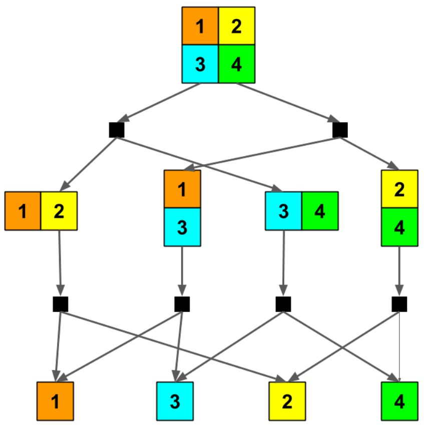
2.5 Poon Domingos architecture

2.5.1 Description

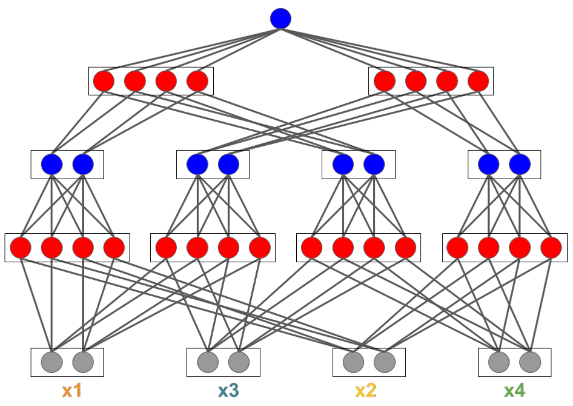
The novelty in the seminal paper [32]'s structure learner lies in generating a dense SPN structure for image data. We will call this structure the Poon-Domingos (PD) architecture.

First, a region graph is constructed from top-down. (See Figure 2.5a, black squares are partitions. Group of coloured squares are regions.)

A region graph is an acyclic directed graph of **regions** and **partitions**. A region is a local rectangular patch of the image, while a partition is a vertical or horizontal split of a region. Every partition has one parent region and two child regions. As an example, Figure 2.6 shows two possible partitions of a 2×3 region. In general, an $m \times n$ region has $m - 1$ possible horizontal partitions and $n - 1$ possible vertical partitions.

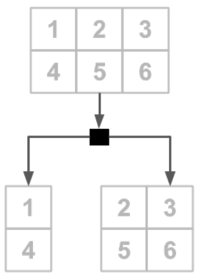


(a) Region Graph

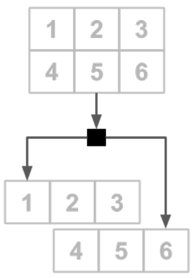


(b) SPN constructed from Region Graph

Fig. 2.5 How to construct PD SPN from a region graph



(a) Vertical partition



(b) Horizontal partition

Fig. 2.6 Partitions of a region

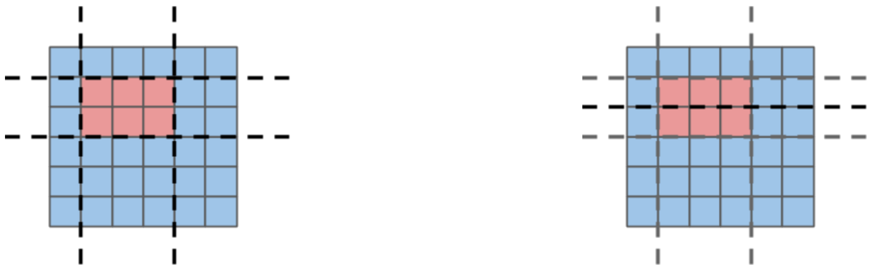

(a) A region (b) A partition

Fig. 2.7 How to count size of PD architecture

Starting from the root region (the entire image), we build a region graph by recursively adding all partitions of leaf regions into the graph, until the leaves are 1×1 regions. If a child region of a partition already exists in the region graph, that child is shared between the multiple partition parents. By making every region unique, the region graph shares substructures and grows deep without exponential blowup in size.

However, sometimes we may not need to add every possible partition to the region graph. We may first *coarsegrain* the image before partitioning. In other words, we only add every c^{th} horizontal and vertical partition of a region, resulting in a region graph with $c \times c$ (or sometimes smaller) leaf regions. Partitioning then continues, at perhaps another smaller coarsegrain lengthscale c_2, until only 1×1 leaf regions are left.

Next, a dense SPN structure is built from the region graph. (See Figure 2.5b. Blue circles are sum nodes. Red circles are product nodes)

Every non-leaf region corresponds to a layer of K_{sum} sum nodes, while every leaf region corresponds to a layer of K_{leaf} leaf nodes. A partition also corresponds to a layer of product nodes, but this time, the number of product nodes is the product of the number of nodes in its child regions. If both child regions are non-leaf layers, then there are K_{sum}^2 product nodes; if one is a leaf region, then there are $K_{sum}K_{leaf}$ product nodes; or if both are leaf regions, then there are K_{leaf}^2. The product layer is densely connected to its parent and child layers. At the root region, we use only one sum or product node, representing the overall distribution of the SPN.

2.5.2 Network size

The PD architecture is defined recursively on local image patches. To get a handle on the total size of the architecture constructed, I will explain how to count the number of regions and partitions. In the discussion below, I will ignore coarsegraining.

In an m by n image, a region is a rectangle that fits within the image. The number of such rectangles is then the number of ways to pick two row and two column boundaries.

$$\#\text{ regions} = \binom{m+1}{2}\binom{n+1}{2} = \frac{m(m+1)n(n+1)}{4} \tag{2.8}$$

A partition of a region must lie between the column (or row) boundaries. So the number of partitions is the number of ways to pick two row (or column) and three column (or row) boundaries.

$$\#\text{ partitions} = \binom{m+1}{3}\binom{n+1}{2} + \binom{m+1}{2}\binom{n+1}{3} = \frac{m(m+1)n(n+1)(m+n-2)}{12} \tag{2.9}$$

2.5.3 Equivalence to tensorial decomposition

The regularities within PD architecture suggests that there may be a more abstract description of the dense SPN. Indeed, though not often mentioned in literature, PD architecture can be interpreted as a tensorial mixture model[36]. In the tensorial formulation, the sums and products are implicit within the matrix multiplications. To show this equivalence, we will describe PD architecture more formally.

Every region is labelled by its scope X. Every sum (or leaf) node in each region is indexed with integer $k \in [0,K]$. So the k^{th} probability distribution of scope X is denoted as

$$p^k(X) \tag{2.10}$$

Every partition children of a node is indexed with an integer $n \in [0,N]$. Additionally, let's define a convenience function $\mathbf{child}(X,n,L/R)$ that maps a parent region X, partition index n, and binary argument (L or R) to a child region. Using Figure 2.6 as an example:

$$\begin{aligned}
\mathbf{child}(\{1..6\},0,L) &= \{1,4\}\\
\mathbf{child}(\{1..6\},0,R) &= \{2,3,5,6\}\\
\mathbf{child}(\{1..6\},1,L) &= \{1,2,3\}\\
\mathbf{child}(\{1..6\},1,R) &= \{4,5,6\}
\end{aligned}$$

Every non leaf region X has two probability tensors — A^k_n and $B^n_{k1,k2}$. A^k_n is the mixing probability of the n^{th} partition for the k^{th} sum node in region X. Having selected the n^{th} partition, $B^n_{k1,k2}$ is the mixing probability of the joint pair of k_1^{th} node of the left child region and the k_2^{th} node of the right child region.

So in Einstein's summation notation, the probability distribution represented by the k^{th} sum node of region X is recursively defined as: (implicit sum over n, k_1, k_2)

$$p^k(X) = A^k_n \quad B^n_{k1,k2} \quad p^{k1}(\mathbf{child}(X,n,L)) \quad p^{k2}(\mathbf{child}(X,n,R)) \tag{2.11}$$

Chapter 3

Variational Autoencoder

"A VAE is an easy way to learn a high dimensional $p(X|Z)$."

Variational Autoencoders (VAEs) are an active area of research in machine learning. This chapter only describes the basic concepts of VAEs most relevant to this thesis.

3.1 What is a VAE?

At its heart, a variational autoencoder (VAE) is a density network, which is a continuous latent variable model proposed [23] by Mackay. A density network defines a prior over the latent variable $p_\theta(Z)$, and a conditional distribution $p_\theta(X|Z)$ implemented by a neural network. The distribution of interest is the marginal of the joint probability distribution.

$$p(X) = \int p_\theta(X|Z=z)p_\theta(Z=z)dz \tag{3.1}$$

This integral (Eq 3.1) scales exponentially with the dimension of Z. If Z is high dimensional, learning of parameters θ through gradients methods is intractable because every gradient update requires an expensive marginalization. Parameters θ also cannot be learnt by EM algorithm because the conditional probability distribution $p_\theta(X|Z)$ is implemented by a neural network and is thus difficult to invert.

In Mackay's paper [23], Z was kept low dimensional and parameters θ were still learnable by calculating gradients of a Monte Carlo approximation of Eq 3.1. If Z is high dimensional, Mackay suggested using Hamiltonian Monte Carlo sampling of the posterior to generate importance weighted samples for the Monte Carlo integral.

Fast forward two decade, two seminal papers [21, 34] proposed a way to avoid the exponential scaling of Eq 3.1. With an inference network $q_\phi(Z|X=x)$ that approximates the true posterior $p_\theta(Z|X=x)$ for all $=x$, importance weighted samples z can be cheaply generated for the integral. Both the density network and the inference network can be be jointly trained if the evidence lower bound

(ELBO) was used as an objective. To allow the gradients to flow pass the sampling operation and reach the inference network, both papers proposed a reparameterization trick (stochastic backpropagation).

Traditionally in variational inference, the parameters of the variational distribution $q_\phi(Z|X=x_i)$ has to be learnt individually for every single datapoint. For example if $q_\phi(Z|X=x_i)$ is a Gaussian, $\phi = \{\{\mu_i, \Sigma_i\}...\}$. Instead for a VAE, a neural network learns a function to predicts the variational parameters for all possible x_i.

$$q_\phi(Z|X=x_i) = \mathscr{N}(Z|\mu_i, \Sigma_i) \quad , \quad \mu_i, \Sigma_i = f_\phi(x_i) \tag{3.2}$$

VAEs are thus a method to combine probabilistic reasoning with the flexibility of deep learning. From the lens of traditional variational inference [43], the cost of inference is said to be “amortized" over all the training examples. Thus learning VAEs with is scalable to high dimensions of Z and large amounts of data $\{x_i\}$.

Since then, VAEs have attracted strong interest in the machine learning community: first, for the density network's potential to generate new observations similar to the examples it has been trained on, and second, for its inference network's potential to find compact and possibly interpretable representations of high dimensional observations.

3.2 Deriving the ELBO

The ELBO $\mathscr{L}$ is a variational lower bound to the marginal likelihood because its error is the KL divergence between the approximate posterior and the true marginal likelihood.

In classical variational bayes, we want to do full Bayesian prediction which requires a posterior over some unknowns weights given data.

$$\int f(y|W) \underbrace{p(W|\mathscr{D})}_{\text{needed}} dw \tag{3.3}$$

While $p(W|\mathscr{D})$ is often intractable, we can approximate it with some tractable function $q_\phi(W)$ parameterized by ϕ. To select the best q, we optimize the KL divergence between q_ϕ and $p(W|\mathscr{D})$.

$$\phi^* = \arg\min_\phi KL[q_\phi(W)||p(W|\mathscr{D})] \tag{3.4}$$

However, since $p(w|\mathscr{D})$ is intractable, we rewrite the KL as:

$$KL[q_\phi(W)||p(W|\mathscr{D})] = \int q_\phi(W=w) \log \frac{q_\phi(W=w)}{p(W=w,\mathscr{D})} dw + \int q_\phi(W=w) \log p(\mathscr{D}) dw \tag{3.5}$$

$$= -\mathscr{L} + \log p(\mathscr{D}) \tag{3.6}$$

Since the KL divergence is always positive, $\mathscr{L}$ is a lower bound of $\log p(\mathscr{D})$. To minimize the KL, we maximize $\mathscr{L}$.

This derivation, done in the context of Bayesian prediction for a posterior over weights, can equivalently be done in the context of density networks for a posterior over latent variables.

$$p(X) = \int p_\theta(X, Z = z) dz \tag{3.7}$$

$$= \int \frac{p_\theta(X, Z)}{p(Z = z|X)} \underbrace{p(Z = z|X)}_{\text{needed}} dz \tag{3.8}$$

$$\approx \int \underbrace{\frac{p_\theta(X, Z)}{q_\phi(Z = z|X)}}_{\text{approx}} \underbrace{q_\phi(Z = z|X)}_{\text{approx}} dz \tag{3.9}$$

So

$$KL[q_\phi(Z|X)||p(Z|X)] = \int q_\phi(Z = z|X) \log \frac{q_\phi(Z = z|X)}{p(X, Z = z)} dz + \int q_\phi(Z = z|X) \log p(X) dz \tag{3.10}$$

$$= -\mathscr{L} + \log p(X) \tag{3.11}$$

$\mathscr{L}$ can then be estimated through importance weighted Monte Carlo integration with cheaply generated samples.

$$\mathscr{L} = \int q_\phi(Z = z|X) \log \frac{p(X, Z = z)}{q_\phi(Z = z|X)} dz \tag{3.12}$$

$$\approx \frac{1}{N} \sum_n \log \frac{p(X, Z = z_n)}{q_\phi(Z = z_n|X)} \quad , \quad z_n \sim q_\phi(Z|X) \tag{3.13}$$

Empirically, Kingma and Welling [21] found that N can be as few as 1 if the batch size M is large enough (>100).

3.3 Importance weighted autoencoder

There is a more direct way to derive $\mathscr{L}$ using Jensen's inequality.

$$\log p(X) = \log \int p_\theta(X, Z = z) dz \tag{3.14}$$

$$= \log \int q_\phi(Z = z|X) \frac{p_\theta(X, Z = z)}{q_\phi(Z = z|X)} dz \tag{3.15}$$

$$\geq \int q_\phi(Z = z|X) \log \frac{p_\theta(X, Z = z)}{q_\phi(Z = z|X)} dz = \mathscr{L} \tag{3.16}$$

Now, let $F(z) = \frac{p_\theta(X, Z=z)}{q_\phi(Z=z|X)}$. How wide $F(z)$ varies for different z determines how tight the variational bound is. If $F(z)$ is almost constant, the inequality becomes exact. This insight motivates constructing tighter bounds by reducing the variance of $F(z)$. A simple way to do so is to increase the dimension of the problem.

$$F(z) = \frac{p_\theta(X, Z = z)}{q_\phi(Z = z|X)} \quad \longrightarrow \quad F(z_1, ..., z_K) = \frac{1}{K}\sum_{k=1}^{K} \frac{p_\theta(X, Z = z_k)}{q_\phi(Z = z_k|X)} \tag{3.17}$$

In the larger space of integration, the integrand looks smoother. This is the smoothing trick at the heart of the importance weighted autoencoder (IWAE) by Burda et al [5].

$$\log p(X) = \log \int q_\phi(Z = z|X)F(z)dz \tag{3.18}$$

$$= \log \int \prod_{k=1}^{K} q_\phi(Z = z_k|X)F(z_1, ..., z_K)dz_1...dz_K \tag{3.19}$$

$$\geq \int \prod_{k=1}^{K} q_\phi(Z = z_k|X) \log F(z_1, ..., z_K)dz_1...dz_K = \mathscr{L}_K \tag{3.20}$$

By shrinking the variance of $F(z)$ before the application of Jensen's inequality, Burda et al [5] proved that ever tighter bounds can be constructed.

$$\mathscr{L} = \mathscr{L}_1 \leq \mathscr{L}_2 \leq \mathscr{L}_3 ... \leq \mathscr{L}_K \tag{3.21}$$

$\mathscr{L}_K$ can then be estimated by importance weighted Monte Carlo integration with K times more samples. N is again usually set to 1.

$$\mathscr{L}_K \approx \frac{1}{N}\sum_n \log F(z_{n1}, ..., z_{nK}) \quad , \quad z_{nk} \sim q_\phi(Z|X) \tag{3.22}$$

Since the variance of F has decreased, the Monte Carlo error in estimating F also decreased.

