



Leon Tiefenböck

Learning Probabilistic Circuits through sliced score matching

Bachelors's Thesis

to achieve the university degree of

Bachelor of Science

Bachelor's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Ass.-Prof. Dipl.-Ing. Dr. techn. Robert Peharz

Graz, October 2024

Abstract

Probabilistic Circuits (PCs) have shown to be a promising approach to probabilistic modelling, due to their many efficient and exact inference opportunities while still being complex enough to model arbitrary distributions. However they still are difficult to train and lack in results compared to other state of the art approaches in probabilistic modelling, especially with large high dimensional datasets. (Researchers fear that during the gradient based Maximum Likelihood optimization the algorithm gets stuck in local minima.) In this thesis I try to experimentally see if by using other learning objectives, namely sliced score matching, which recently gained prominence in learning energy based models (EBMs), we can work around these issues and achieve better results with PCs. I do this by training PCs using different algorithms on 2D and also high dimensional image data and comparing results.

Contents

Abstract	iii
1 Introduction	1
2 Background	2
2.1 Probabilistic Modelling	2
2.2 Probabilistic Circuits	3
2.3 Score Matching	5
2.4 Sliced Score Matching	5
3 Methods	7
4 Experimental Results	8
4.1 Dataset	8
5 Discussion	9
5.1 Interpretation of Experimental Results	9
6 Conclusions and Future Work	10
Bibliography	12

1 Introduction

2 Background

2.1 Probabilistic Modelling

At its very core Machine Learning (ML) aims to create algorithms/programs that learn from data in order to reason about it, make future predictions or perform all kinds of different inference tasks. One possible way to achieve this, especially when there is an inherent uncertainty in the data, which is the case for most real-world scenarios, is Probabilistic Modelling. In Probabilistic Modelling we use probabilities to express this uncertainty and the rules of probability theory for inference.

To make this more clear let's say we have some two dimensional data, the variables X and Y and assume that this data was drawn randomly from some unknown distribution. We would call this distribution $P^*(X, Y)$.

Definition 1 (joint distribution). A distribution over all input variables $P(X, Y)$ is called a joint distribution.

If the distribution $P^*(X, Y)$ was known to us all inference tasks would boil down to applying the basic rules of probability theory. The sum rule to compute marginals, the product rule to compute conditionals and more complex rules, derived from these two, to perform harder inference tasks.

However since we don't know $P^*(X, Y)$, we have to do something different, which leads us to a basic definition of probabilistic modelling: Through machine learning we create a framework that models $P(X, Y)$, approximating the unknown true distribution $P^*(X, Y)$, of which we only have limited samples, as closely as possible in order to reason about it with the rules of probability theory. [1].

Definition 2 (generative model). A framework that models a joint distribution $P(X, Y)$ is called a generative model.

Before continuing with different approaches to probabilistic modelling, I want to introduce two key concepts that are often used to discuss these different methods.

Definition 1: Expressiveness - a probabilistic model is called expressive if the learned distribution approximates the original distribution to a high degree. For instance this can be measured with Kullback-Leibler (KL) divergence [2], which measures how similar two probability distributions are. Two identical distributions would then have a KL-divergence of 0. In the Probabilistic Modelling case a KL-divergence of 0 would mean that the model is most expressive.

Definition 2: Tractability - a probabilistic model is called tractable with respect to an inference task, if the model can complete this task exactly and efficiently. Exactly in this case means without relying on approximation e.g. Monte-Carlo Simulation and efficiently means in linear time with respect to the model size.

In recent years expressive capability has increased considerably through models based on neural networks like Generative-Adversarial Networks (GANs) [3], Variational Autoencoder (VAE) [4] and many others. However what these models excel in for instance generating very realistic images or compelling text in language modelling, they lack in tractability for all except the most simplest inference tasks. [1]

On the other hand there are many mostly older, less complex models like Gaussian Mixture Models (GMMs), Hidden Markov Models (HMMs) and so on, that lack in expressiveness and only work well enough for very simple data but can compute most if not all inference tasks tractably.

2.2 Probabilistic Circuits

Probabilistic Circuit (PC) is an umbrella term for probabilistic models that can perform most inference tasks tractably but can be highly expressive at the same time. In general this is achieved by only introducing complexity in a structured manner. More specifically for a PC to be valid it has to adhere to certain structural properties, but more on that later.

Prominent members of the PC class include ..., but I will only be focusing on Sum Product Networks (SPNs) [5], which to my knowledge has seen the widest adoption.

In General one can think of a SPN as a structured neural network that consists of leaf nodes, sum nodes and product nodes. A SPN then recursively computes weighted mixtures, with sum nodes, or factorizations, with product nodes, from simple input distributions (leaf nodes). These inputs can basically be any probability distribution

like Gaussians, Bernoullies, a Categorical distribution and so on, however most of the time we will talk about Gaussians.

Considering this, one could notice that the simplest form of a SPN is just a basic Gaussian Mixture Model (GMM), where one sum node mixes two leaf nodes, as depicted in Figure 2.1.

