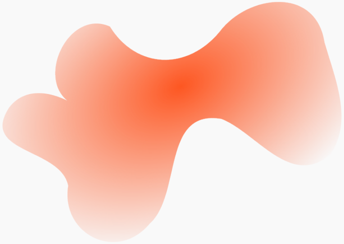
Probabilistic Circuits

Robert Peharz
Graz University of Technology

Generative Modeling Summer School
Copenhagen, 29th June 2023

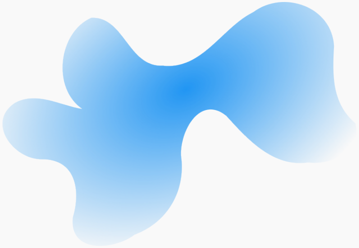
Generative Modeling in a Nutshell

Model Distribution $\mathbb{P}_{\boldsymbol{\theta}}$, $p_{\boldsymbol{\theta}}$



$\approx$

True Distribution $\mathbb{P}^*$, p^*



Why Probability?

1. Generative models are about "generating new data"—it's hard to do that without probability or randomness
2. **Probability is a consistent and optimal tool for reasoning under uncertainty**

Reasoning under Uncertainty

Cat vs. Dog classifier

If you know the true distribution $p^*(\boldsymbol{X}, Y)$ producing i.i.d. samples of $\boldsymbol{x}$ (image) and y (label), the **Bayes optimal classifier** is

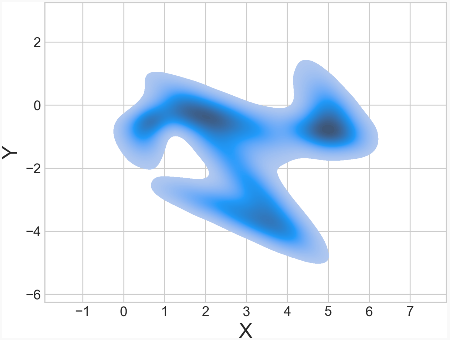
$$\hat{y} = \arg\max_y p^*(y \mid \boldsymbol{x}) = \arg\max_y p^*(y, \boldsymbol{x})$$

No classifier has a higher accuracy.

Reasoning under Uncertainty

Regression

Say you want to predict Y from X, and you know the true distribution $p^*(Y, X)$ producing i.i.d. samples of X, Y.

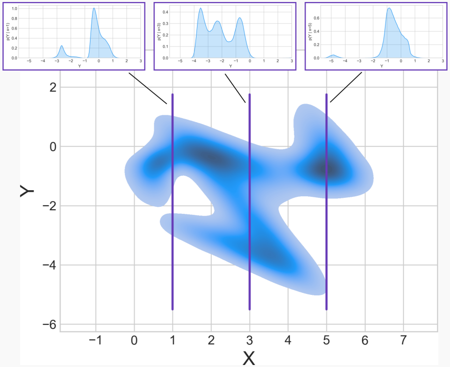


Reasoning under Uncertainty

You might compute the **conditional distribution**

$$p^*(Y \mid x) = \frac{p^*(Y, x)}{\underbrace{p^*(x)}_{=\int p^*(x,y)\,\mathrm{d}y}}$$

for all values of x.

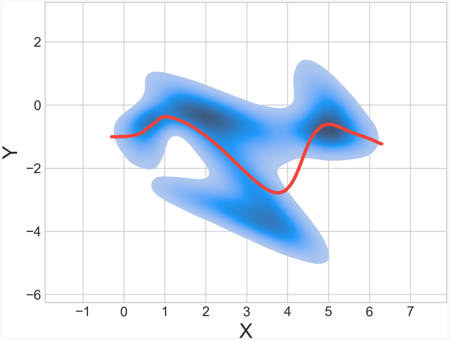


Then you can construct the **true regression function**

$$f(x) := \mathbb{E}_{p^*(Y \mid x)}[Y \mid x]$$

which is just the **expectation** of Y given x

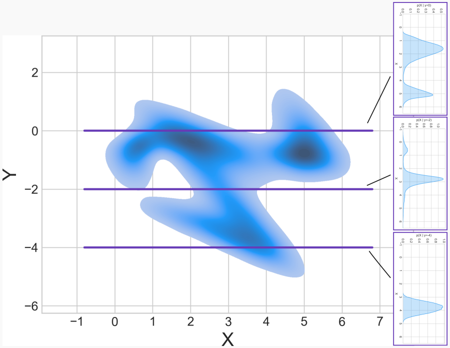
No function has lower squared loss.



Reasoning under Uncertainty

Of course, that Y is output and X is input was arbitrary, so the whole "trick" works also in the other direction:

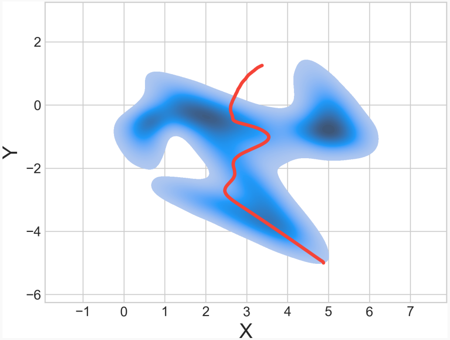
$$p^*(X \mid y) = \frac{p^*(y, X)}{\underbrace{p^*(y)}_{= \int p^*(x,y)\,\mathrm{d}x}}$$



Reasoning under Uncertainty

Of course, that Y is output and X is input was arbitrary, so the whole "trick" works also in the other direction:

$$f(y) := \mathbb{E}_{p^*(X \mid y)}[X \mid y]$$



Probability = consistent and optimal reasoning under uncertainty

"Probability is nothing but common sense reduced to computation."

—P.S. Laplace

"Probabilistic reasoning is about halfway to AI."

—R. Peharz 😉

see also Pearl, Jaynes, Cox Theorem

Inference Routines

Joint probability distribution $\approx$ **knowledge base** + uncertainty

There are many **inference routines** which might be applied (and combined) to answer **queries** from your joint:

sampling	$\boldsymbol{x} \sim p(\boldsymbol{X})$	
density, likelihood	$p(\boldsymbol{x})$	
marginalization	$p(\boldsymbol{X}) = \int p(\boldsymbol{X}, \boldsymbol{y})\,\mathrm{d}\boldsymbol{y}$	($\sum$ in discrete case)
conditioning	$p(\boldsymbol{Y} \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x}, \boldsymbol{Y})}{p(\boldsymbol{x})}$	
expectations	$\mathbb{E}_{p(\boldsymbol{X})}[f(\boldsymbol{X})]$	(e.g. **moments**)
maximization (MAP)	$\arg\max_{\boldsymbol{x}} p(\boldsymbol{x})$	

Inference Routines

Joint probability distribution $\approx$ **knowledge base** + uncertainty

There are many **inference routines** which might be applied (and combined) to answer **queries** from your joint:

sampling	$\boldsymbol{x} \sim p(\boldsymbol{X})$	
density, likelihood	$p(\boldsymbol{x})$	
marginalization	$p(\boldsymbol{X}) = \int p(\boldsymbol{X}, \boldsymbol{y})\,\mathrm{d}\boldsymbol{y}$	($\sum$ in discrete case)
conditioning	$p(\boldsymbol{Y} \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x}, \boldsymbol{Y})}{p(\boldsymbol{x})}$	
expectations	$\mathbb{E}_{p(\boldsymbol{X})}[f(\boldsymbol{X})]$	(e.g. **moments**)
maximization (MAP)	$\arg\max_{\boldsymbol{x}} p(\boldsymbol{x})$	

Can your generative model do this?