There are, however, tradeoffs to using IWAE. First, the expression $\log F(z_1, ..., z_K)$ no longer allows us to break the KL divergence expression into a reconstruction error term and a regularization loss term. In the case of VAE with Gaussian approximate posterior and prior, the regularization loss term can be analytically calculated, a trick with Kingma and Welling exploited to reduce the variance of the gradient estimate. Second, even though the variance of the gradient estimate reduces with increasing K, the magnitude of the gradient signal entering the inference network decreases faster, thus increasing the signal to noise ratio [33].

Chapter 4

SP-VAE

“SP-VAE maintains tractability by keeping its VAE leaves small."

In this chapter, I will define SP-VAE, a hybrid mix of SPNs and VAE investigated in this thesis.

4.1 VAEs as SPN leaves

The SPN operations described in Chapter 2 requires SPN leaves to be able to produce 1) independent samples, 2) likelihood of observation and 3) the argmax of the distribution. Ideally, since there are many leaves in an SPN, each leaf should also be computationally cheap and use little memory.

VAEs do not seem to fit these requirements, where its most immediate disqualification seems to be its canonically intractable likelihood. But this intractability is not fatal. Although VAEs can only compute a lower bound of the log likelihood, ever tighter bounds can be constructed using IWAE [5]. Additionally, if the VAEs are sufficiently small, then the exact likelihood can be sufficiently approximated by Monte Carlo integration.

VAE leaves have benefits over traditional SPN leaves: First, they can represent significantly more complex distributions. A greater complexity reduces the need to learn large SPNs or even many duplicate leaves with the same scope. Second VAEs have a latent variable which opens up a possibility for SPNs to perform representation learning. And third, if this latent variable is shared between leaves, either within a scope or between scopes, perhaps better “communication" or “entanglement" between SPN leaves can be achieved with a smaller SPN.

4.2 SP-VAE operations

Like in Section 2.3, this section will define some probabilistic queries which can be answered by SP-VAE. With more complex leaves, SP-VAE may not be as agile and flexible as SPNs is with traditional leaves. For example, it cannot compile a simpler version of itself after conditioning or marginalizing. There also may not be a case of SP-VAE where an exact argmax algorithm exists (although it may still be worth studying if Monte Carlo maximum likelihood algorithms will work)

4.2.1 Sampling

Definition: Draw independent samples from $p(X)$

Algorithm: Similar as in SPNs. The only difference is that when we reach a VAE leaf, we first sample the latent variable $z \sim p(Z)$, pass z through the decoder, and finally sample from the output distribution of the decoder $x \sim p(X|Z=z)$

4.2.2 Complete Observation

Definition: Compute $p(X = x_{obs})$

Algorithm: The likelihood is canonically intractable, so the importance weighted ELBO is used as a lower bound. After, passing x_{obs} through the VAE's encoder, K samples of z are drawn from the approximate posterior distribution $z_i \sim p(Z|X = x_{obs})$. These samples z_i pass through the decoder to derive K distribution parameters. We can now compute the importance weighted ELBO according to:

$$\mathscr{L} = \log \left[\frac{1}{k} \sum_{i=1}^{k} \frac{p(X = x_{obs}, Z = z_i)}{q(Z = z_i | X = x_{obs})} \right] \tag{4.1}$$

Alternatively, if the VAE latent variables are small enough, we can perform the integration through Monte Carlo integration:

$$\begin{aligned} p(X = x_{obs}) &= \int p(X = x_{obs}, Z = z) dz && (4.2) \\ &\approx \frac{1}{N} \sum_{k=1}^{N} p(X = x_{obs} | Z = z_k) \quad , \quad z_k \sim p(Z) && (4.3) \end{aligned}$$

Leaf activations then propagate up to the sums and products just as in traditional SPNs.

4.2.3 Partial Observation

Definition: Only some random variables $X_{obs} \subseteq X$ are observed. Let's partition X into (X_{obs}, X_m). Compute $p(X_{obs} = x_{obs}) = \int p(X_{obs} = x_{obs}, X_m = x_m) dx_m$

Algorithm: Like when computing likelihood of complete observations, VAE leaves with complete observations still compute their ELBO or Monte Carlo estimate of their likelihood. The only difference is that VAE leaves with partial or no observations need more tricks.

Case 1: If the entire VAE's scope is unobserved, then we can automatically set the likelihood to 1

Case 2: If only one variable is missing, and that variable has closed support between a and b, we can do Monte Carlo integration through the complete observation. This method uses the VAE leaf's inference network.

$$p(X_{obs} = x_{obs}) = \int p(X_{obs} = x_{obs}, X_m = x_m) dx_m \tag{4.4}$$

$$\approx \frac{|b-a|}{N} \sum_{k=1}^{N} p(X_{obs} = x_{obs}, X_m = x_m^{(k)}) \quad , \quad x_m^{(k)} \sim uniform[a,b] \tag{4.5}$$

Case 3: Alternatively, we may avoids use of the inference network and directly integrate the latent variable. This method assumes that the VAE leaf's latent variable is low dimensional, and that the output distribution is diagonal or easily marginalizable.

$$p(X_{obs} = x_{obs}) = \int p(X_{obs} = x_{obs}, Z = z) dz \tag{4.6}$$

$$\approx \frac{1}{N} \sum_{k=1}^{N} p(X_{obs} = x_{obs} | Z = z_k) \quad , \quad z_k \sim p(Z) \tag{4.7}$$

Discussion: This operation motivates the development of future latent variable inference methods that can accept inputs with missing features. If all of X is missing, inference should produce the prior over Z. If more of X is observed, the uncertainty in Z should decrease.

4.2.4 Imputation

Definition: Draw a sample from $p(X,Y,Z)$ with high likelihood,

where Y are the latent variables associated with the SPN's sum nodes, and Z the latent variables of the VAE leaves.

Algorithm: Same greedy algorithm as in SPN of converting sum nodes to max nodes. The only difference is that when we have descended into a VAE, we perform approximate Gibbs sampling as suggested by Rezende et al [34]:

1. Randomly initialize an x
2. Pass x through the encoder.
3. Sample z from the posterior $p(Z|X = z)$ and pass that through the decoder.
4. Sample an x' from the decoder $p(X|Z = z)$
5. Feed that x' back into encoder and repeat until converge.

Discussion: Ideally, we would like to have a tractable argmax algorithm for SP-VAE. However, argmax is hard for a neural network as it is a highly nonconvex function. An easier task would be for SP-VAE to produce a sample x “near" one of its modes. In a VAE, this imputation task can be achieved by Gibbs sampling as described above. Samples drawn after the Gibbs sampler has converged should lie near some mode of the VAE distribution $p(X)$

Now, Gibbs sampling requires a posterior, but only an approximate posterior is available through the inference network. Rezende et al [34] proved that if the marginal error of other approximate posterior is bounded, then the error in the weak fixed point of the Markov Chain is also bounded by a similar amount.

Gibbs sampling has its own set of difficulties, the most well-known of which is the difficulty in judging if the Gibbs sampler has converged. In the context of SP-VAE, it is not yet clear if there are any additional difficulties when performing Gibbs sampling on many tiny VAEs.

As mentioned in Section 2.3.6, the argmax of the conditional $p(X_m|X_{obs})$ is the ideal approach to solving image completion problem. It is difficult to condition the VAE distribution on partially observed data. However, it is still possible for Gibbs sampling to sample

$$p(X_m, X_{obs} = x_{obs}, Z) \tag{4.8}$$

We will follow the same Gibbs sampling procedure, except now, the observed variables are reset to the observed values before passing x' is passed into the encoder.

4.3 Learning SP-VAE

4.3.1 Structure Learning

A dominant fraction of SP-VAE’s modelling power lies in the leaves instead of the structure of the SPN. So the structure of the SPN is less important than in pure SPNs.

As a default used in this thesis, PD architecture coarsegrained at lengthscale c could be used, resulting in VAE leaves over $c \times c$ patches (or smaller). This only require choices of hyperparameters and no structure learning takes place. Alternative, instead of coarsegraining, random partitions, as done in [31], could be used.

If the rectangular blocky nature of PD architecture is undesirable, `ID-SPN` structure learner is a possible alternative. The only difference now is that a VAE implements the multivariate leaves instead of a Markov network.

4.3.2 Parameter Learning

SPNs admit EM and gradient-based learning algorithms. VAEs, however, can only rely on stochastic gradient descent. So gradient-based methods are the simplest way to jointly train SP-VAE.

Chapter 5

Methodology

“Make comprehensive and fair comparisons."

The central aim of this thesis is to understand how does SP-VAE’s performance compare against SPNs and VAEs. This aim motivates and guides the experiments in this thesis. In this chapter, I state the method of evaluation, summarize considerations made to ensure valid and rigorous investigations, and motivate the objectives of experiments.

5.1 Choice of evaluation metric

How do we know how well a model performed at density estimation if the true distribution is unknown (and perhaps even undefined)? Besides evaluating the model through downstream applications, there are two popular approaches to evaluation used in literature: First is via the likelihood on a test set. The second approach is sample fidelity. This can be qualitatively done through visual inspection, 1-NN or even through a visual Turing test, but they are subjective and time consuming.

Some recent research [37] have suggested that test likelihoods and sample fidelity are independent indicators of the quality of fit. Different sample fidelity metrics may even favor different models. The search for a good evaluation metric for generative models is still an active area of research.

In light of this, I choose to use the test likelihood as an evaluation metric, because first, it is an established methodology in literature, and second because it can generalize to non-image data. More specifically, I monitor a model’s performance on a test set and report the best test performance after a sufficiently long period of training. Admittedly, reporting best test likelihood is not standard methodology; training progress is typically measured on a validation set instead of a test set. However, this may have been necessary in this thesis because some CALTECH101 datasets are too small to afford a validation set. In effect, I would be making and comparing optimistic measures of a model’s capacity.

Another difficulty arises because VAEs computes an ELBO instead of a likelihood. I chose nonetheless to compare ELBOs and log likelihoods on equal footing, relying on the assumption that the variational gaps are small compared to differences due to the model architecture. Note that in this thesis, I will use the term “evidence" to ambiguously refer to either log likelihoods or ELBOs.

5.2 Choice of Dataset

The main evaluation dataset used in this thesis is the CALTECH101 image dataset. But MNIST handwritten digits, and cropped Street View House Numbers (SVHN) are also used in smaller scale experiments.

5.2.1 MNIST

MNIST is a classic image dataset of greyscale handwritten digits. It consists of 60000 training and 10000 test images. Each image is 28 x 28 pixels. Most of the image’s background is black (pixel value 0) while the digit is bright.

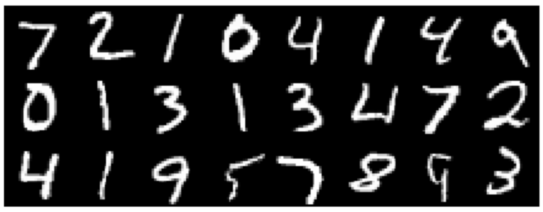


Fig. 5.1 Samples from MNIST dataset

5.2.2 CALTECH101

CALTECH101 is a collection of 102 datasets of various object categories ranging from accordions to yin yangs. This is a challenging dataset for density estimation for a few reasons: First, the size of each category’s dataset varies widely, ranging from 800 images (`airplanes` category) to 31 images (`inline_skate` category). Second, most images even within the same category do not have the same dimensions. The number of rows ranges from 92 to 3999 and number of columns range from 80 to 3481.

Since our models require fixed image dimensions, the images need to be center padded to a square shape. The images are colored as well and need to be converted to greyscale. [32] has already done these and additionally resized the images down to 100 x 100 pixels. Using their processed CALTECH101 dataset, I further shrunk the images to 25 x 25 pixels. Finally, I split each category’s dataset into test/train sets with 33.3% / 66.6% probability.

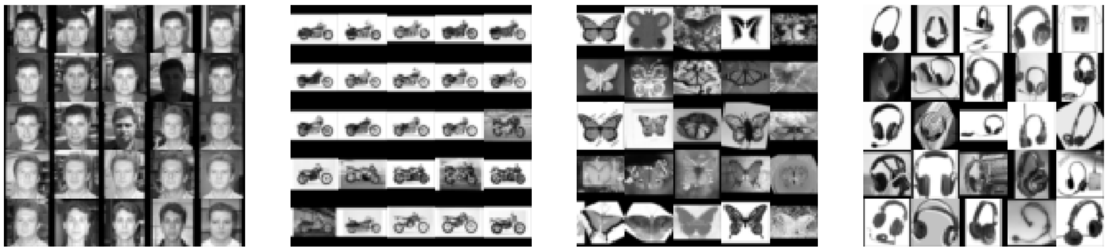


Fig. 5.2 Samples from CALTECH101 dataset

5.2.3 SVHN

SVHN is a dataset of digits like MNIST, but it is significantly more challenging for multiple reasons: Each image is larger (32 x 32 pixels), and colored. The digits may be rotated, blurred, of different scales, aspect ratios and styles. The background is textured, while additional circles or squares may frame the digit. The digit's contrast against the background may be poor, or even inverted. Most challengingly, there may be distracting digits to the left or right of the digit in the center of the image. There are 73257 training and 26032 test images.

I greyscaled the images using the luminosity kernel (0.299, 0.587, 0.114)

Fig. 5.3 Samples from SVHN dataset

5.3 Processing for Distribution Type

A dataset is interpreted as a collection of independent observations of a random multivariate variable (an image in this case). I considered the case where the random variable could be binary, continuous, or categorical, and built models based on Bernoulli, Gaussian, and Binomial distributions accordingly. The datasets should ideally reflect the discrete or continuous nature of the random variable considered. Otherwise, pathologies may occur like exploding densities when fitting continuous distributions onto a discrete dataset, and zero probability masses when fitting discrete distributions on a continuous dataset.

5.3.1 Binary random variable

MNIST dataset is easy to binarize because of the clear distinction between digit and background. I calculated the mean pixel value of each image and used it as a global threshold of binarization.