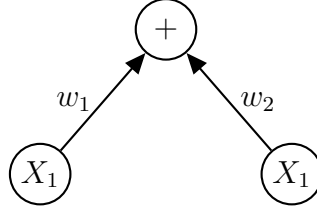


Figure 2.1: Simplest SPN

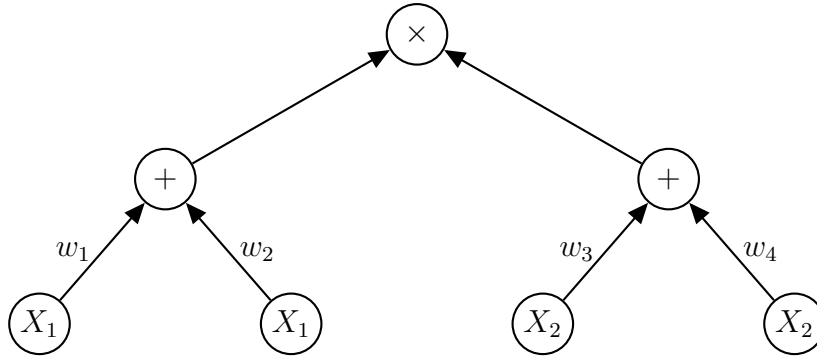


Figure 2.2: Simple SPN

Furthermore an only slightly more complex SPN with a product node and two sum node can be seen in Figure 2.2. Although this is still very very basic it is a great starting point to introduce and make sense of two central structural properties that allows SPNs to be expressive and tractable.

In the following Definitions PC and SPN can be used interchangeably.

Definition 3 (Scope). If a PC P models the joint distribution of a set of variables \mathbf{x} , each node of P models a distribution over a subset of \mathbf{x} . This subset is called the scope of the node. For a leaf node the scope is the input variable to that leaf, for all other nodes the scope is the union of its children's scopes. So the root node always has scope \mathbf{x} . [1]

Definition 4 (Smoothness). A sum node is *smooth* if all its inputs have identical scopes. A PC is smooth if all its sum nodes are smooth. [1]

Definition 5 (Decomposability). A product node is *decomposable* if all its input scopes do not share variables. A PC is decomposable if all of its product nodes are decomposable. [1]

In simple terms this basically means that sum nodes are only allowed to have inputs over the same variable and product nodes are only allowed to have inputs over different variables. The two depicted SPNs are smooth and decomposable.

There are more structural properties that a SPN or PC can fulfill, however these two already guarantee tractable computation of marginals (MAR) and conditionals (CON)[1], which many other models can't do and is already very useful in many scenarios. Generally the more structure a PC has, which means more structural properties it must fulfill, the more inference tasks it can compute tractably.

In [1] there is in depth explanation, why these properties guarantee tractable inference and further discussion on other properties.

Using a SPN in a real-world scenario then would basically entail first deciding on a structure (which properties and leaf distributions to use and how the nodes should be arranged in the graph) depending on the task at hand and then do a Maximum Likelihood Estimation with the weights of the sum nodes and the parameters of the leaf distributions (e.g. means and variances for a Gaussian). This optimization is usually done via Gradient Descent or the Expectation-Maximization (EM) algorithm, which is also very popular with Gaussian Mixture Models.

It is also noteworthy here that a SPN is a normalized model, which means it models a proper density (that integrates to 1 over the entire real space). This makes the optimization with Maximum Likelihood very straightforward as the model directly outputs the Likelihood when evaluated at one datapoint and we can just minimize the negative of this output, in turn maximizing the Likelihood. Though normally, as with most models, we model the log-likelihood for numerical stability.

2.3 Score Matching

2.4 Sliced Score Matching

Why score matching instead of minimizing NLL? should i even write about unnormalized models? how much detail for sm and ssm?

As described before, we want to learn the PDF of the data. In the case of PCs we directly model a normalized (integrates to 1) density but most other approaches model

a unnormalized density. This is because the normalization constant is very costly or impossible to compute in these scenarios.

In score matching [6]

Which leads to this score matching objective:

However this introduces another problem. With high dimensional data the trace of the hessian is very costly to compute.

Here sliced score matching [7] is introduced. Instead of doing the costly hessian calculation the multidimensional data is projected onto n random vectors (slices). The authors in the paper show that ..

Which leads to this final sliced score matching objective:

3 Methods

4 Experimental Results

4.1 Dataset

5 Discussion

5.1 Interpretation of Experimental Results

6 Conclusions and Future Work

Appendix

Bibliography

- [1] YooJung Choi, Antonio Vergari, and Guy Van den Broeck. “Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Models.” In: (Oct. 2020). URL: <http://starai.cs.ucla.edu/papers/ProbCirc20.pdf> (cit. on pp. 2–5).
- [2] S. Kullback and R. A. Leibler. “On Information and Sufficiency.” In: *The Annals of Mathematical Statistics* 22.1 (1951), pp. 79–86. ISSN: 00034851. URL: <http://www.jstor.org/stable/2236703> (visited on 09/12/2024) (cit. on p. 3).
- [3] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML]. URL: <https://arxiv.org/abs/1406.2661> (cit. on p. 3).
- [4] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2022. arXiv: 1312.6114 [stat.ML]. URL: <https://arxiv.org/abs/1312.6114> (cit. on p. 3).
- [5] Hoifung Poon and Pedro Domingos. *Sum-Product Networks: A New Deep Architecture*. 2012. arXiv: 1202.3732 [cs.LG]. URL: <https://arxiv.org/abs/1202.3732> (cit. on p. 3).
- [6] Aapo Hyvärinen and Peter Dayan. “Estimation of non-normalized statistical models by score matching.” In: *Journal of Machine Learning Research* 6.4 (2005) (cit. on p. 6).
- [7] Yang Song et al. “Sliced Score Matching: A Scalable Approach to Density and Score Estimation.” In: (2019). arXiv: 1905.07088 [cs.LG]. URL: <https://arxiv.org/abs/1905.07088> (cit. on p. 6).