Inference Routines

For which model class and inference routine does there exist an **exact** and **efficient** algorithm?

	GANs	VAEs	EBMs	Flows	ARMs	PCs
sampling	✓	✓	✓	✓	✓	✓
density	✗	✗	✗	✓	✓	✓
marginals	✗	✗	✗	✗	✗	✓
condition	✗	✗	✗	✗	✗	✓
moments	✗	✗	✗	✗	✗	✓
max (MAP)	✗	✗	✗	✗	✗	✓ (✗)
$\mathbb{E}$	✗	✗	✗	✗	✗	✓ (✗)

PCs can do all these amazing things. Where's the catch?

- still many interesting queries which are intractable in PCs
- **tractability-vs-expressivity dilemma**
  - expressive models are less tractable
  - tractable models are less expressive
- **however, don't jump to conclusion!**
  - tractable models get you surprisingly far
  - we have by far not reached their true potential
  - ultimately, it's the right mix of methods for the win

Probabilistic Circuits: Definitions

What are Probabilistic Circuits?

- **probabilistic circuits (PCs)**: special type of neural network
- they represent a density $p_{\boldsymbol{\theta}}(\boldsymbol{X})$ for given random variables $\boldsymbol{X}$
- basic mode of operation: "sample $\boldsymbol{x}$ in, $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ out"
- inference routines (marginals, conditionals, etc.) are computed via "modified" network passes
- in practice, implemented in the **log-domain** ($\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$)

Three Types of Units

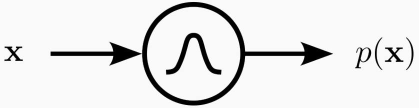
distribution node, **input node**, **leaf**

⊕ **sum node**, **mixture**

⊗ **product node**, **factorization**

Distribution Node, Input Node, Leaf

- a **distribution node**, **input node**, **leaf** L is some “simple” tractable distribution function, e.g. Gaussian, Categorical, etc.



- input: sample $\boldsymbol{x}$
- output: $p(\boldsymbol{x})$, density evaluated at $\boldsymbol{x}$
- density as special case of “non-linear activation function”
- **Important:** a leaf is usually a distribution over only a subset of all variables, denoted as **scope** $sc(L)$
- the scope as like a “receptive field” in neural nets

Gaussian Leaves

Example

Parameters $\boldsymbol{\mu} = (\mu_1, \dots, \mu_D)$, $\Sigma = \begin{pmatrix} v_{11} & \cdots & v_{1D} \\ \vdots & \ddots & \vdots \\ v_{D1} & \cdots & v_{DD} \end{pmatrix}$

All kinds of tractable inference routines: ❤

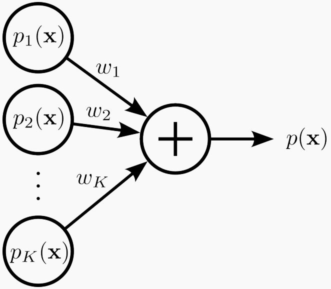
- **sampling:** via eigen decomposition of Σ
- **density:** $p(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^D|\Sigma|}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{\mu})\right)$
- **marginals:** simply discard rows/columns from μ and Σ
- **conditionals:** $$\boldsymbol{\mu}_{q|e} = \boldsymbol{\mu}_q + \Sigma_{qe}\Sigma_{ee}^{-1}(\boldsymbol{x}_e - \boldsymbol{\mu}_e)$$ $$\Sigma_{q|e} = \Sigma_{qq} + \Sigma_{qe}\Sigma_{ee}^{-1}\Sigma_{eq}$$
- **mean, (co)variance:** $\boldsymbol{\mu}$, Σ
- **max:** $\boldsymbol{\mu}$

Sum Node, Mixture

- let p_1, p_2, . . . , p_K be **components**, i.e. distributions over the same scope
- let w_1, w_2, . . . , w_K be **weights** with $w_k \geq 0$, $\sum_k w_k = 1$
- a **sum node**, **mixture** S computes the distribution

$$p(\boldsymbol{X}) = \sum_k w_k \, p_k(\boldsymbol{X})$$

- for any $p_1, \ldots, p_K$, the mixture p is a proper distribution



Mixtures inherit tractability!

- assume $p_1, \ldots, p_K$ have tractable marginals and conditionals
- then also the mixture distribution is tractable
- **Marginals:** assume we want to marginalize X_i

$$\begin{aligned} p(\boldsymbol{X}_{\setminus i}) &= \int \sum_k w_k \, p_k(\boldsymbol{X}_{\setminus i}, x_i) \, \mathrm{d}x_i \\ &= \sum_k w_k \underbrace{\int p_k(\boldsymbol{X}_{\setminus i}, x_i) \, \mathrm{d}x_i}_{\text{tractable}} = \sum_k w_k \, p_k(\boldsymbol{X}_{\setminus i}) \end{aligned}$$

- of course, this also works with many variables
- "marginal of mixture = mixture of component marginals"

Mixtures inherit tractability! Cont'd

- **Conditionals:** say we want to compute $p(\boldsymbol{X}_q \mid \boldsymbol{x}_e)$

$$p(\boldsymbol{X}_q \mid \boldsymbol{x}_e) \propto p(\boldsymbol{X}_q, \boldsymbol{x}_e) = \sum_k w_k \, p_k(\boldsymbol{X}_q, \boldsymbol{x}_e)$$

$$= \sum_k \overbrace{w_k p_k(\boldsymbol{x}_e)}^{\text{unnormalized weight}} \underbrace{p_k(\boldsymbol{X}_q \mid \boldsymbol{x}_e)}_{\text{tractable, normalized}}$$

- hence, by setting $\tilde{w}_k = \frac{w_k p_k(\boldsymbol{x}_e)}{\sum_l w_l p_l(\boldsymbol{x}_e)}$ we get

$$p(\boldsymbol{X}_q \mid \boldsymbol{x}_e) = \sum_k \tilde{w}_k \, p_k(\boldsymbol{X}_q \mid \boldsymbol{x}_e)$$

- "conditional of mixture $\triangleq$ mixture of component conditionals"

Mixtures inherit tractability! Cont'd

- **Maximization:** in general intractable in mixture models, even if components allow tractable maximization...
- however, if the **supports** of the components don't overlap, maximization becomes tractable
- specifically, if for each $\boldsymbol{x}$ we have that at most one $p_k(\boldsymbol{x}) > 0$, we say the mixture is **deterministic**
- then, we can write

$$\max_{\boldsymbol{x}} \sum_k w_k \, p_k(\boldsymbol{x}) = \max_{\boldsymbol{x}} \overbrace{\max_k}^{\text{deterministic sum}} w_k \, p_k(\boldsymbol{x}) = \max_k w_k \overbrace{\max_{\boldsymbol{x}} p_k(\boldsymbol{x})}^{\text{tractable}}$$

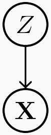
- we can also keep track of the *arg max*
- "max of mixture = mixture of component max"

Mixtures inherit tractability! Cont'd

- **Sampling:** a mixture model can be seen as a **latent variable model**, involving a marginalized latent variable Z

$$p(\boldsymbol{X}) = \sum_k \overbrace{p(Z = k)}^{w_k} \, p(\boldsymbol{X} \mid Z = k) = \sum_k p(Z = k, \boldsymbol{X})$$

- Sampling can be done in two stages:
    - sample $Z \sim \mathit{Categorical}(w_1, \ldots, w_K)$
    - sample from selected component $\boldsymbol{x} \sim p_Z$



- "sample of mixture = selecting among component samples"

Mixtures inherit tractability! Cont'd

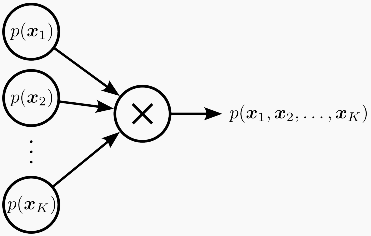
Inference in mixtures reduces to inference at components!