CALTECH101 is harder to binarize because it is a collection of natural images. Using global thresholds leads to large monotone patches, rendering the image object unrecognizable. An alternative of dithering based methods also leads to poor results due to the small size of the images. In the end, I chose to use adaptive Gaussian thresholds with blocksize of 3. This binarization method retains fine grained image details (like the eyes of a face and texture of background), which are precisely the challenge of natural images for density models.

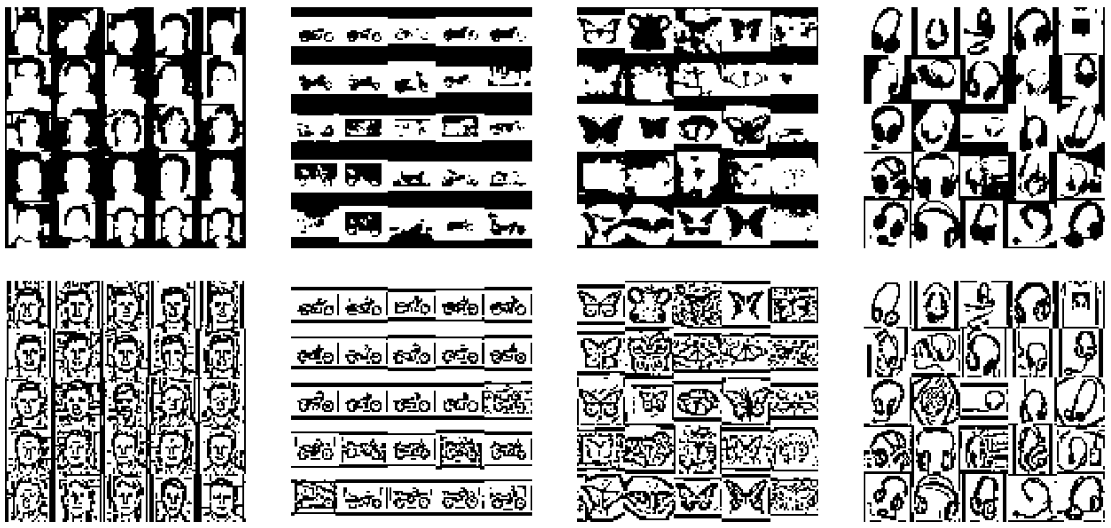
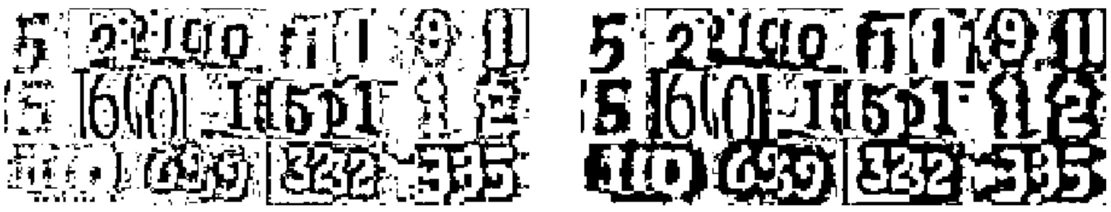


Fig. 5.4 Two different ways of binarizing CALTECH101 images
Top row: global threshold set by mean of image (reject)
Bottom row: adaptive Gaussian threshold; blocksize = 3 (accept)

SVHN is also hard to binarize for the same reasons as CALTECH101. In the end, I used an adaptive Gaussian thresholds with blocksize of 15.



(a) Blocksize = 5 (Reject) (b) Blocksize = 15 (Accept)

Fig. 5.5 Binarizing SVHN with adaptive Gaussian thresholds

5.3.2 Continuous random variable

Assuming pixel values are `uint8` integers ranging from 0 to 255, standard normal random noise is added to every pixel of every image in all datasets. If pixel values range from 0 to 1, Gaussian noise is scaled by 1/255. Pixel values exceeding the minimum and maximum are clipped.

5.3.3 Categorical random variable

Raw pixel values take integer values between 0 and 255. So they are already discrete in nature and no further processing is needed.

5.4 Controls for Fair Model Comparison

SPNs, SP-VAE, and VAEs are different model architectures, each with its own performance tradeoffs, training behaviors, and scaling properties. In order to make meaningful comparisons between the model architectures' performances, ideally we should control for other factors which may affect the model's performance. Differences in model complexity or training efficiency can then be attributed purely (or at least with more certainty) to differences in model architectures.

5.4.1 Model Size

The first factor we controlled for is “model size". As a heuristic, model size is measured by the number of parameters in the model. This may not be a perfect measure of model size because the number of parameters may not reflect the actual degrees of freedom in the model, and not all degree of freedoms are equivalent. Nonetheless, the number of parameters is a useful gauge because it can be measured across a diverse range of parametric models, and is intuitively understood by the machine learning community. And since model size is not well defined, control for it cannot be exact anyway.

5.4.2 Hyperparameters

The second factor we controlled for is hyperparameters. Since we want to compare the *best* performance of one architecture against another, we need to tune hyperparameters to their optimal for each dataset. But this is a resource intensive process made even worse by the fact that we do it by hand. As such this thesis cannot guarantee that the hyperparameters found are optimal.

In the process of manual tuning, we found that performance is most sensitive to learning rate and batch size. This is consistent with observations by other machine learning practicioners.[17]. Concretely, we found that a smaller learning rate tends to give better performance. However, it comes at a cost of longer training times. So the optimal learning rate depends also on amount of training time we are willing to tolerate.

Model Code	Distribution Type	Model Description
M1	Bernoulli	Deep SPN
M2		Shallow SPN
M3		SP-VAE
M4		VAE
M5	Gaussian	Deep SPN
M6		Shallow SPN
M7		SP-VAE
M8		VAE
M9	Binomial	Deep SPN
M10		Shallow SPN
M11		SP-VAE
M12		VAE

Table 5.1 Color codes for architectures to be evaluated

5.5 Choice of models

5.5.1 Model Code

There are four main architectures evaluated in this thesis. To be comprehensive, they are evaluated when implemented as three distribution types — Bernoulli, Gaussian, and binomial

1. **deepSPN** : A PD-SPN, coarsegrained at lengthscale c, that decomposes an image down to individual pixels. Univariate Bernoulli, Gaussian, or Binomial distributions used as leaves
2. **shallowSPN** : A PD-SPN, coarsegrained at lengthscale c, that decomposes an image down to $c \times c$ superpixels. c^2-dimensional diagonal Bernoulli, Gaussian, or binomial distributions are used as leaves
3. **SP-VAE** : A PD-SPN, coarsegrained at lengthscale c, that decomposes an image down to $c \times c$ superpixels. VAEs with 2-dimensional latent variables and c^2-dimensional observations are used as leaves. VAE's generative distribution may be diagonal Bernoulli, Gaussian, or binomial distributions.
4. **VAE** : A vanilla VAE whose encoder and decoder are implemented by MLPs. Prior and variational distributions are Gaussian. Generative distribution may be diagonal Bernoulli, Gaussian, or binomial distributions.

To ease comparisons, these architectures are given numerical and color codes as shown in Table 5.1.

5.5.2 Design Choices

All SPN structures, including SP-VAE, follow the PD architecture, which means that scopes are partitioned by horizontal and vertical splits in the image domain. Specifically, they follow the superpixel architecture, which means that the image dimension is a multiple of the coarsegraining level and that all leaf regions have disjoint scopes.

For the pure SPN, I initially chose to decompose the image down to individual pixels. However, even with coarsegraining and efforts at optimization, the resulting model takes too much memory and time to compute in Tensorflow. So a shallow version that decomposes an image down to the same superpixels as in SP-VAE was introduced. Its leaves are a product of univariate distributions. The original SPN is thus named deepSPN, and the latter shallowSPN.

All VAE encoders and decoders, both pure and as VAE leaves, are implemented by MLPs with 2 hidden layers. All hidden layers within a VAE have the same number nodes. All activation functions in MLPs are softplus.

If the VAE is an SPN leaf, its latent variable is two dimension regardless of the size of the leaf region. If the VAE is pure (modelling the full scope), its latent dimension is equal to $2\times$ (number of leaf regions in an equivalent SP-VAE).

We made the latent dimension of pure VAE depend on the SP-VAE it is compared to because we want to maintain equal “latent representation power" between both models. If the total number of latent dimension is equal between SP-VAE and VAE, we can imagine SP-VAE as disentangling some latent dimensions and forcing them to specialize on some superpixel of the image.

The design choices described above constrain the number of free parameters available when choosing a model topology. For a VAE, the only free parameter left is the number of nodes per hidden layer n_h. For an SPN, the free parameters are number of sum nodes per leaf region K_{leaf} and the number of sum nodes per region K_{sum}.

The coarsegraining level c seems like a free parameter but is in fact highly constrained by the dataset because we want c to be a factor of the image dimension.

Table 5.2 Coarsegraining levels for SPNs

Dataset	image dimension	choices for c	chosen c
CALTECH101	25 x 25	5	5
MNIST	28 x 28	2, 4, 7	4
SVHN	32 x 32	2, 4, 8, 16	8

5.5.3 Model Sizes

With fewer free parameters, it becomes easier to count the number of parameters in a model, and also easier to create a model with approximately the desired number of parameters.

If we further constrain $K = K_{leaf} = K_{sum}$, then the total number of parameters can easily be counted by the formula below:

Table 5.3 Model Size

Model Code	# parameters
M2/M10	$\#spn(n'_r, n'_c, K) + n'_r n'_c C^2 K$
M3/M11	$\#spn(n'_r, n'_c, K) + n'_r n'_c K \#vae(n_z, n_h, C^2)$
M4/M12	$\#vae(n_z, n_h, n_x) = 2n_h^2 + 2n_x n_h + 3n_z n_h + 4n_h + 2n_z + n_x$
M6	$\#spn(n'_r, n'_c, K) + 2n'_r n'_c C^2 K$
M7	$\#spn(n'_r, n'_c, K) + n'_r n'_c K \#vae(n_z, n_h, C^2)$
M8	$\#vae(n_z, n_h, n_x) = 2n_h^2 + 3n_x n_h + 3n_z n_h + 4n_h + 2n_z + 2n_x$

$$\#spn(n_r, n_c, K) = \#parts(n'_r, n'_c)K^3 + (n'_r + n'_c - 2)(K^2 - K^3)$$

5.6 Experiment Objectives

Many experimental objectives were defined at the start of the project. However, they were highly adapted and reprioritized as we gained new insights and faced unexpected challenges. The overall sequence of investigations is now best grouped into four experiments.

Experiment 1: Best performances (main evaluation)

Experiment 2: Learning without inference networks

Experiment 3: Data efficiency

Experiment 4: Balance between SPN and VAE

Chapter 6

Implementation

“SPNs are unnatural for TensorFlow."

In this chapter, I summarize relevant implementation hurdles, considerations made to solving them, and final design choice made if there were any.

6.1 Choice of Deep Learning Framework

Since this thesis involve VAEs, experiments have to be implemented in one of the modern deep learning libraries. TensorFlow was selected as the framework of choice because it has a well supported collection of primitive operations, an ecosystem of convenience libraries, and code examples online. In particular, TensorFlow's optimization library (`tf.train`), and, to a lesser extent, its input pipeline library (`tf.data`) and an monitoring library (`tf.summary`) made TensorFlow an attractive choice. TensorFlow's automatic handling of distributed computing (multiple CPU/GPUs) is an added bonus.

While there are many reference implementations online of VAEs in TensorFlow, there are no publicly available implementation of SPNs in TensorFlow. This is mostly because 1) SPNs were conceptualized before TensorFlow was popular, 2) most published implementations are in Java or python, and 3) SPNs is a probabilistic model with sampling operations that are not natural to TensorFlow's dataflow programming regime.

As such, besides implementing SPNs for the sake of experiments, this thesis hopes to contribute a simple TensorFlow implementation of SPNs in Keras' functional API.

6.2 Implementing SPNs in TensorFlow

6.2.1 Log Domain

Images are high dimensional, so likelihoods over them are usually on the order of -100 nats and if not more. Likelihoods computed have to be kept in the log domain to prevent underflows. Consequently, products are realised via additions and sums are realised via logsumexps.

A quick profile of relevant TensorFlow operations showed that logsumexp is the most expensive. However, this operation is unavoidable if we were to stay in the log domain. I considered the possibility of running experiments on small images with 128 bit floating point precision so that likelihoods can be computed in the linear domain without underflow. Unfortunately, TensorFlow does not support such high precision.

6.2.2 Probability Vectors

Every SPN sum node has a categorical distribution associated with it. The probability vector ρ parameterizing the categorical distribution must satisfy two constrains: $\rho_i \geq 0$ and $\sum_i \rho_i = 1$. In TensorFlow, however, maintaining these contrains is unnatural. I considered a few options: 1) renormalize the probability vector after every gradient update, 2) using Lagrange Multipliers, 3) reparameterized from the simplex to real space using the softmax.

$$\rho_i = \frac{\exp(w_i)}{\sum_k \exp(w_k)} \tag{6.1}$$

The last option is the simplest to implement and was thus chosen.

6.2.3 Regularization

The reparameterization of probability vectors introduces degeneracy. To see this, consider how $\frac{\exp(w_i)}{\sum_k \exp(w_k)} = \frac{\exp(w_i+c)}{\sum_k \exp(w_k+c)}$. There are infinitely many $\boldsymbol{w} = \boldsymbol{w}_0 + c$, where c is an arbitrary real number, corresponding to the same unique $\boldsymbol{\rho}$. In weight space, these degenerate $\boldsymbol{w}$ form “equipotential lines".

The SPN model function is agnostic to degenerate weights, so its gradients are perpendicular to these equipotential lines. As the model trains, there is no guarantee that these weights will be kept small. Regularization is desirable to prevent $\boldsymbol{w}$ from drifting into unnecessarily large values, which may cause overflow in the computation graph.

I chose to regularize probability weights using $L1$ penalty with weight decay = 0.1. This regularization scheme penalizes overconfidence of the SPN (low entropy distributions), encouraging balanced probabilities in the sum node instead of sparsity. An example to illustrate this effect is given in Appendix A

6.2.4 Computation Graph Size

A major limitation of implementing PD-SPN in TensorFlow is the $O(m^3n^3)$ growth in network size, where the image problem is $m \times n$ in dimensions. While Poon and Domingos [32] could virtualized the network structure, and logically retrieve or create a subset of nodes activation whenever needed by their hard EM algorithm, TensorFlow cannot and needs to implement the entire network as a computation graph residing in memory.

It takes TensorFlow about one hour on a modern Sandy Bridge Core i7 Intel CPU to construct a computation graph of an SPN with about 200000 nodes, decorate the graph with gradient update operations, and initialize variables. The entire TensorFlow session takes large amount of memory (~10GB) and time (~10s) to perform a single upward pass.

The core problem is that there are too many atomic operations explicitly specified in the computation graph, like additions or logsumexps between activations of tiny scopes. Operation definitions and communication overheads dominate the computations.

One possible way to reduce memory usage (and hence improve runtime) is not to make product nodes explicit in the computation graph. This leads us to the tensorial decomposition formulation of SPNs. The SPN's computation graph can be built with `log_tensordot` operations if computation is done in the log domain. Alas, TensorFlow does not support `log_tensordot` as a primitive operation. However, `log_tensordot` can still be indirectly implemented using `logsumexp` or directly as a custom C++ kernel for TensorFlow. An untested implementation of tensorial decomposition using `logsumexp` and array broadcasting tricks shows an approximately 10% to 20% reduction in computation graph size. Directly implementing the C++ kernel is left for future work.

6.3 Implementing VAEs in TensorFlow

Implementations of VAEs in TensorFlow are readily found online, so little is discussed here besides noting design choices. Importance weighted variational objective with $K = 5$ were used. Encoders and decoders were implemented with two hidden layer MLPs with $L2$ regularization and weight decay = 0.1.

6.4 Benchmark

As a guide for how to deploy various experiments, I performed rough benchmarks on different machines.

The environment used for this benchmarks are:

- TensorFlow: v1.8
- CPU (4 cores machine): i5-6200U CPU
- GPU : Tesla K40c

Experiment implementation changes frequently and actual run time may differ significantly because the computing environment may be shared with other users, and/or have many more CPU cores available. Thus these benchmark numbers are only meant to give order of magnitude estimates when allocating time for experiments.

In memory measurements, the large GraphDef size of deepSPNs clearly shows the intractability of implementing pure SPNs in TensorFlow. ShallowSPNs have a smaller GraphDef than SP-VAE because all of the former's K_{leaf} leaves can be wrapped and defined as a single K_{leaf} dimensional

Table 6.1 Benchmark of Bernoulli models on CALTECH101-0

Model	Time for 10 epoch		GraphDef size
	4 core	GPU	
deepSPN	-	61s	332MB
shallowSPN	56s	2.4s	10MB
SP-VAE	4.5s	3.5s	22MB
VAE	0.76s	0.08s	238kB

tensor. A VAE's GraphDef is one order of magnitude smaller than SP-VAE because only one VAE is defined.

In runtime measurements, deepSPNs take too much memory to be benchmarked on a 4 core CPU machine with 16GB RAM. Otherwise, all models invariably run faster on GPUs. It is curious that although SP-VAE is more complex than shallowSPN, the former still runs faster than the latter on CPU.

6.5 Training Models in TensorFlow

All models were trained using ADAM with its default hyperparameters ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$).

6.5.1 Difficulties with Optimizing Variances

The univariate Gaussian distribution has two parameters — mean μ and variance σ^2. Learning σ^2 by gradient methods is delicate for multiple reasons: First, it may shrink to zero, creating arbitrarily large probability density which disrupts maximum likelihood learning. Practitioners often handle this by placing a lower bound on σ^2 for numerical stability during training, and/or adding noise to the training data so that the optimal σ^2 is at least at the noise threshold. Second, σ^2 must be positive. While we could use constrained optimization, it is more convenient to reparameterize σ^2 onto the real line. Third, this choice of reparameterization may have significant effects on the training dynamics. For example, consider how the negative log likelihood is convex with respect to σ^{-2} but not σ^2 (Section 7.1 of [4]). Finally, the learnt σ^2 is sensitive to initialization.

To illustrate the importance of choosing a good parameterization, Figure 6.1 shows the training curves of two parameterization of σ^2. Both models (shallow SPNs with diagonal Gaussian leaves) were trained on the same dataset with the same hyperparameters. Parameterizing σ^{-2} as $10^{-7} + w_\sigma^2$ gives smoother and higher log likelihoods than parameterizing σ^2 as $10^{-7} + w_\sigma^2$.

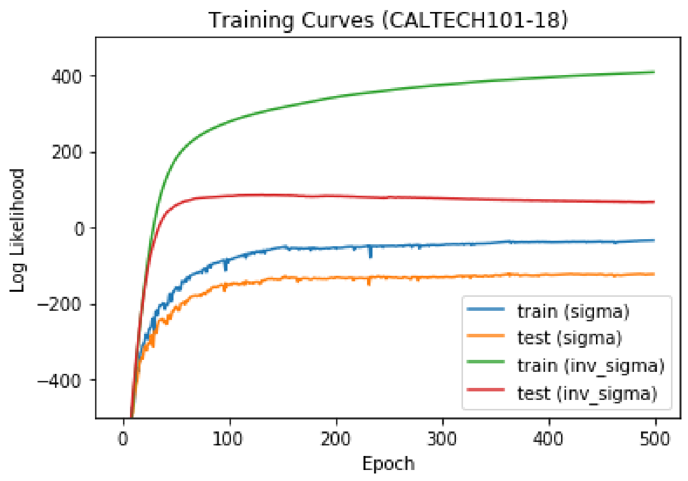


Fig. 6.1 (Best view in color) Learning of SPN with Gaussian leaves is sensitive to parameterization of σ^2. blue and orange: $\sigma^2 = 10^{-7} + w_\sigma^2$; green and red: $\sigma^{-2} = 10^{-7} + w_\sigma^2$. Model optimized w_σ using ADAM. w_σ initialized with standard normal noise.

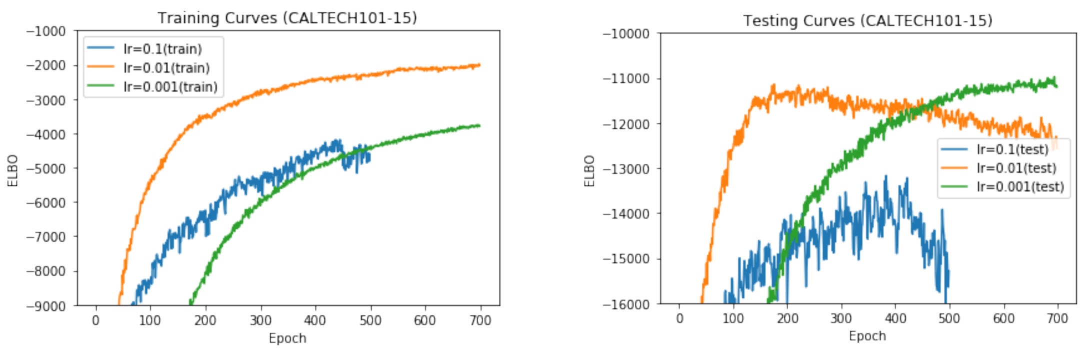


Fig. 6.2 Tuning Learning Rate on a single dataset

6.5.2 Hyperparameter Tuning: Human in the Loop

A good selection of hyperparameters can make significant difference in training speed and test outcomes. As an illustration, Figure 6.2 shows some results from tuning learning rate. A learning rate that is too large (blue) may not lead to faster convergences and in fact risks producing NaNs in the computation graph. A learning rate that is too small (green) achieves marginally higher test scores but takes too long to converge.

Realizing the impact of hyperparameters, I decided to tune them, but only manually: Starting from some default batch size (Full batch for CALTECH101, 128 for MNIST and 100 for SVHN) and learning rate (0.1 for pure SPN, 0.01 for SP-VAE, 0.001 for VAEs), a trial training run is done. I would then analyse the training and test curves, before proposing a new set of batch size, learning rate and number of epochs.

In future work, more systemic grid search of hyperparameters should be used.

Chapter 7

Experiments

“SP-VAEs are competitive, more robust, and easier to train than VAEs!"

7.1 Experiment 1: Main Evaluation

7.1.1 Motivation

“How does SP-VAE compare against SPNs and VAEs?" In this first exploratory experiment, I want to comprehensively evaluate the models on a diverse range of datasets of varying difficulty, thus CALTECH101 was chosen. Later experiments can then be developed from patterns found in this experiments.

7.1.2 Setup

SP-VAE model size was first arbitarily picked and the other models' sizes were calibrated to match it. Table 7.1, records the choice of free parameters after this calibration.

Table 7.1 Experiment 1 Model Size

Model Code	Region Graph	SPN		VAE		# Weights			% spn
	coarsegrain	sumK	leafK	n_h	n_z	total	leaves	spn	
M1 / M9	1, 5	2	2	-	-	126018	1250	124768	99.0%
M2 / M10	5	7	7	-	-	207823	4375	203448	97.9%
M3 / M11	5	2	2	25	2	143718	138950	4768	3.3%
M4 / M12	-	-	-	90	50	143285	-	-	
M5	1, 5	2	2	-	-	127268	2500	124768	98.0%
M6	5	7	7	-	-	212198	8750	203448	95.9%
M7	5	2	2	25	2	176218	171450	4768	2.7%
M8	-	-	-	80	50	176470	-	-	

7.1.3 Results and Discussions

VAEs's training curves have noise due to Monte Carlo ELBO

Figure 7.1 shows the training and test curves of all models on CALTECH101-18 dataset after hyperparameter tuning.

First notice that the blue training curves are noisy for VAEs and SP-VAE. This is because VAE's training objective is a Monte Carlo estimate of the ELBO. Even if batch noise is absent, as is the case in this experiment, there will still be noise in the training curves. Theoretically, we would expect smaller noise in SP-VAE's training curve because its VAEs are smaller. However this cannot be observed in Figure 7.1 as VAE's learning rate is an order of magnitude lower than SP-VAE (0.001 vs 0.01).

SPNs's training curves are smooth

In contrast, notice that the blue training curves of pure SPNs (Figure 7.1), are smooth and monotonic, even at learning rate of 0.1. Such smoothness is rare for a gradient-based optimization of a nonconvex function. From experience optimizing dense MLPs, even if full batch and deterministic objectives were used, such monotonicity is rarely achieved at 0.1 learning rate.

To be sure, the optimization landscape of an SPN is nonconvex. An intuitive argument for its non-convexity is given as follows: If we permute sum nodes within a scope and ensure that the weights in the parent scopes also permute to counteract that change, then the SPN will have the same loss but at a different part of weight space. In other words, any local minimum (or maximum) is necessarily replicated throughout many other parts of weight space.

Yet this monotonicity at high learning rate is a strong suggestion that the local minima in SPN's loss surface have lengthscales much greater than the step sizes taken by the gradient optimizer. The possible simplicity of SPN's loss surface is corroborated by the fact that SPNs are multilinear functions of distributions; their only nonlinearities apart from the distributions are simple multiplications. Future mathematical studies into the optimization landscape of SPNs can verify this hypothesis.

If indeed the lengthscales in the optimization landscape of SPN weights is vastly different from that of VAE weights, then vanilla gradient descent should not be used for SP-VAE. Instead, a learning rate that customizes to individual variables like ADAM is needed, or even some other algorithm involving natural gradients [2]. Or perhaps, a mix of EM algorithm and gradient methods should be used to train SP-VAE.

SPN are robust against overfitting

A pertinent worry in this thesis is if the reported test evidence is overly optimistic. The testing curves in Figure 7.1 shows that test evidence do sometimes peak and decay as training progresses. However, it seems that the test evidence decays to a smaller extent in SPNs than in VAEs.

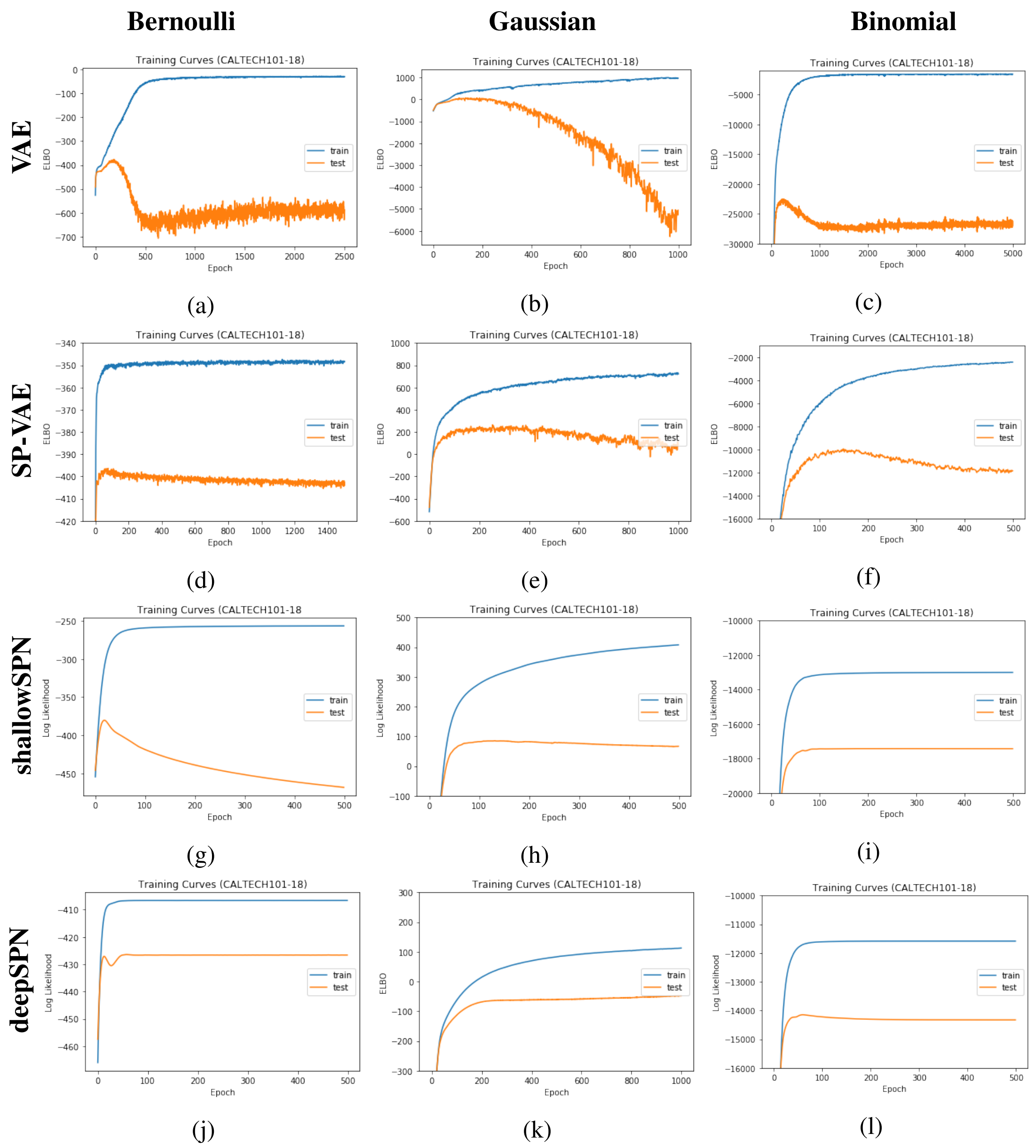


Fig. 7.1 Training curves of 12 different models on `CALTECH101-18`. Blue curves are training evidence; Orange curves are test evidence.