Product Node, Factorization

- let $p(\boldsymbol{X}_1)$, $p(\boldsymbol{X}_2)$, ..., $p(\boldsymbol{X}_K)$ be distributions over **disjoint** sets of variables $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_K$
- a **product node** P computes the distribution

$$p(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_K) = \prod_k p(\boldsymbol{X}_k)$$

- that is, a product node is simply a **factorized** distribution over $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_K\}$



Factorizations inherit tractability!

- let $p(\boldsymbol{X}_1), \dots, p(\boldsymbol{X}_K)$ have tractable marginals and conditionals
- then also the factorized distribution is tractable
- **Marginals:** assume we want to marginalize X_i
- since $\boldsymbol{X}_1, \boldsymbol{X}_2, \dots, \boldsymbol{X}_K$ are disjoint, X_i only appears in one set, say $\boldsymbol{X}_j$

$$\begin{aligned} p(\boldsymbol{X}_{\setminus i}) &= \int \left(\prod_{k \neq j} p(\boldsymbol{X}_k) \right) p(\boldsymbol{X}_{j \setminus i}, x_i) \, \mathrm{d}x_i \\ &= \left(\prod_{k \neq i} p(\boldsymbol{X}_k) \right) \overbrace{\int p(\boldsymbol{X}_{j \setminus i}, x_i) \, \mathrm{d}x_i}^{\text{tractable}} \end{aligned}$$

- of course, marginalization also works with many variables
- "marginal of product = product of marginals"

Factorizations inherit tractability! cont'd

- **Conditionals:** say we want to compute $p(\boldsymbol{X}_q \mid \boldsymbol{x}_e)$
- let $\boldsymbol{X}_{qk} = \boldsymbol{X}_q \cap \boldsymbol{X}_k$ and $\boldsymbol{X}_{ek} = \boldsymbol{X}_e \cap \boldsymbol{X}_k$

$$p(\boldsymbol{X}_q \mid \boldsymbol{x}_e) = \frac{p(\boldsymbol{X}_q, \boldsymbol{x}_e)}{p(\boldsymbol{x}_e)} = \frac{\prod_k p(\boldsymbol{X}_{qk}, \boldsymbol{x}_{ek})}{\prod_k p(\boldsymbol{x}_{ek})} = \prod_k p(\boldsymbol{X}_{qk} \mid \boldsymbol{x}_{ek})$$

- "conditionals of product = product of conditionals"

Factorizations inherit tractability! cont'd

- **Maximization:**

$$\max_{\boldsymbol{x}} \prod_k p(\boldsymbol{x}_k) = \prod_k \max_{\boldsymbol{x}_k} p(\boldsymbol{x}_k)$$

- arg max of a product is simply the concatenation of the arg max'es of the individual factors
- "max of product = product of max"

- **Sampling:**
- just independently sample factors
- "sampling of product = concatenate samples of factors"

Factorizations inherit tractability! Cont'd

Inference in factorizations reduces to inference at factors!

Three Types of Units

∩	**distribution node**	tractable inference by default
⊕	**sum node**	inference reduces to inference in components
⊗	**product node**	inference reduces to inference in factors

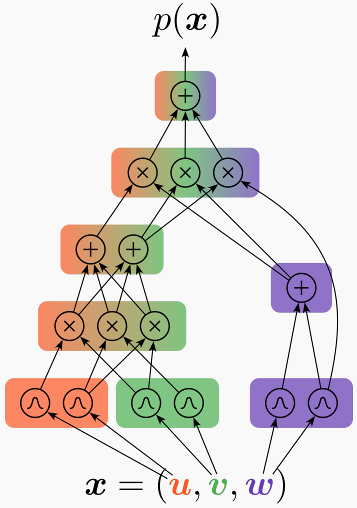
- **probabilistic circuits (PCs)** are networks of these units
- by induction, all nodes are a distribution over their scope
- inference reduces to inference at the leaves!

Smoothness and Decomposability

Leaves are distributions (L), internal nodes are sums (S) or products (P).

A PC is **smooth**, if for all sum nodes it holds, that all input nodes are over same scope—that just means that all sums are proper mixtures.

A PC is **decomposable**, if for all product nodes it holds, that all input nodes have disjoint scope—that just means that all products are proper factorizations.


$p(\boldsymbol{x})$
$\boldsymbol{x} = (\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$

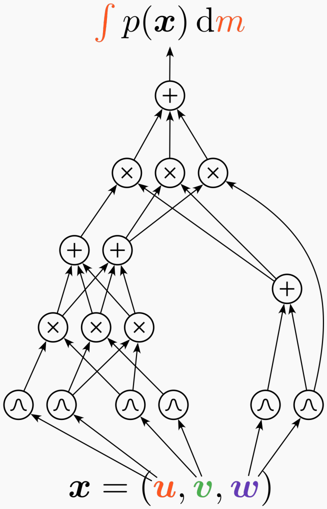
Inference in PCs

- the inference "tricks" for mixtures and products recursively apply to PCs
- ultimately, inference in PCs reduces to inference at the leaves (plus some modifications of parameters/structures)
- think of this as a neural network with "many modes of operation"
- **smoothness** and **decomposability** enable tractable marginalization and conditioning
- **determinism** enables tractable maximization
- there are more interesting constraints and corresponding inference routines

Marginalization

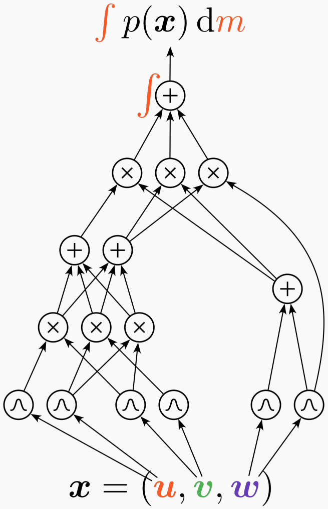
Example

Assume we want to marginalize a variable M, which is contained in $\boldsymbol{U}$

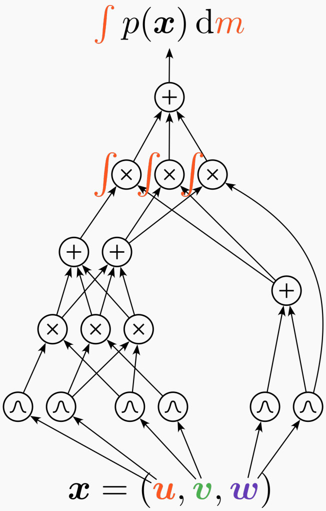


$\int p(\boldsymbol{x})\,\mathrm{d}m$
$\boldsymbol{x} = (\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$

Marginalization


$\int p(\boldsymbol{x})\,\mathrm{d}m$
$\int$
$\boldsymbol{x} = (\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$

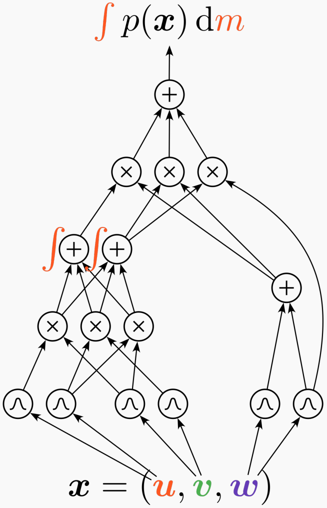
Marginalization


$\int p(\boldsymbol{x})\,\mathrm{d}m$
$\int$
$\int$
$\int$
$\boldsymbol{x} = (\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$

Marginalization

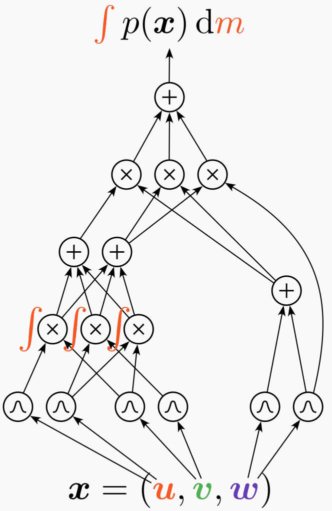
Example

Due to decomposability, M appears only in one child of each product node



$\int p(\boldsymbol{x})\,\mathrm{d}m$
$\int$
$\int$
$\boldsymbol{x} = (\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$

Marginalization

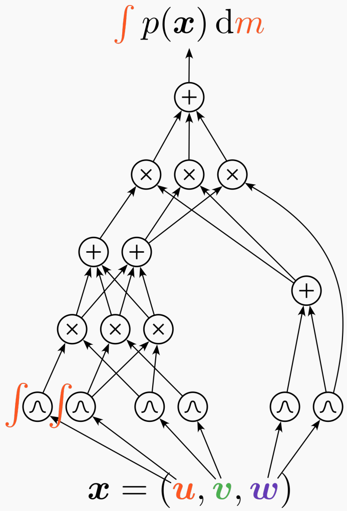

$\int p(\boldsymbol{x})\,\mathrm{d}m$
$\int$
$\int$
$\int$
$\boldsymbol{x} = (\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$

Marginalization

Example

Reduces to marginalization at leaves + standard forward pass!

Special case $\boldsymbol{U} = \{M\}$: marginalization at leaf delivers constant 1



$\int p(\boldsymbol{x})\,\mathrm{d}m$
$\boldsymbol{x} = (\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$

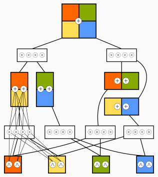
Learning PCs

- PCs are between neural nets and probabilistic graphical models
- learning techniques are carry over from these approaches
- **Parameters:**
  - closed form maximum likelihood (deterministic PCs)
  - gradient based maximum likelihood
  - expectation-maximization

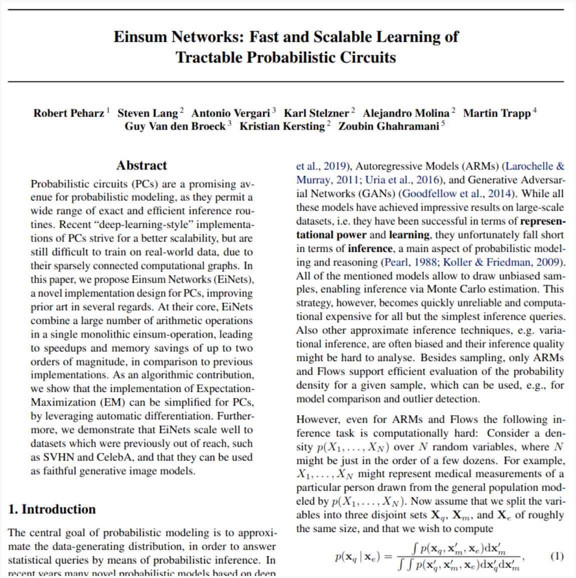
    Peharz et al. "On the latent variable interpretation in Sum-Product Networks"
  - Bayesian
- **Structure:**
  - compile from other models (e.g. Bayesian networks)

    Poon & Domingos, "Sum-Product Networks: A New Deep Architecture"
  - hand-designed inductive bias (cf. convolutions)

    Darwiche, "A Differential Approach to Inference in Bayesian Networks"
  - structure learning
    - top-down clustering
    - decision tree learning



Selected Recent Work



Einsum Networks: Fast and Scalable Learning of Tractable Probabilistic Circuits

Robert Peharz[1] Steven Lang[2] Antonio Vergari[3] Karl Stelzner[2] Alejandro Molina[2] Martin Trapp[4] Guy Van den Broeck[3] Kristian Kersting[2] Zoubin Ghahramani[5]

Abstract

Probabilistic circuits (PCs) are a promising avenue for probabilistic modeling, as they permit a wide range of exact and efficient inference routines. Recent "deep-learning-style" implementations of PCs strive for a better scalability, but are still difficult to train on real-world data, due to their sparsely connected computational graphs. In this paper, we propose Einsum Networks (EiNets), a novel implementation design for PCs, improving prior art in several regards. At their core, EiNets combine a large number of arithmetic operations in a single monolithic einsum-operation, leading to speedups and memory savings of up to two orders of magnitude, in comparison to previous implementations. As an algorithmic contribution, we show that the implementation of Expectation-Maximization (EM) can be simplified for PCs, by leveraging automatic differentiation. Furthermore, we demonstrate that EiNets scale well to datasets which were previously out of reach, such as SVHN and CelebA, and that they can be used as faithful generative image models.

1. Introduction

The central goal of probabilistic modeling is to approximate the data-generating distribution, in order to answer statistical queries by means of probabilistic inference. In recent years many novel probabilistic models based on deep

et al., 2019), Autoregressive Models (ARMs) (Larochelle & Murray, 2011; Uria et al., 2016), and Generative Adversarial Networks (GANs) (Goodfellow et al., 2014). While all these models have achieved impressive results on large-scale datasets, i.e. they have been successful in terms of **representational power** and **learning**, they unfortunately fall short in terms of **inference**, a main aspect of probabilistic modeling and reasoning (Pearl, 1988; Koller & Friedman, 2009). All of the mentioned models allow to draw unbiased samples, enabling inference via Monte Carlo estimation. This strategy, however, becomes quickly unreliable and computational expensive for all but the simplest inference queries. Also other approximate inference techniques, e.g. variational inference, are often biased and their inference quality might be hard to analyse. Besides sampling, only ARMs and Flows support efficient evaluation of the probability density for a given sample, which can be used, e.g., for model comparison and outlier detection.