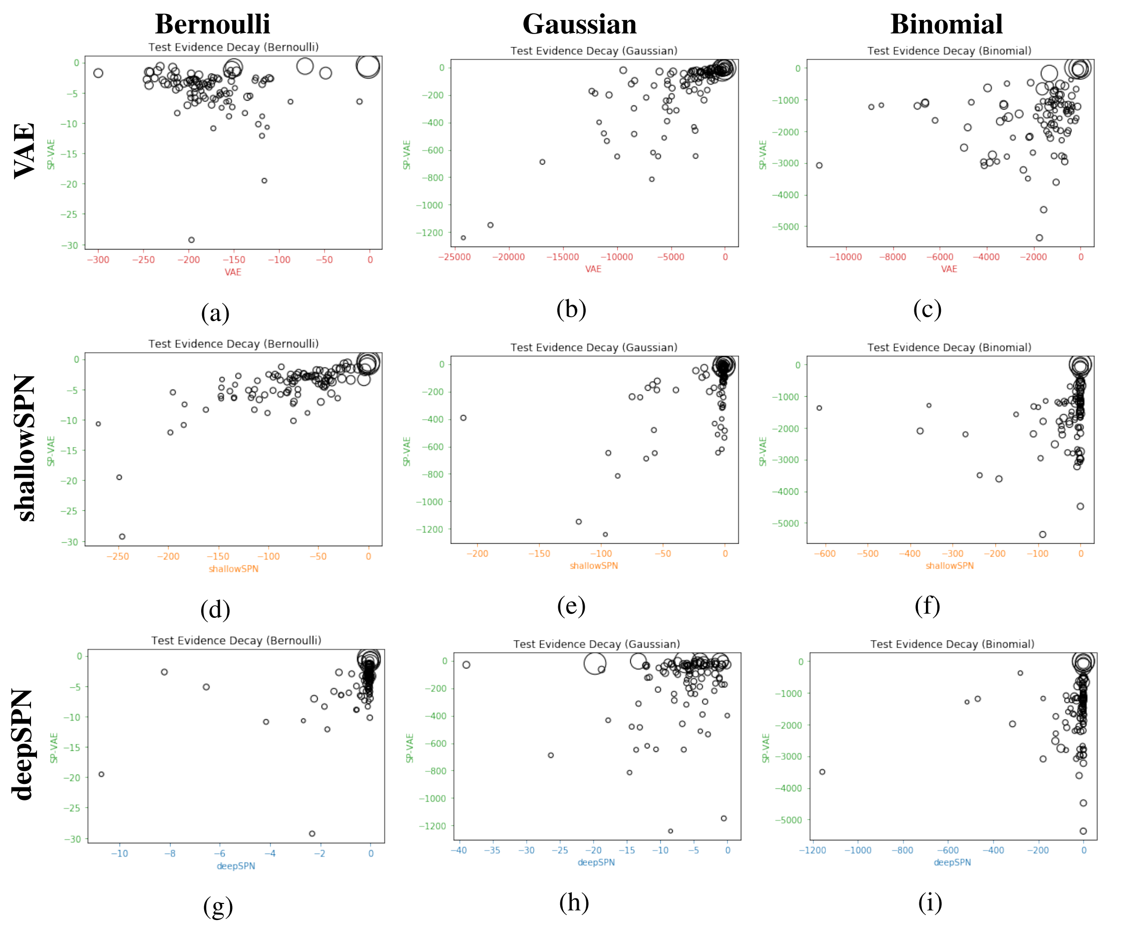


Fig. 7.2 (Best viewed in color) “test evidence decay" of SP-VAE (green) against that of VAE (red), shallowSPN (orange), and deepSPN (blue) on 102 CALTECH101 datasets. Area of scatterpoint indicates size of dataset. Attention should be paid to the scales of the axes

To ensure that this is not unique to CALTECH101-18, Figure 7.2 plots the “test evidence decay"[1] of SP-VAE against that of other models. The results are consistent with the initial observation of CALTECH101-18. For example, consider the Gaussian models from Figure 7.2. Mean “test evidence decays" are around 4000 nats for VAEs, 200 nats for SP-VAE, 13 nats for shallowSPN and 7 nats for deepSPN. This strongly suggests that SPNs are more limited in the extent to which they can overfit than VAEs. A simple reason for this may be that SPNs are less expressive than VAEs. As a hybrid of the two, SP-VAE retains some of SPN’s robustness against overfitting.

Promising results for SP-VAE

SP-VAE’s best evaluated evidence on the test set (y-axis) is compared against that of other pure models (x-axis) in Figure 7.3. If a point falls in the upper triangle, SP-VAE performed better.

[1]test evidence decay: defined as best evidence on test set minus evidence on test set after training has converged (No early stopping)

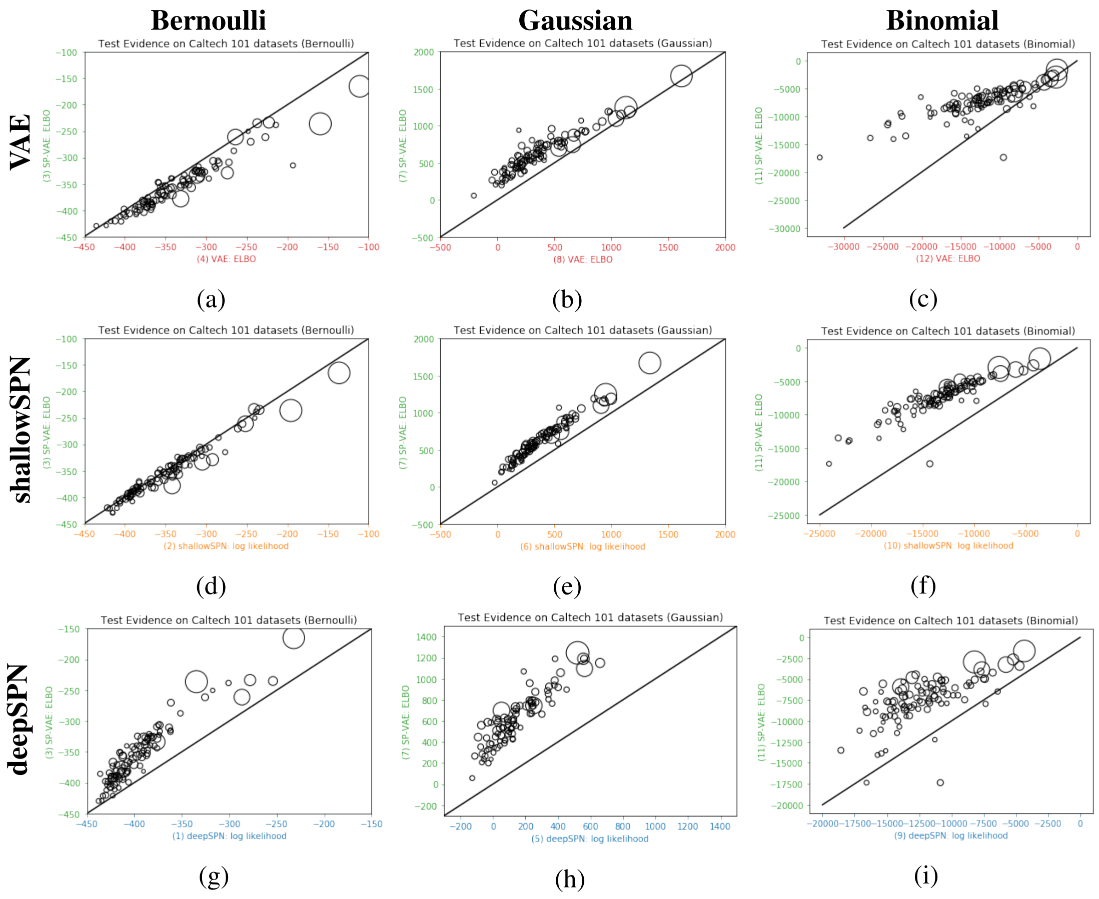


Fig. 7.3 (Best viewed in color) Comparing the *best* test evidence of SP-VAE (green) against that of VAE (red), shallowSPN (orange), and deepSPN (blue) on 102 CALTECH101 datasets. Area of scatterpoint indicates size of dataset. If a scatterpoint falls in the upperright triangle, SP-VAE outperform the model is compared to on the x-axis. Many deepSPN models did not converge as training took too long. Except for (a) and (d), all figures show that SP-VAE is superior.

In the Gaussian and Binomial case of Figure 7.3, almost all 102 scatterpoints fell in the upper right triangle, indicating that SP-VAE is the best architecture.

SP-VAE outperformed SPNs as expected. On a 5x5 patch, the shallowSPN leaves are mixtures of 7 diagonal distributions, while the deepSPN leaves over the same patch are mixtures of 2 subSPNs. The SP-VAE leaves over the same patch are 2 layer MLP density networks which, without proof, I assume is more expressive than the others.

That SP-VAE outperformed VAEs is a new unintuitive finding, because if SPNs are less expressive than VAEs, then the hybrid SP-VAE should also be less expressive. To test this intuition, side experiments tested VAEs with increased model capacity, and again surprisingly, that increased theoretical flexibility translated to worse test outcomes. Perhaps one reason why SP-VAEs are superior is that it removed unnecessary modelling power but retains the ability to model local image correlations. In a simpler optimization landscape, the optimizer has a better chance of finding a good local optimum.

SP-VAE's comparative advantage seems to lie in smaller datasets. In Figure 7.3, scatter points with lower test evidence tends to lie in SP-VAE's favor (upper left triangle). An exception to this trend seems to be Bernoulli and Gaussian deepSPNs, where higher scoring datasets are in SP-VAE's favor. But this is because deepSPN models ended training before convergence as training took too long.

Anomaly

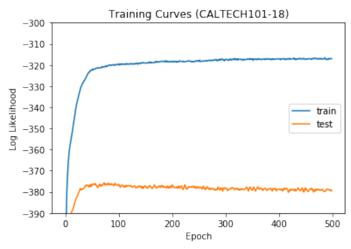
However, in the Bernoulli case, SP-VAE performs poorer than VAEs. This is puzzling and may be due to two reasons:

First, pixels throughout a binary image are globally correlated. In a 5x5 patch, it is difficult to predict what one pixel is given the other 24; information from other patches is needed. Consequently, the modelling power between patches is important.

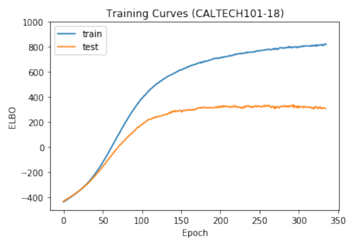
Second, SP-VAE does not have sufficient modelling power between its leaves. As seen in the Table 7.1, SP-VAE stands out for placing most of its modelling power in the leaves (~97% of parameters are VAE parameters), while deepSPN and shallowSPN places most of their modelling power in the SPN structure (~96% of parameters are SPN sum node weights). As demonstrated in the Gaussian and binomial case, weaker interactions between leaves need not be a fatal feature. Instead this suggests that the ratio of modelling power within leaves and between leaves should be tuned. In this binary CALTECH101 dataset, perhaps more modelling power than necessary have been given to the Bernoulli VAE leaves.

Deep SPNs are the worst

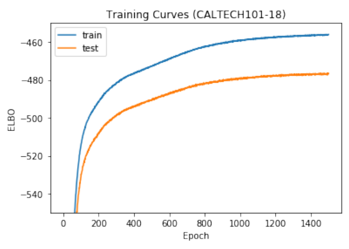
In all distribution types, deepSPNs fared the worst, and by a wide margin. Perhaps I had used insufficient number of leaves per region. But multiple authors [32, 14] also noted difficulty in training deep SPNs by gradient methods, a difficulty they attributed to the gradient diffusion problem.



(a) Bernoulli Density Net

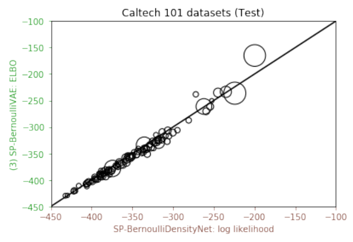


(b) Gaussian Density Net

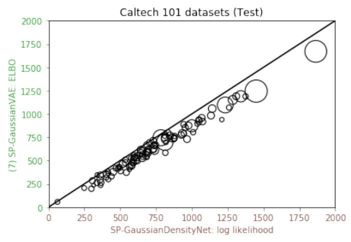


(c) Binomial Density Net

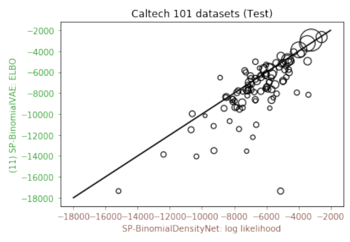
Fig. 7.4 Training Curves



(a) Bernoulli Density Net



(b) Gaussian Density Net



(c) Binomial Density Net

Fig. 7.5 Comparing best scores of SP-VAE vs SP-DensityNet

While SPNs are very deep indeed (18 levels at its deepest in our models), I doubt if vanishing gradients may be the cause. First because training was done via by ADAM which adapts learning rates. Second because there are also many “residual connections"[18] between nodes of vastly different depths. Investigating the poor performance of deep SPN is left for future work.

7.2 Experiment 2: Learning without Inference Networks

7.2.1 Motivations

In SP-VAEs, the VAE leaves are tiny. They only need to model a subset of the problem scope, and hence their latent variable need not be high-dimensional — perhaps only one or two. This begs the question if the VAE leaves' marginal likelihood (Eq 3.1) is intractable in the first place.

By breaking a large VAE into many small VAEs, SP-VAE shifts the cost of marginalization from the exponent to the multiplicative constant. I hypothesize that SP-VAE can still train without the inference networks of the VAEs. The resulting model is called SP-DensityNet.

7.2.2 Results and Discussions

SP-DensityNet can work

As seen in Figure 7.5, SP-DensityNet tends to achieves higher evidence than SP-VAE. This is expected because the former computes an estimate of the log marginal likelihood while the latter computes a lower bound.

Interestingly, in the case of Bernoulli and Gaussian SP-VAEs, the ELBO is close to the true log likelihood. On average, SP-DensityNet is only 4 nats higher than SP-VAE in the Bernoulli case, and 100 nats higher in the Gaussian case. This begs the question of what affects the bound gap between SP-VAE and SP-DensityNet. Preliminary experiments showed that the tightness of the bound is largely determined by the number of importance weighted samples K in the ELBO objective. Increasing the dimension of latent variable of the VAE leaves from 2 to 5 does little to worsen to bound gap.

An anomaly is the binomial case, where the SP-DensityNet evidence can be many thousands of nats higher than that of VAE. This is very curious and it suggests either that importance weighted ELBO of Binomial VAE is still not tight. Having a tight ELBO is important because ultimately we would like to optimize our generative models according to the marginal likelihood.

This experiment demonstrates that SP-VAE does not need an inference network to train. If the individual tiny inference networks can be removed, perhaps they also can be aggregated. A single large inference network that simultaneously predicts the posterior over latent variables for all density network leaves may be more convenient if downstream applications require an encoder.

In the larger context of variational inference, this result is promising because training density networks is fundamentally unscalable with higher dimensional latent variables. It was only recently that VAEs were proposed as a tractable and scalable way of training them. SPNs offer an alternative divide and conquer approach towards density network training and inference.