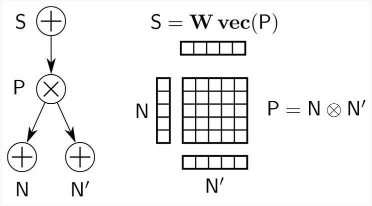
However, even for ARMs and Flows the following inference task is computationally hard: Consider a density $p(X_1, \dots, X_N)$ over N random variables, where N might be just in the order of a few dozens. For example, $X_1, \dots, X_N$ might represent medical measurements of a particular person drawn from the general population modeled by $p(X_1, \dots, X_N)$. Now assume that we split the variables into three disjoint sets $\mathbf{X}_q$, $\mathbf{X}_m$, and $\mathbf{X}_e$ of roughly the same size, and that we wish to compute

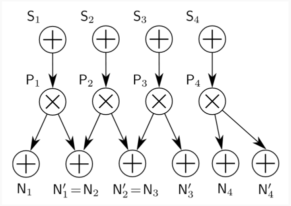
$$p(\mathbf{x}_q \mid \mathbf{x}_e) = \frac{\int p(\mathbf{x}_q, \mathbf{x}'_m, \mathbf{x}_e) \mathrm{d}\mathbf{x}'_m}{\int \int p(\mathbf{x}'_q, \mathbf{x}'_m, \mathbf{x}_e) \mathrm{d}\mathbf{x}'_q \mathrm{d}\mathbf{x}'_m}, \qquad (1)$$

`https://github.com/cambridge-mlg/EinsumNetworks` ICML'20

Einsum Networks (EiNets)

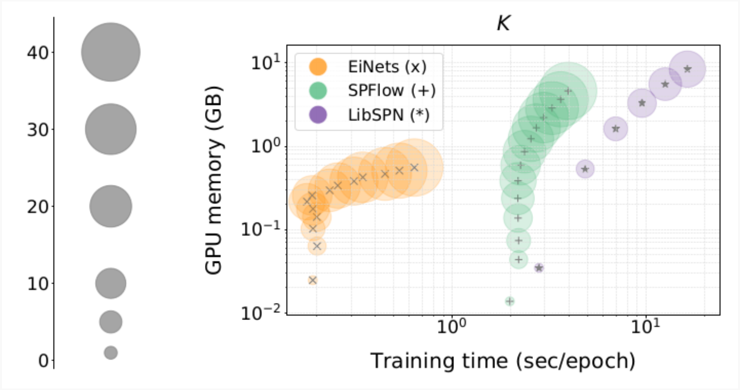
- PCs used to be slow, even on Tensorflow, PyTorch, etc.
- reason: sparse and cluttered computational graph, yielding many small calls to GPU kernels
- **idea:** summarize many sum-product operations in one layer





- all nodes on same level computed with a single GPU call: `einsum('bip,bjp,ijop -> bop', leftx, rightx, params)`

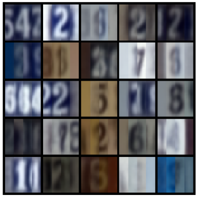
Einsum Networks – Runtime and Memory

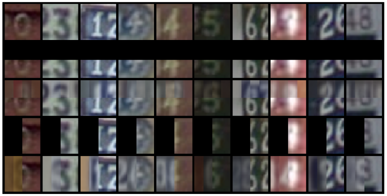


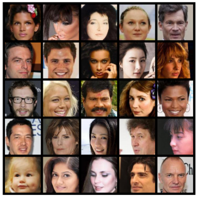
- runtime and memory comparison on random structures
- model sizes range from 10k-10M
- **improvements of 1-2 orders of magnitude**
- PCs now similarly efficient as DNNs

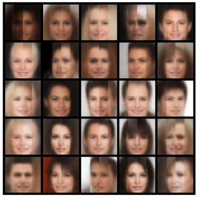
Einsum Networks as Image Models

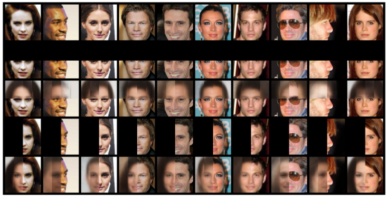
(a) Real SVHN images.


(b) EiNet SVHN samples.


(c) Real images (top), covered images, and EiNet reconstructions


(d) Real CelebA samples.


(e) EiNet CelebA samples.


(f) Real images (top), covered images, and EiNet reconstructions

Einsum Networks for Outlier Detection

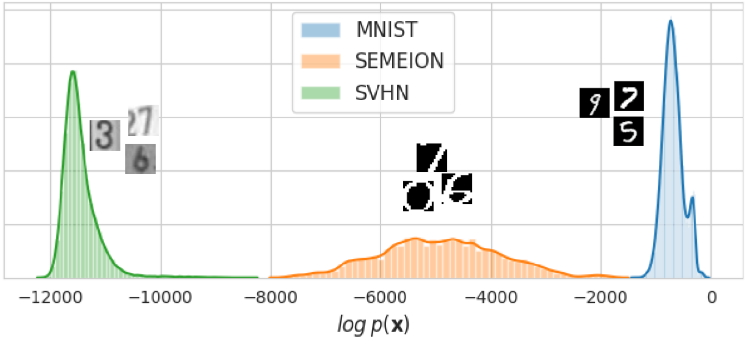
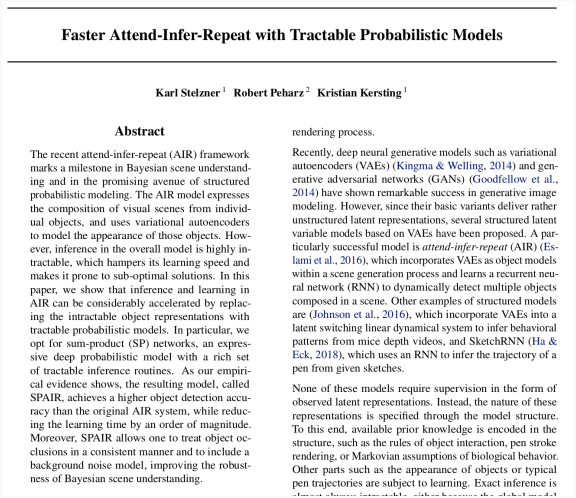


Figure 6. Histograms of sample-wise log-probabilities of the test sets of MNIST, SEMEION and SVHN, under EiNet trained on MNIST training data. Note that the histograms do not overlap.


Faster Attend-Infer-Repeat with Tractable Probabilistic Models

Karl Stelzner[1] Robert Peharz[2] Kristian Kersting[1]

Abstract

The recent attend-infer-repeat (AIR) framework marks a milestone in Bayesian scene understanding and in the promising avenue of structured probabilistic modeling. The AIR model expresses the composition of visual scenes from individual objects, and uses variational autoencoders to model the appearance of those objects. However, inference in the overall model is highly intractable, which hampers its learning speed and makes it prone to sub-optimal solutions. In this paper, we show that inference and learning in AIR can be considerably accelerated by replacing the intractable object representations with tractable probabilistic models. In particular, we opt for sum-product (SP) networks, an expressive deep probabilistic model with a rich set of tractable inference routines. As our empirical evidence shows, the resulting model, called SPAIR, achieves a higher object detection accuracy than the original AIR system, while reducing the learning time by an order of magnitude. Moreover, SPAIR allows one to treat object occlusions in a consistent manner and to include a background noise model, improving the robustness of Bayesian scene understanding.

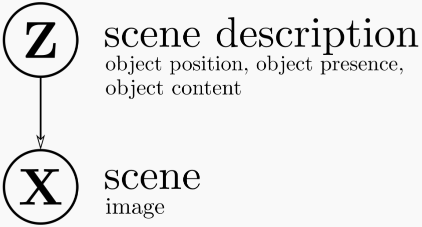
rendering process.

Recently, deep neural generative models such as variational autoencoders (VAEs) (Kingma & Welling, 2014) and generative adversarial networks (GANs) (Goodfellow et al., 2014) have shown remarkable success in generative image modeling. However, since their basic variants deliver rather unstructured latent representations, several structured latent variable models based on VAEs have been proposed. A particularly successful model is *attend-infer-repeat* (AIR) (Eslami et al., 2016), which incorporates VAEs as object models within a scene generation process and learns a recurrent neural network (RNN) to dynamically detect multiple objects composed in a scene. Other examples of structured models are (Johnson et al., 2016), which incorporate VAEs into a latent switching linear dynamical system to infer behavioral patterns from mice depth videos, and SketchRNN (Ha & Eck, 2018), which uses an RNN to infer the trajectory of a pen from given sketches.

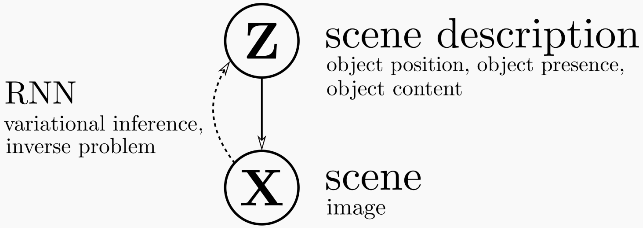
None of these models require supervision in the form of observed latent representations. Instead, the nature of these representations is specified through the model structure. To this end, available prior knowledge is encoded in the structure, such as the rules of object interaction, pen stroke rendering, or Markovian assumptions of biological behavior. Other parts such as the appearance of objects or typical pen trajectories are subject to learning. Exact inference is almost always intractable, either because the global model

https://github.com/stelzner/supair [ICML'19]

Attend-Infer-Repeat – Vision as Inverse Graphics



Attend-Infer-Repeat – Vision as Inverse Graphics


Z
scene description
object position, object presence, object content
RNN
variational inference, inverse problem
X
scene
image

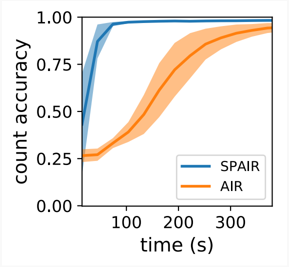
- Attend-Infer-Repeat (AIR) based on **variational inference over thousands of variables** (scene parameters, pixels, latent codes)
- we replaced the vision models with PCs and "collapsed them out" of the posterior, leaving a posterior over **only a few dozens of latent variables** (scene parameters)
- **hybrid between PCs and variational inference**

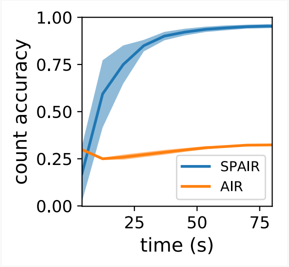
Learning Speed and Stability

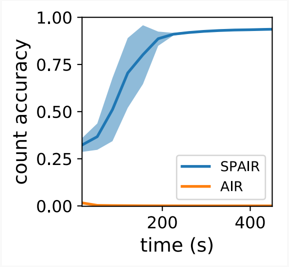
Multi-MNIST

Sprites

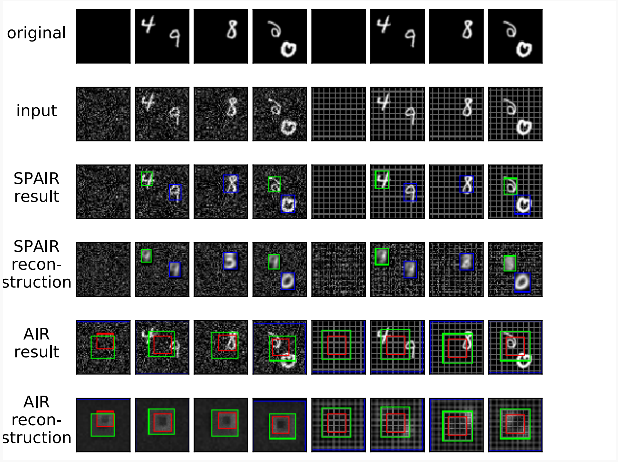
Noisy MNIST





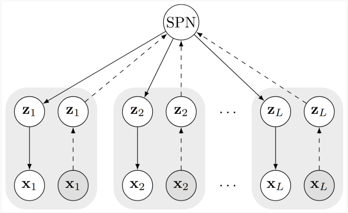


Background and Reconstructions



More Hybrids. . .

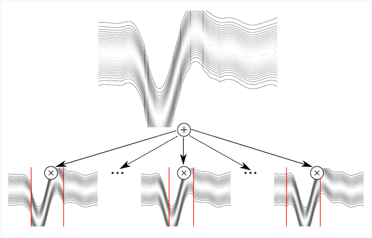
- **Tan and Peharz, "Hierarchical Decompositional Mixtures of Variational Autoencoders"**
  - PCs with VAE leaves
  - main insight: PCs can incorporate noisy ELBOs in a meaningful way
  - VAE leaves are over few dimensions $\rightarrow$ "easier inference problem"
  - delivered higher likelihoods than vanilla VAEs



`https://github.com/cambridge-mlg/SPVAE` ICML'19

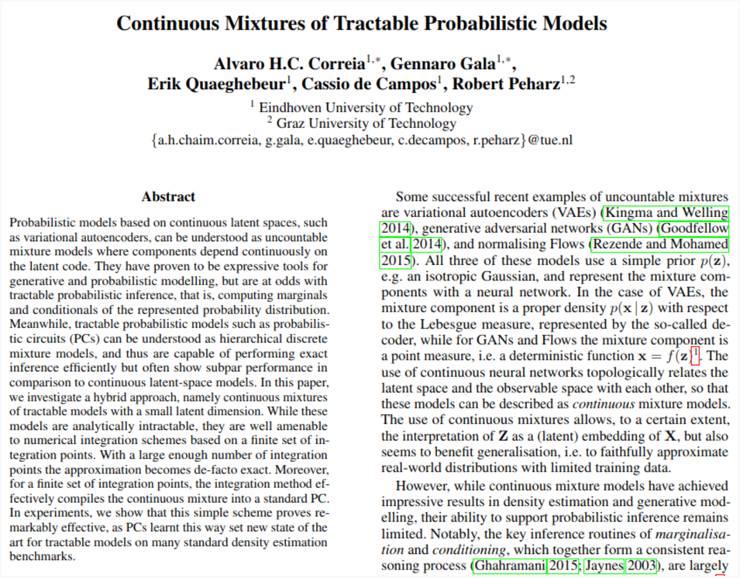
More Hybrids. . .