SP-DensityNet vs SP-VAE

If SP-DensityNet gives higher evidences and computes exact likelihood, should it always be preferred over SP-VAE? The answer depends on the application. If only density estimates are required, then SP-DensityNet is preferred.

SP-DensityNet has the additional advantage of simplicity. This is important since VAEs may suffer from the problem of posterior mismatch and recent literature has proposed various modifications to the inference network (for example [39, 20]) to improve the quality of the approximate posterior. Without an inference network, not only do we avoid posterior mismatch, we also avoid making unnecessary design choices and worrying about the influence of those choices on the quality of the approximate posterior. As a bonus, the training will converge faster as well.

However, the inference network is still needed if we need to approximately sample the mode of the VAE distribution (argmax operation), or perform representation learning.

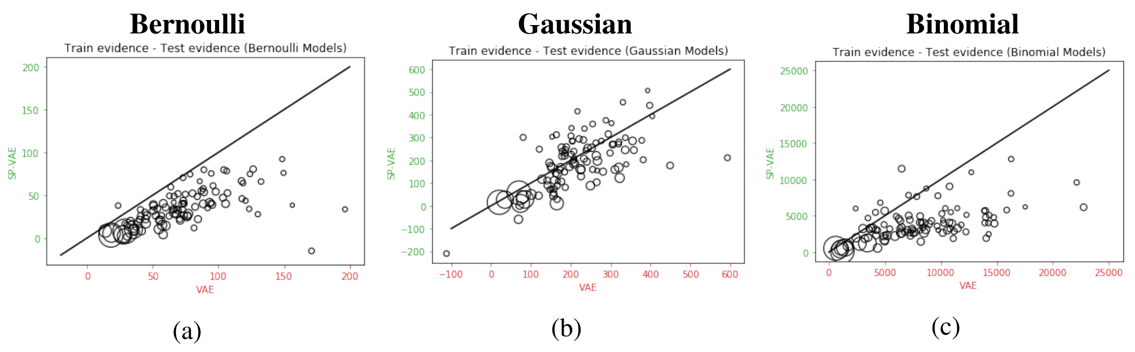


Fig. 7.6 Test gap (train evidence - test evidence) at peak test performance of SP-VAE vs VAE on CALTECH101 datasets.

7.3 Experiment 3: Data Efficiency

7.3.1 Motivation

In experiment 1, SP-VAE tends to do better than VAE on smaller datasets, and par with VAE on larger datasets. Also, when examining the the models training curves, it is hard to miss the fact that the gap between the training and test evidence tends to be smaller in SP-VAE than VAEs, and even smaller still in deepSPNs compared to SP-VAE.

Figure 7.6 shows a scatterpoint of test gaps (train evidence - test evidence) at peak test performance of SP-VAE (y-axis) against VAE (x-axis). The size of the scatter point indicates the size of the dataset. If a scatterpoint falls in the lower right triangle, SP-VAE has a smaller test gap.

These observations suggest that perhaps when SPNs are combined with VAEs, the SPNs have a regularizing effect on the VAE leaves which constrains the extent to which they can overfit. To investigate this intuition, I compared the best performances of three models (shallowSPN, SP-VAE, and VAE) on MNIST as the amount of training data is varied.

7.3.2 Setup

I selected the free hyperparameters of each models such that all models have approximately the same number of parameters. They are detailed in Table 7.2.

Amount of training data is varied by taking a randomly selected fraction ($datafrac$) of the 60,000 images in the original MNIST dataset. Because the reduced datasets are not consistent across models, the experimental sweep through $datafrac$ is repeated at least twice to ensure that results are consistent and reproducible.

7.3.3 Results and Discussions

Figure 7.7 shows the best test evidence on MNIST of shallowSPN, SP-VAE, and VAEs against pruning fraction. The higher the test evidence, the better the performance.

Table 7.2 Experiment 3 Model Size

Model Code	Region Graph	SPN		VAE		# Weights		
	coarsegrain	sumK	leafK	n_h	n_z	total	leaves	spn
M2 / M10	4	4	4	-	-	203264	3136	200128
M3 / M11	4	2	2	16	2	143032	117992	25040
M4 / M12	-	-	-	71	98	143548	-	-
M6	4	4	4	-	-	206400	6272	200128
M7	4	2	2	16	2	169688	144648	25040
M8	-	-	-	61	98	170856	-	-

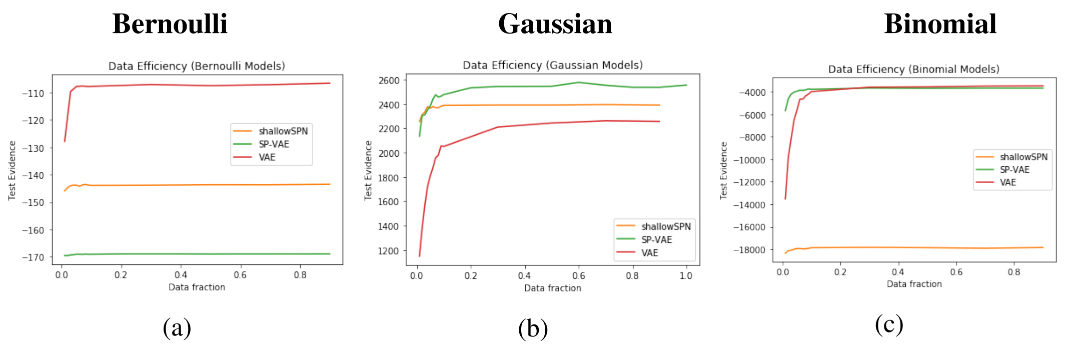


Fig. 7.7 (Best viewed in color) Test evidence of 9 different models on `MNIST` as amount of training data is varied

As an aside, Figure 7.8 presents and compares the fastest training and testing curves of shallowSPN, SP-VAE, and VAE. All comparable models presented here were trained with the same batch size, while the learning rate was tuned by grid search.

SPNs are robust to low resource data

In Figure 7.7, the test evidence of all three model architectures all show the same logarithmic style of performance saturation as the amount of training data grows. However, SPNs are the first to reach saturation point, beyond which additional training examples do not improve test evidence; VAEs are the last to reach saturation. The marginal gain from additional training examples is also minimal in SPNs, while VAEs stand to gain the most. Perhap unsurprising for a hybrid, SP-VAE's saturation point and marginal gain are intermediate between SPNs and VAEs.

In an online learning setting where the amount of training data grows incrementally, SP-VAE will be advantageous over VAE during the early stage of few training examples. Unlike SPNs, this does not come at a cost of worse performance in the later stages when there are more training examples.

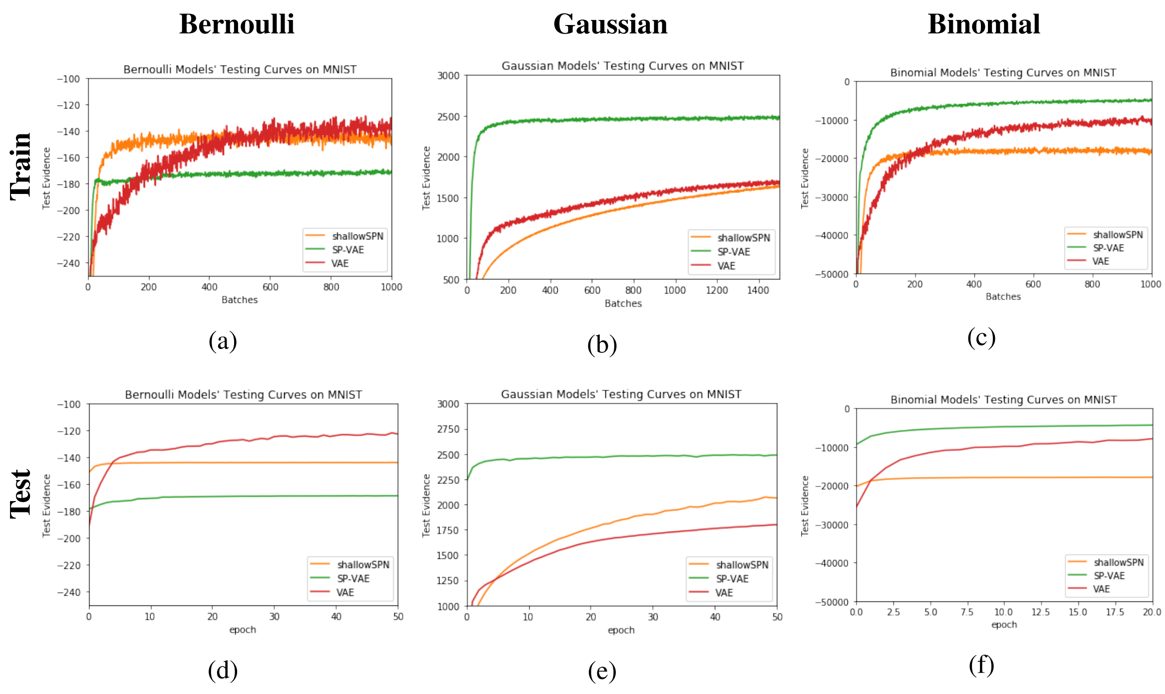


Fig. 7.8 (Best viewed in color) Fastest converging training and test curves of 9 different models on `MNIST`. Notice the green curve.

Aside: SP-VAE converges faster than VAEs

Initially, I wanted to investigate if datasets with fewer examples allow for higher learning rates and larger batch sizes, and thereby allowing faster convergence, but I noticed a more fundamental trend — the different model architectures consistently show different convergence rates. The difference is consistent across datasets pruned to various sizes.

In Figure 7.8, SP-VAE converged the fastest in all cases. At first sight, such a comparison between models of different complexities may not be fair — a model may be increasing its training evidence faster but that may be due to it converging to a higher evidence. But the consistency of its fast convergence irregardless of its converged training likelihood suggests that there may be some form of training acceleration implicit in SP-VAE.

To seek more experimental verification of SP-VAE's convergence speed, I looked back at Experiment 1's results. VAEs may sometimes increase its training evidence faster than SP-VAE, but SP-VAE invariably converged earlier than VAEs. In the binomial case, deepSPNs converged the fastest.

If SP-VAE converges faster than VAEs, the cause is not clear. Perhaps one reason is that the depth offered by the SPN to the VAE leaves created an implicit training acceleration. This is a recent insight by Sanjeev et al [3] into the role of depth in deep learning. As unintuitive as it sounds, [3] showed that increasing the depth in linear models with L_p regularization where $p > 2$ leads to faster convergence. Linear models were chosen because increasing their depth only increases the number of parameters in the model but not the model complexity. The authors noted that depth in non linear models may

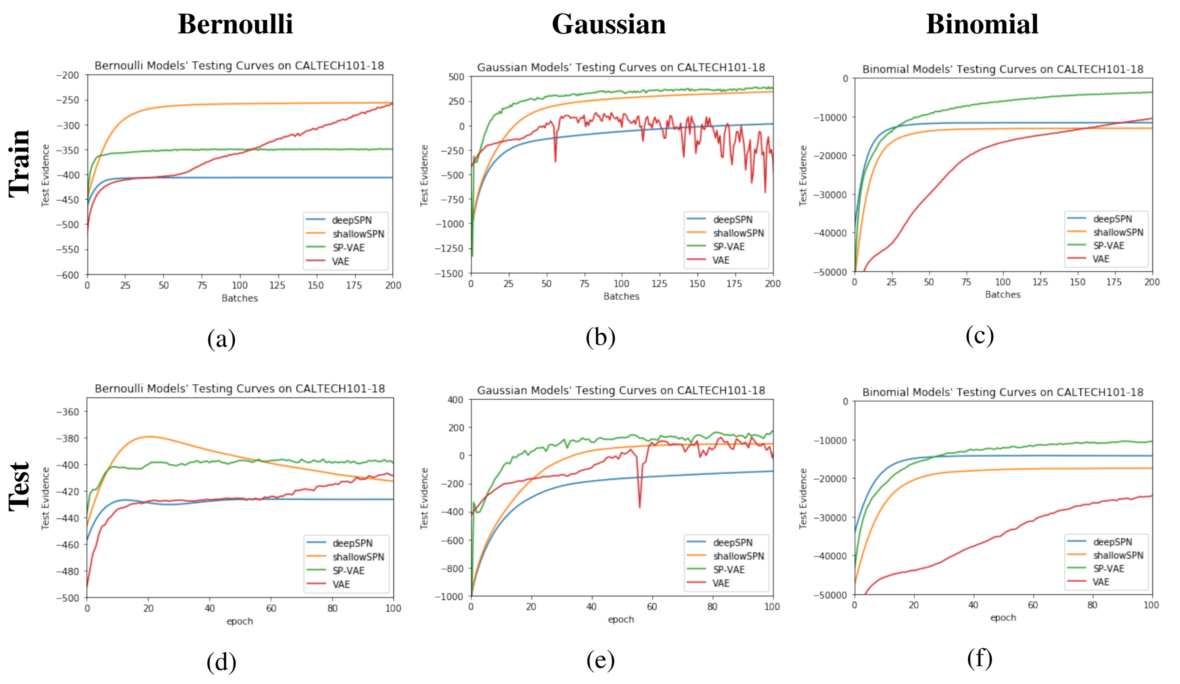


Fig. 7.9 (Best viewed in color) Training and test curves of 12 different models on `CALTECH101-18`. Notice the green curve

have the same effect but it is not guaranteed. Ever increasing depth is also not ideal as the vanishing gradient problem starts to affect training.

Another reason may be the divide-and-conquer nature of SP-VAE. When a large VAEs is broken into smaller pieces and combined exactly through an SPN, the variance in the ELBO estimate has smaller variances. With less noise in the gradients, fewer gradient updates are needed. Additionally, since the weights in a mixture of VAEs are “more decoupled" than the weights in a single large VAE, optimization of the former may be an easier problem. To test this intuition, a future experiment may compare SP-VAEs against VAEs as the number of pixels in the image problem scales up.

7.4 Experiment 4: Ratio of SPNs to Leaves

7.4.1 Motivation

A major role of the SPN is to learn the correlations between its leaves. In previous experiments, SP-VAE were constructed with most of its modelling power is in its leaves. Could better performance be obtained if there were more modelling power in the SPN?

In this experiment, I compare how SP-VAE, and shallowSPN’ performances on SVHN change as the ratio between the number of sum node weights and number of VAE weights is varied.

Table 7.3 Experiment 4 model sizes : SPVAE

Distribution Type	Parameters		# Weights			% spn
	K	n_h	leaves	spn	total	
Bernoulli Binomial	2	31	200576	1576	202152	0.78
	3	22	195456	5292	200748	2.64
	4	17	191488	12512	204000	6.13
	5	13	176000	24400	200400	12.18
	6	10	158208	42120	200328	21.03
	7	7	126784	66836	193620	34.52
	8	5	103424	99712	203136	49.09
	9	3	72000	141912	213912	66.34
Gaussian	2	35	308864	1576	310440	0.51
	3	25	308736	5292	314028	1.69
	4	19	300288	12512	312800	4.00
	5	15	288960	24400	313360	7.79
	6	12	273024	42120	315144	13.37
	7	9	236544	66836	303380	22.03
	8	7	210432	99712	310144	32.15
	9	5	171648	141912	313560	45.26
	10	3	120960	194600	315560	61.67

7.4.2 Setup

For SP-VAE, there are two free parameters: 1) *nh* is the number of hidden nodes per layer and 2) $K = K_{leaf} = K_{sum}$ is the number of sum nodes or leaf nodes per region. Similarly there are two free parameters for shallowSPN: 1) K_{sum} is the number of sum nodes per region while 2) K_{leaf} is the number of leaf nodes per leaf region.