- **Trapp, et al., "Deep Structured Mixtures of Gaussian Processes"**
  - PCs over K Gaussian process experts
  - well defined mixture of GPs with exact inference
  - reduces cubic inference cost of GPs by divide-and-conquer approach



`https://github.com/trappmartin/DeepStructuredMixtures`

AISTATS'20



Continuous Mixtures of Tractable Probabilistic Models

Alvaro H.C. Correia[1,*], **Gennaro Gala**[1,*],
Erik Quaeghebeur[1], **Cassio de Campos**[1], **Robert Peharz**[1,2]

[1] Eindhoven University of Technology
[2] Graz University of Technology
{a.h.chaim.correia, g.gala, e.quaeghebeur, c.decampos, r.peharz}@tue.nl

Abstract

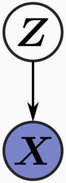
Probabilistic models based on continuous latent spaces, such as variational autoencoders, can be understood as uncountable mixture models where components depend continuously on the latent code. They have proven to be expressive tools for generative and probabilistic modelling, but are at odds with tractable probabilistic inference, that is, computing marginals and conditionals of the represented probability distribution. Meanwhile, tractable probabilistic models such as probabilistic circuits (PCs) can be understood as hierarchical discrete mixture models, and thus are capable of performing exact inference efficiently but often show subpar performance in comparison to continuous latent-space models. In this paper, we investigate a hybrid approach, namely continuous mixtures of tractable models with a small latent dimension. While these models are analytically intractable, they are well amenable to numerical integration schemes based on a finite set of integration points. With a large enough number of integration points the approximation becomes de-facto exact. Moreover, for a finite set of integration points, the integration method effectively compiles the continuous mixture into a standard PC. In experiments, we show that this simple scheme proves remarkably effective, as PCs learnt this way set new state of the art for tractable models on many standard density estimation benchmarks.

Some successful recent examples of uncountable mixtures are variational autoencoders (VAEs) (Kingma and Welling 2014), generative adversarial networks (GANs) (Goodfellow et al. 2014), and normalising Flows (Rezende and Mohamed 2015). All three of these models use a simple prior $p(\mathbf{z})$, e.g. an isotropic Gaussian, and represent the mixture components with a neural network. In the case of VAEs, the mixture component is a proper density $p(\mathbf{x} \mid \mathbf{z})$ with respect to the Lebesgue measure, represented by the so-called decoder, while for GANs and Flows the mixture component is a point measure, i.e. a deterministic function $\mathbf{x} = f(\mathbf{z})$[1]. The use of continuous neural networks topologically relates the latent space and the observable space with each other, so that these models can be described as *continuous* mixture models. The use of continuous mixtures allows, to a certain extent, the interpretation of $\mathbf{Z}$ as a (latent) embedding of $\mathbf{X}$, but also seems to benefit generalisation, i.e. to faithfully approximate real-world distributions with limited training data.

However, while continuous mixture models have achieved impressive results in density estimation and generative modelling, their ability to support probabilistic inference remains limited. Notably, the key inference routines of *marginalisation* and *conditioning*, which together form a consistent reasoning process (Ghahramani 2015; Jaynes 2003), are largely

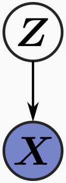
Discrete vs. Continuous Mixtures

Discrete Mixture



$$p(\boldsymbol{x}) = \sum_k \overbrace{p(Z = k)}^{w_k}\, p(\boldsymbol{x} \mid Z = k)$$

Continuous Mixture

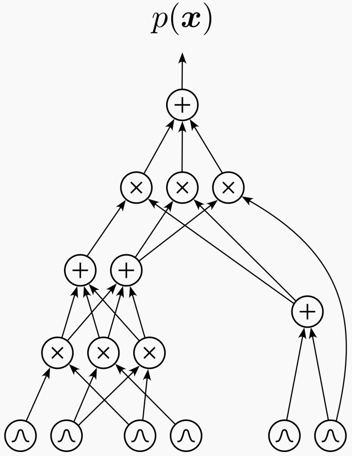


$$p(\boldsymbol{x}) = \int p(\boldsymbol{z})\, p(\boldsymbol{x} \mid \boldsymbol{z})\, \mathrm{d}\boldsymbol{z}$$

- most well known continuous mixture model: VAEs
- trouble of continuous mixtures: solving the integral over $\boldsymbol{Z}$.
- VAEs address this with a variational approach/the ELBO

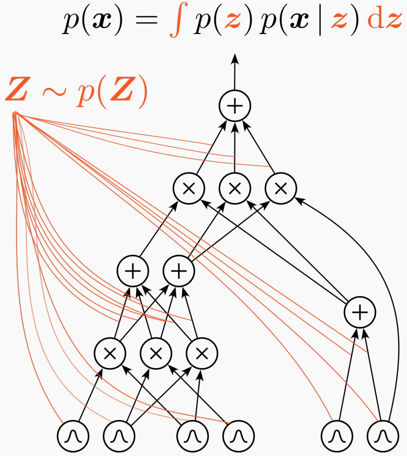
Continuous Mixtures of PCs

From standard PCs. . .



Continuous Mixtures of PCs

. . . to continuous PCs mixtures



$p(\boldsymbol{x}) = \int p(\boldsymbol{z})\, p(\boldsymbol{x} \mid \boldsymbol{z})\, \mathrm{d}\boldsymbol{z}$
$\boldsymbol{Z} \sim p(\boldsymbol{Z})$

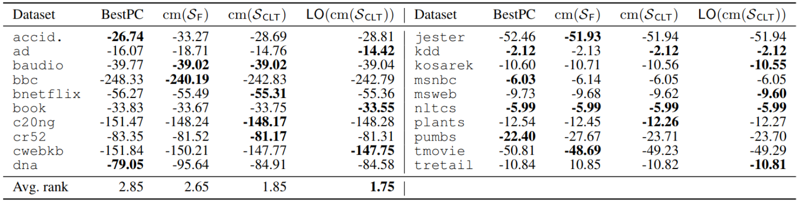
Numerical Integration

- high-dimensional integrals are hard
- however, if we restrict the dimensionality of $\boldsymbol{Z}$, we can use numerical integration techniques such as
  - (randomized quasi) Monte Carlo
  - quadrature rules
  - sparse grids
- all these numerical integration techniques approximate the integral with a finite mixture:

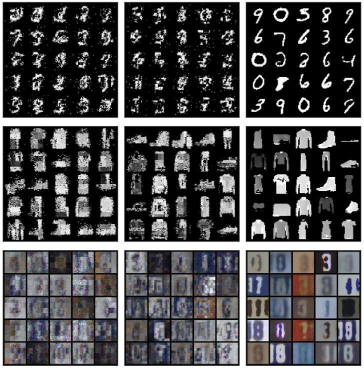
$$p(\boldsymbol{x}) \approx \sum_{k=1}^{K} w(\boldsymbol{z}_k)\, p(\boldsymbol{x} \mid \boldsymbol{z}_k)$$