The free parameters of SP-VAE and shallowSPN were varied while keeping the overall model size approximately constant:

7.4.3 Results and Discussions

Figure 7.10 presents the test evidence of shallowSPN and SP-VAE the ratio of number of SPN sum node weights to number of leaf parameters is varied. As part of hyperparamter tuning, training was done with three learning rates, 0.1 (green), 0.01 (blue), and 0.001 (orange). Figure 7.10 presents the results of these learning rate tuning attempts with a standard batch size of 100.

Ratio affects optimal learning rate of SP-VAE

In Figure 7.10, the Gaussian and binomial case of SP-VAE suggest that there is no single optimal learning rate for SP-VAE across all range of sum nodes to leaf ratios. If a greater fraction of weights

Table 7.4 Experiment 4 model sizes : shallowSPN

Distribution Type	Parameters		# Weights			% spn
	$Kleaf$	$Ksum$	leaves	spn	total	
Bernoulli Binomial	53	2	54272	146008	200280	72.90
	44	3	45056	161748	206804	78.21
	37	4	37888	167744	205632	81.57
	31	5	31744	167920	199664	84.10
	27	6	27648	178200	205848	86.57
	22	7	22528	175196	197724	88.61
	18	8	18432	180352	198784	90.73
	14	9	14336	186192	200528	92.85
	10	10	10240	194600	204840	95.00
	5	11	5120	198748	203868	97.49
Gaussian	59	2	120832	179416	300248	59.76
	49	3	100352	197388	297740	66.30
	43	4	88064	218432	306496	71.27
	37	5	75776	224080	299856	74.73
	33	6	67584	240408	307992	78.06
	28	7	57344	239708	297052	80.70
	24	8	49152	247168	296320	83.41
	21	9	43008	266328	309336	86.10
	17	10	34816	273560	308376	88.71
	13	11	26624	283228	309852	91.41
	8	12	16384	285408	301792	94.57
	3	13	6144	296192	302336	97.97

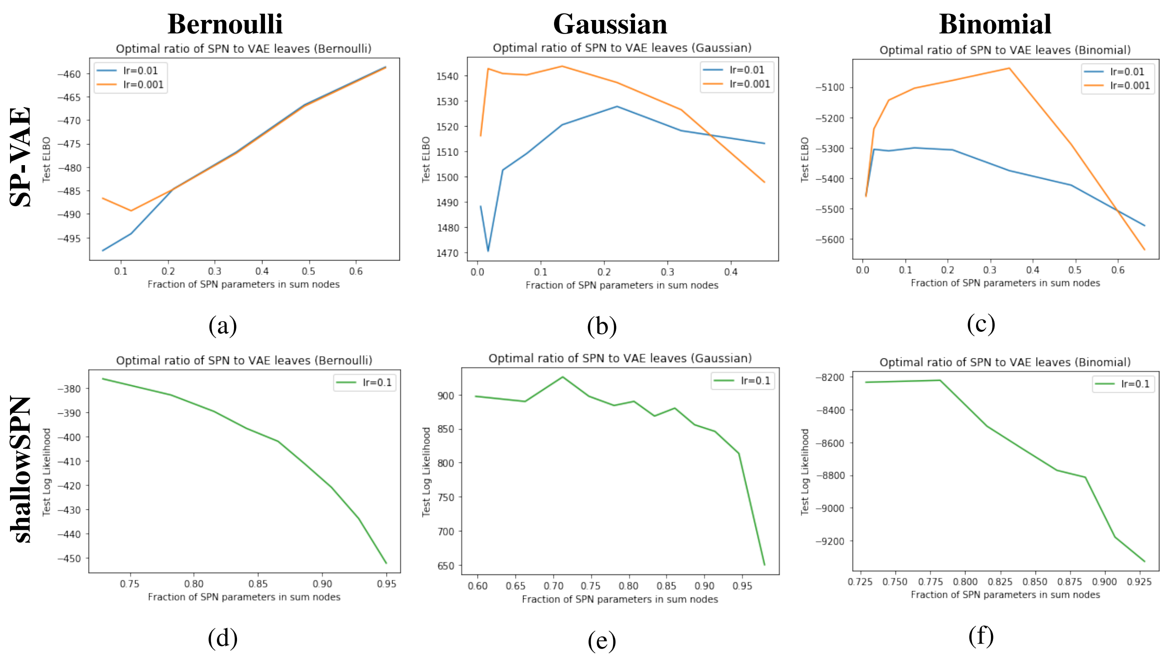


Fig. 7.10 Test evidence on SVHN against fraction of SPN parameters in sum nodes

are in sum nodes, a larger learning rate is better. This agrees with observations from experiment 1 that SPNs a greater magnitude of learning rate compared to VAEs, and gives further motivation to developing hybrid training methods for SP-VAE.

Balanced ratios give higher test performance

Intuitively, one would expect that SP-VAE would perform poorly if most of its parameters are in its leaves or SPN sum nodes only. Indeed, this inversed U shape is observed in the Gaussian and binomial case of SP-VAE. Presumably it is not observed in the other cases because the sweep of sum node weights fraction did not pass the optimal ratio and enter the opposite regime.

In general, the optimal ratio of sum node weights to leaf weights for SP-VAE seems to lie near 50:50. An exception to this trend is Bernoulli SP-VAE, reflecting the same anomaly found in experiment 1. This suggest that binary datasets are fundamentally challenging for the divide and conquer approach of SP-VAE.

Chapter 8

Further Work

“Maybe we could try..."

In the process of defining, implementing and evaluating SP-VAE, there were multiple ideas to explore which were not prioritized in this thesis. This section will summarize these ideas.

8.1 Learning and Inference Methods

8.1.1 Gradient-based Argmax for SPNs and SP-VAEs

An MPE inference algorithm for selective SPNs is often employed on non-selective SPNs to efficiently obtain an approximate optimum.

After obtaining an approximate MPE estimate, perhaps the estimate can be further optimised to the true MPE solution using gradient-based optimimzation. Differentiating with respect to the input X is similar to approaches used in generating adversarial examples [42]. If X is discrete, this gradient-based approach is still possible by first relaxing the discrete variable onto the continuous simplex using the concrete distribution [24, 19].

Similarly in VAE leaves, assuming that the latent variables of the leaves are low-dimensional, a Monte Carlo estimate of the marginal likelihood function $\tilde{p}$ can be constructed and differentiated with respect to the input X.

$$\tilde{p}(X = x) = \frac{1}{N} \sum_{i=1}^{N} p_{\theta}(X = x | Z = z_i) \quad , \quad z_i \sim p_{\theta}(z) \tag{8.1}$$

The quality and costs of the gradient-based argmax should be analyzed.

8.1.2 Hybrid EM + Gradient-based Parameter Learning of SP-VAE

In a recent preprint, [31] showed that EM algorithm converges in much fewer iterations than gradient methods on SPNs. So an hybrid training algorithm could possibly be to interleave EM algorithm for the SPN and stochastic gradient methods for the VAE.

8.1.3 Different Learning Rates SP-VAE

A single learning rate was tuned when optimizing SP-VAE. However, since it is found that SPNs and VAEs have fundamentally different optimal learning rates, perhaps future optimization of SP-VAE should tune two separate learning rates.

8.2 Effects to Investigate

8.2.1 Regularization

In this thesis, $L1$ regularization was used on the sum node weights. However, beyond the intuitive effect that it favors balanced probabilities, little else is known. Is $L1$ the best way to regularize sum nodes? Or could a concrete distribution prior [24] be placed on the sum node weights.

8.2.2 Vanishing Gradients

As mentioned in the discussion of deepSPN, multiple authors [32, 14] have reported difficulties of training deep SPNs with gradient-based methods, and the reason is often attributed to the vanishing gradient problem. Yet deepSPNs have many residual connections, which are supposed to alleviate the vanishing gradient problem. An experimental verification of the vanishing gradient problem can clarify this doubt.

8.3 Architectures

8.3.1 Shared SP-VAE

The SP-VAE used in this thesis had coarsegrain lengthscale c that was an integer factor of the image dimensions. This meant that PD-SPN will decompose the image down to “superpixels". All superpixels will be disjoint and of the same size of $c \times c$. However, if the image dimensions are not multiples of the coarsegraining length, then leaf scopes are of different sizes, while some leaf scopes have overlapping random variables.

It seems almost costly that some random variables are repeated in multiple leaf scopes. Instead of design an SPN architecture or coarsegraining policy that minimises overlap in leaf scopes, we could design an SPN architecture that maximizes the overlap in leaf scopes and instead share the weights of the VAEs that implements the leaf nodes in the leaf scopes. This motivates us to propose a Shared SPN architecture.

The benefit of sharing the weights of a VAE is that a single (or multiple) VAEs of size nxm can be trained on NxM images, where $n < N$ and $m < M$. When a high modelling power leaf is mixed with a low modelling power leaf, the likelihood of data can be maximized by placing the high powered leaf on the complicated foreground, and the less complicated leaves on the uninteresting background.

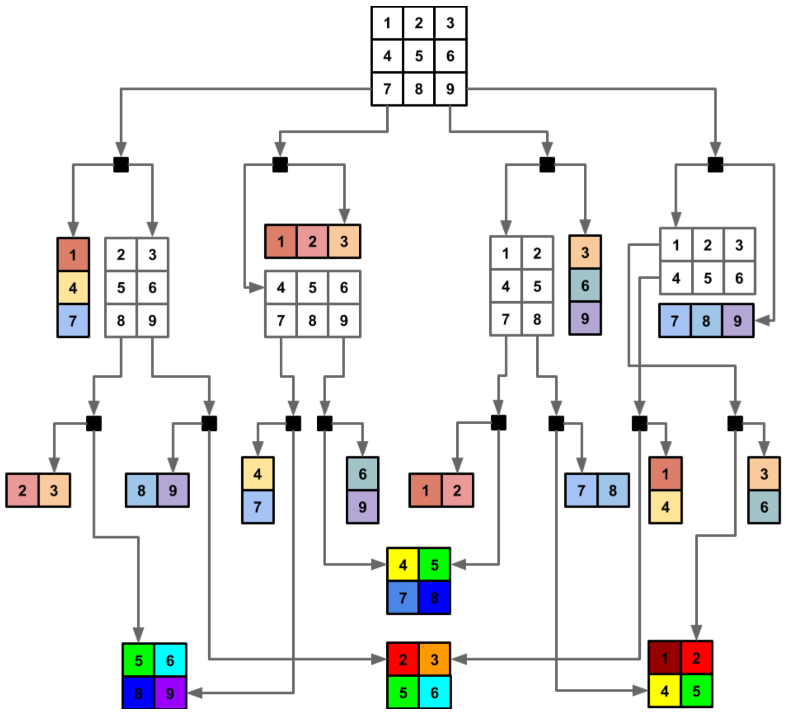


Fig. 8.1 Bright colored regions will be modelled by a shared VAE. Light colored regions can be implemented by diagonal distributions.

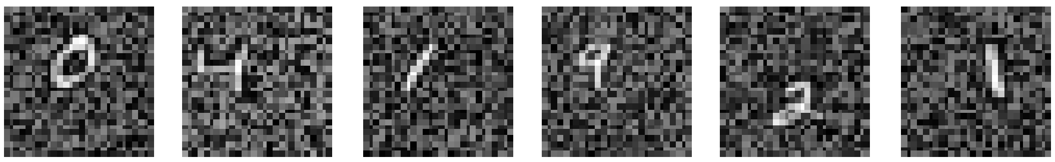
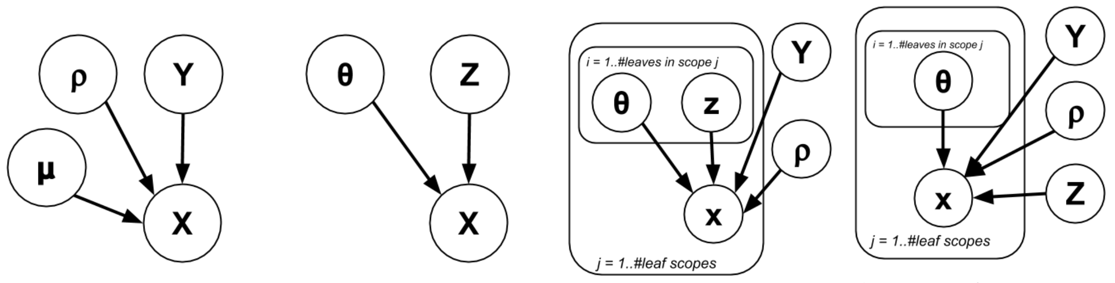


Fig. 8.2 Synthetic problem for shared SP-VAE.

If there are multiple VAEs, perhaps the model could also learn different classes of images in an unsupervised way.

To illustrate what I mean, consider a reduced 14×14 MNIST with on a 28×28 Gaussian noise background in Figure 8.2. The expectation for shared SP-VAE is not only to learn how to generate the 14 x 14 MNIST digits given 28×28 training images, but also learn their classes and location within the training image, all in an unsupervised way. A recent paper [13] has been exploring a similar problem with a synthetic dataset called Multi-MNIST.



(a) SPN (b) VAE (c) SPVAE (d) SPVAE for representation learning?

Fig. 8.3 Graphic Models. Y is a multivariate discrete variable related to the SPN sum nodes. Z is a multivariate continuous variable related to the VAE. θ are the parameters of the VAE. ρ are the categorical distributions of the SPN sum nodes. μ are the parameters of the traditional SPN leaves.

8.3.2 Representation Learning

Instead of sharing the entire VAE, in the current superpixel architecture, the latent variables of the SPN may also be shared improve entanglement between the VAE leaves. VAE leaves entangled through their latent variables may have greater difficulty performing inference. This may be where a shared global inference network may be helpful.

Since every VAE leaf only models a smaller sub-scope of the problem, it would be interesting to investigate if this encourages disentangled latent representations.

8.3.3 SP-NADEs

SP-VAE was motivated by the observation that the leaves of an SPN needs, in its most basic requirements, to be a distribution with tractable likelihood. Besides VAEs, there are also other neural network based density estimator like Neural Autoregressive Density Estimator [38]. On images, autoregressive density estimation methods tend to read the image pixel by pixel, and row by row. Perhaps if NADE were used as SPN leaves, they could offer SPNs its expressiveness in density estimation, while SPNs could offer NADE training and inference benefits by its ability to “divide and conquer" a problem.

8.4 Engineering

8.4.1 Tensorial Decomposition Formulation of PD-SPN

At the moment, deep learning frameworks are not well suited to implement SPNs which have numerous but cheap operations. The tensorial decomposition formulation of PD-SPN, which makes sums and products implicit within a `tensordot` operation, is a promising way to more compactly define the operations in an PD-SPN.

To prevent underflow, likelihood computations have to be done in the log domain and unfortunately, `log_tensordot` is not yet supported in any deep learning frameworks. Writing a C++ `log_tensordot` kernel for TensorFlow is a possible solution.

Once SPNs have better hardware and software support, research on SPN can iterate faster and experiment with more complex and heterogenous mix of leaves.

Chapter 9

Summary and Conclusions

“SP-VAEs are promising!"

This thesis proposed a hybrid combination between SPNs and VAE, referred to as SP-VAE, and successfully evaluated it against pure SPNs and vanilla VAEs on the problem of density estimation of CALTECH101 datasets.

SP-VAEs were clearly superior over pure SPNs due to greater model complexity, and except for binarized datasets, SP-VAE showed competitive, and if not superior results than VAEs on all datasets. By keeping VAEs small and composing more of them with an SPN as the problem scope grows, I was able to train SP-VAEs without inference networks. Compared to a single large VAE, a composition of many smaller VAEs has the benefits of faster convergence and greater robustness against overfitting.

While this thesis shows that SP-VAEs are promising, there is scope for more work to be done to further demonstrate the credibility and viability of SP-VAE as a model:

On the theoretical front, SP-VAE's inference algorithm may be improved upon by sophisticated Monte Carlo maximum likelihood algorithms. The insight I found that SP-VAE requires different learning rates for its SPN and VAE parts motivates future hybrid parameter learning algorithms.

On the evaluation front, a metric other than the *best* test likelihood could be used to more convincingly demonstrate SP-VAE's potential. The relative benefits of SP-VAEs over VAEs as the problem difficulty scales up has also yet to be investigated.

On the application front, the machine learning community is currently less interested in density estimation and more in learning an interpretable representation of data in an unsupervised fashion — a task considered as one of the “holy grails" of machine learning. SP-VAE could be studied along these lines since they naturally decompose dimensions of a latent variable to subscopes of the problem.

Looking beyond this thesis and SP-VAE, the machine learning community has strong interests in combining probabilistic graphical models with deep learning in order to create a balanced solution that capitalizes on the strengths of both fields. This thesis evaluated one way in which this can be done between SPNs and VAEs, with promising results. Perhaps SPN holds potential for introducing structure, and ultimately tractable inference, into other deep learning models.

References

[1] Adel, T., Balduzzi, D., and Ghodsi, A. (2015). Learning the structure of sum-product networks via an svd-based algorithm. In *UAI*, pages 32–41.

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

[3] Arora, S., Cohen, N., and Hazan, E. (2018). On the optimization of deep networks: Implicit acceleration by overparameterization. *CoRR*, abs/1802.06509.

[4] Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.

[5] Burda, Y., Grosse, R., and Salakhutdinov, R. (2015). Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*.

[6] Cheng, W.-C., Kok, S., Pham, H. V., Chieu, H. L., and Chai, K. M. A. (2014). Language modeling with sum-product networks. In *Fifteenth Annual Conference of the International Speech Communication Association*.

[7] Choi, A. and Darwiche, A. (2017). On relaxing determinism in arithmetic circuits. *CoRR*, abs/1708.06846.

[8] Darwiche, A. (2001). Decomposable negation normal form. *J. ACM*, 48(4):608–647.

[9] Darwiche, A. (2003). A differential approach to inference in bayesian networks. *Journal of the ACM (JACM)*, 50(3):280–305.

[10] Delalleau, O. and Bengio, Y. (2011). Shallow vs. deep sum-product networks. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 24*, pages 666–674. Curran Associates, Inc.

[11] Dennis, A. and Ventura, D. (2012). Learning the architecture of sum-product networks using clustering on variables. In *Advances in Neural Information Processing Systems*, pages 2033–2041.

[12] Dennis, A. and Ventura, D. (2015). Greedy structure search for sum-product networks. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 932–938. AAAI Press.

[13] Eslami, S. M. A., Heess, N., Weber, T., Tassa, Y., Kavukcuoglu, K., and Hinton, G. E. (2016). Attend, infer, repeat: Fast scene understanding with generative models. *CoRR*, abs/1603.08575.

[14] Gens, R. and Domingos, P. (2012). Discriminative learning of sum-product networks. In *Advances in Neural Information Processing Systems*, pages 3239–3247.

[15] Gens, R. and Pedro, D. (2013). Learning the structure of sum-product networks. In *International conference on machine learning*, pages 873–880.

[16] Germain, M., Gregor, K., Murray, I., and Larochelle, H. (2015). MADE: masked autoencoder for distribution estimation. *CoRR*, abs/1502.03509.

[17] Hardt, M., Recht, B., and Singer, Y. (2015). Train faster, generalize better: Stability of stochastic gradient descent. *CoRR*, abs/1509.01240.

[18] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

[19] Jang, E., Gu, S., and Poole, B. (2016). Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

[20] Kingma, D. P., Salimans, T., and Welling, M. (2016). Improving variational inference with inverse autoregressive flow. *CoRR*, abs/1606.04934.

[21] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

[22] Lowd, D. and Rooshenas, A. (2013). Learning markov networks with arithmetic circuits. In *Artificial Intelligence and Statistics*, pages 406–414.

[23] Mackay, D. J. and Gibbs, M. N. (1999). Statistics and neural networks. chapter Density Networks, pages 129–145. Oxford University Press, Inc., New York, NY, USA.

[24] Maddison, C. J., Mnih, A., and Teh, Y. W. (2016). The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*.

[25] Peharz, R. (2015). *Foundations of sum-product networks for probabilistic modeling*. PhD thesis, PhD thesis, Graz University of Technology.

[26] Peharz, R., Geiger, B. C., and Pernkopf, F. (2013). Greedy part-wise learning of sum-product networks. In *Proceedings of the 2013th European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part II*, ECMLPKDD'13, pages 612–627, Berlin, Heidelberg. Springer-Verlag.

[27] Peharz, R., Gens, R., and Domingos, P. (2014a). Learning selective sum-product networks. In *LTPM workshop*.

[28] 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.

[29] Peharz, R., Kapeller, G., Mowlaee, P., and Pernkopf, F. (2014b). Modeling speech with sum-product networks: Application to bandwidth extension. In *ICASSP*, pages 3699–3703.

[30] Peharz, R., Tschiatschek, S., Pernkopf, F., and Domingos, P. (2015). On theoretical properties of sum-product networks. In *Artificial Intelligence and Statistics*, pages 744–752.

[31] Peharz, R., Vergari, A., Stelzner, K., Molina, A., Trapp, M., Kersting, K., and Ghahramani, Z. (2018). Probabilistic deep learning using random sum-product networks. *arXiv preprint arXiv:1806.01910*.

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

[33] Rainforth, T., Kosiorek, A. R., Le, T. A., Maddison, C. J., Igl, M., Wood, F., and Teh, Y. W. (2018). Tighter variational bounds are not necessarily better. *arXiv preprint arXiv:1802.04537*.

[34] Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*.

[35] Rooshenas, A. and Lowd, D. (2014). Learning sum-product networks with direct and indirect variable interactions. In *International Conference on Machine Learning*, pages 710–718.

[36] Sharir, O., Tamari, R., Cohen, N., and Shashua, A. (2016). Tensorial mixture models. *CoRR*, abs/1610.04167.

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

[38] Uria, B., Côté, M.-A., Gregor, K., Murray, I., and Larochelle, H. (2016). Neural autoregressive distribution estimation. *The Journal of Machine Learning Research*, 17(1):7184–7220.

[39] van den Berg, R., Hasenclever, L., Tomczak, J. M., and Welling, M. (2018). Sylvester normalizing flows for variational inference. *arXiv preprint arXiv:1803.05649*.

[40] van den Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. (2016a). Pixel recurrent neural networks. *CoRR*, abs/1601.06759.

[41] van den Oord, A., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A., and Kavukcuoglu, K. (2016b). Conditional image generation with pixelcnn decoders. *CoRR*, abs/1606.05328.

[42] Yuan, X., He, P., Zhu, Q., Bhat, R. R., and Li, X. (2017). Adversarial examples: Attacks and defenses for deep learning. *CoRR*, abs/1712.07107.

[43] Zhang, C., Bütepage, J., Kjellström, H., and Mandt, S. (2017). Advances in variational inference. *CoRR*, abs/1711.05597.

[44] Zhao, H., Melibari, M., and Poupart, P. (2015). On the relationship between sum-product networks and bayesian networks. *CoRR*, abs/1501.01239.

Appendix A

L1 Regularization on SPN Sum Nodes

In this appendix, I demonstrate through an example why L1 regularization of sum node weights favors balanced probabilities

A.1 Example of L1 Regularization

Consider a set of binary observations $\mathscr{D} = \{0,0,1,0,0,1,0,...1\}$ from a random bernoulli distributed random variable X. We wish to infer the bernoulli parameter ρ in a bayesian framework. But instead of using a Beta distribution as a prior on ρ, we will reparameterise ρ in terms of w

$$\rho = f(w) = \frac{\exp(w)}{1+\exp(w)}$$

and place a laplace prior on w, which has the form:

$$p(w) = \frac{\exp(-\beta|w|)}{\int_{-\infty}^{\infty}\exp(-\beta|w|)dw} = \frac{\beta}{2}\exp(-\beta|w|) \tag{A.1}$$

According to the bernoulli distribution, the likelihood of observing $\mathscr{D}$ is

$$p(\mathscr{D}|w) = \prod_{i=1}^{N}\left(\frac{\exp(w)}{1+\exp(w)}\right)^{x_i}\left(\frac{1}{1+\exp(w)}\right)^{1-x_i} \tag{A.2}$$

Simplifying the likelihood using logarithms,

$$\log p(\mathscr{D}|w) = \sum_{i=1}^{N}\left[x_i w - \log(1+\exp(w))\right] \tag{A.3}$$

$$= nw - N\log(1+\exp(w)) \tag{A.4}$$

If we solve for the *maximum likelihood estimate* (MLE), we get the commonsense answer of $\rho_{MLE} = \frac{n}{N}$

$$\arg\max_{w} \log p(\mathscr{D}|w) \implies \frac{d}{dw}\log p(\mathscr{D}|w) = 0 \tag{A.5}$$

$$n - N\rho_{MLE} = 0 \tag{A.6}$$

Now, the posterior of w is given by Bayes' rule:

$$\log p(w|\mathscr{D}) = \log p(\mathscr{D}|w) + \log p(w) - \log p(\mathscr{D}) \tag{A.7}$$

$$= nw - N\log(1+\exp(w)) - \beta|w| + constant \tag{A.8}$$

If we solve for the *maximum a posteriori* (MAP) estimate, we discover the laplace prior on w encourages more balanced probabilities.

$$\arg\max_{w} \log p(w|\mathscr{D}) \implies \frac{d}{dw}\log p(w|\mathscr{D}) = 0 \tag{A.9}$$

$$n - N\rho - \beta = 0 \qquad \text{if } \rho > 0.5 \tag{A.10}$$

$$n - N\rho + \beta = 0 \qquad \text{if } \rho < 0.5 \tag{A.11}$$

$$\rho = \begin{cases} \frac{n-\beta}{N} = & \rho > 0.5 \\ \frac{n+\beta}{N} = & \rho < 0.5 \end{cases} \tag{A.12}$$

Here, ρ is pushed away from the MLE estimate toward 0.5.

A.2 Aside: Change of Variables on Laplace Prior

In the above example, significant amount of derivation is needed to answer the question of whether the L1 regularization favours balanced or sparsed probabilities. Perhaps the question can be more intuitively answered if we transform the prior on weight space into a prior on the simplex. Since ρ and w are related by a smooth invertible function f, this can be done via a change of variables:

$$p(\rho)d\rho = p(w)dw \tag{A.13}$$

$$p(\rho) = p(w)\frac{dw}{d\rho} \tag{A.14}$$

$$= \frac{\beta}{2}\exp(-\beta|w|)\frac{(1+\exp(w))^2}{\exp(w)} \tag{A.15}$$

$$= \frac{\beta}{2}\exp\left(-\beta\left|\log(\frac{\rho}{1-\rho})\right|\right)\frac{\rho/(1-\rho)}{\rho^2} \tag{A.16}$$

(A.17)

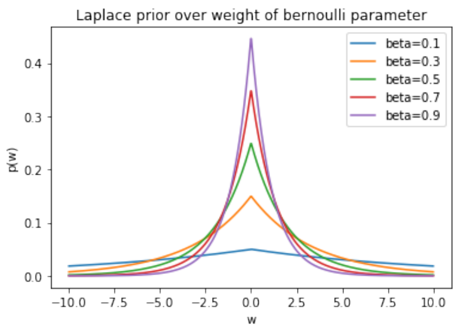


Fig. A.1 Laplace prior in w space $p(w)$ for different β

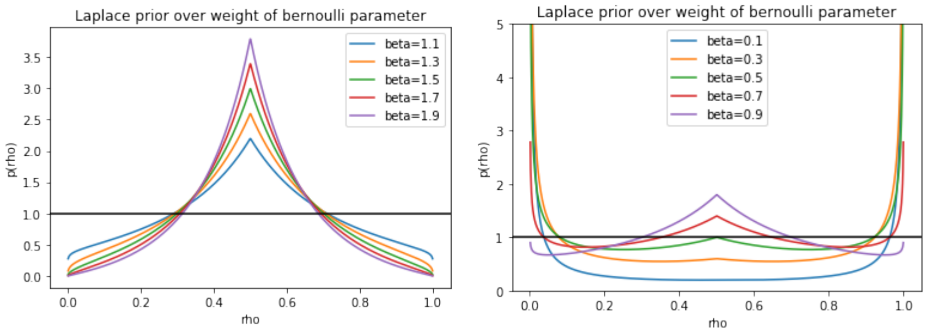


Fig. A.2 Laplace prior in w space for different β, after change of variables to ρ

$p(\rho)$ preserves the kink at $w = 0$. But now for small values of β, the maximum density is an undefined singularity at $\rho = 0$ or $\rho = 1$. This makes sense as many large values of w corresponds to almost the same ρ. Probability mass contained in vast regions of w is then compressed into a tiny region of ρ. Only when $\beta > 1$ will the laplace probability density decay faster than the compression effect.

A.3 Aside: MAP Estimation in Simplex Space

Now, $p(\rho)$ and $p(w)$ represent the same prior distribution over the same uncountably many Bernoulli models. However, they are have fundamentally different density types. $p(\rho)$ has closed support while $p(w)$ has infinite support. $p(w)$ has no singularities while $p(\rho)$ may. Consequently MAP inference in ρ space is different from MAP inference in w space.

$$\arg\max_{\rho} p(\rho|\mathscr{D}) \neq f(\arg\max_{w} p(w|\mathscr{D}))$$

This suggests that MAP inference is sensitive to the parameterization of priors and likelihoods. The only time when parameterization does not matter is when we are performing full bayesian inference.