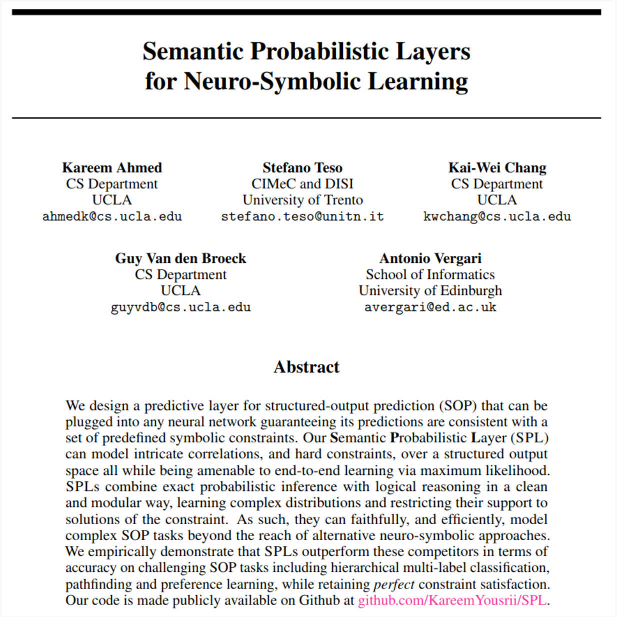
- This is again a PC!
- Thus, this can be seen as a new way to train PCs

Results



Dataset	BestPC	$\mathrm{cm}(\mathcal{S}_\mathsf{F})$	$\mathrm{cm}(\mathcal{S}_\mathsf{CLT})$	$\mathsf{LO}(\mathrm{cm}(\mathcal{S}_\mathsf{CLT}))$	Dataset	BestPC	$\mathrm{cm}(\mathcal{S}_\mathsf{F})$	$\mathrm{cm}(\mathcal{S}_\mathsf{CLT})$	$\mathsf{LO}(\mathrm{cm}(\mathcal{S}_\mathsf{CLT}))$
accid.	**-26.74**	-33.27	-28.69	-28.81	jester	-52.46	**-51.93**	-51.94	-51.94
ad	-16.07	-18.71	-14.76	**-14.42**	kdd	**-2.12**	-2.13	**-2.12**	**-2.12**
baudio	-39.77	**-39.02**	**-39.02**	-39.04	kosarek	-10.60	-10.71	-10.56	**-10.55**
bbc	-248.33	**-240.19**	-242.83	-242.79	msnbc	**-6.03**	-6.14	-6.05	-6.05
bnetflix	-56.27	-55.49	**-55.31**	-55.36	msweb	-9.73	-9.68	-9.62	**-9.60**
book	-33.83	-33.67	-33.75	**-33.55**	nltcs	**-5.99**	**-5.99**	**-5.99**	**-5.99**
c20ng	-151.47	-148.24	**-148.17**	-148.28	plants	-12.54	-12.45	**-12.26**	-12.27
cr52	-83.35	-81.52	**-81.17**	-81.31	pumbs	**-22.40**	-27.67	-23.71	-23.70
cwebkb	-151.84	-150.21	-147.77	**-147.75**	tmovie	-50.81	**-48.69**	-49.23	-49.29
dna	**-79.05**	-95.64	-84.91	-84.58	tretail	-10.84	10.85	-10.82	**-10.81**
Avg. rank	2.85	2.65	1.85	**1.75**					





Semantic Probabilistic Layers for Neuro-Symbolic Learning

Kareem Ahmed
CS Department
UCLA
ahmedk@cs.ucla.edu

Stefano Teso
CIMeC and DISI
University of Trento
stefano.teso@unitn.it

Kai-Wei Chang
CS Department
UCLA
kwchang@cs.ucla.edu

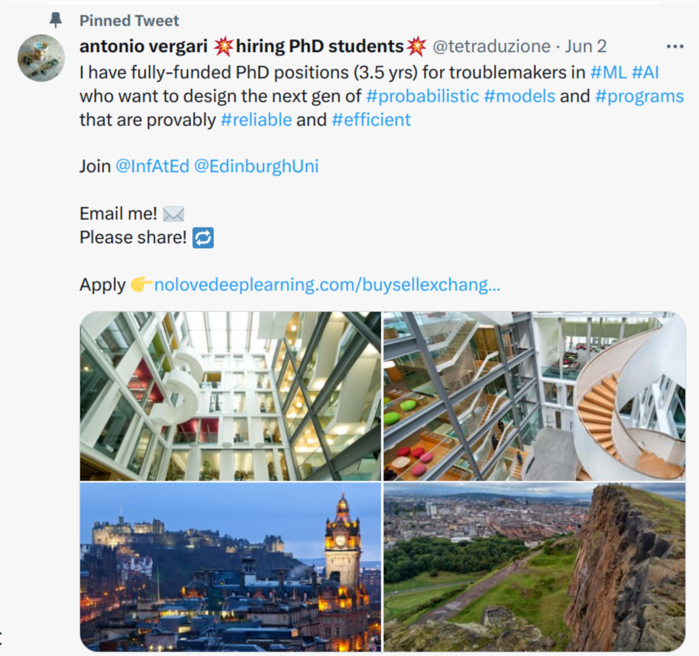
Guy Van den Broeck
CS Department
UCLA
guyvdb@cs.ucla.edu

Antonio Vergari
School of Informatics
University of Edinburgh
avergari@ed.ac.uk

Abstract

We design a predictive layer for structured-output prediction (SOP) that can be plugged into any neural network guaranteeing its predictions are consistent with a set of predefined symbolic constraints. Our **S**emantic **P**robabilistic **L**ayer (SPL) can model intricate correlations, and hard constraints, over a structured output space all while being amenable to end-to-end learning via maximum likelihood. SPLs combine exact probabilistic inference with logical reasoning in a clean and modular way, learning complex distributions and restricting their support to solutions of the constraint. As such, they can faithfully, and efficiently, model complex SOP tasks beyond the reach of alternative neuro-symbolic approaches. We empirically demonstrate that SPLs outperform these competitors in terms of accuracy on challenging SOP tasks including hierarchical multi-label classification, pathfinding and preference learning, while retaining *perfect* constraint satisfaction. Our code is made publicly available on Github at github.com/KareemYousrii/SPL.

`https://github.com/KareemYousrii/SPL` NeurIPS'22



Pinned Tweet

antonio vergari 💥hiring PhD students💥 @tetraduzione · Jun 2

I have fully-funded PhD positions (3.5 yrs) for troublemakers in #ML #AI who want to design the next gen of #probabilistic #models and #programs that are provably #reliable and #efficient

Join @InfAtEd @EdinburghUni

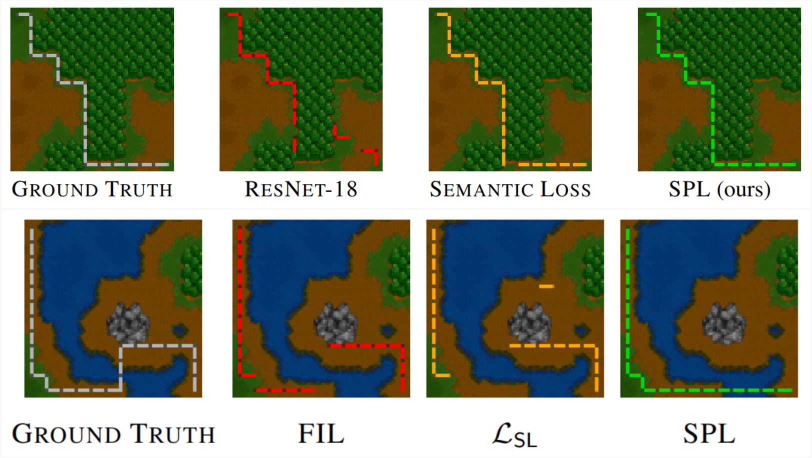
Email me! ✉️
Please share! 🔁

Apply 👉nolovedeeplearning.com/buysellexchang...

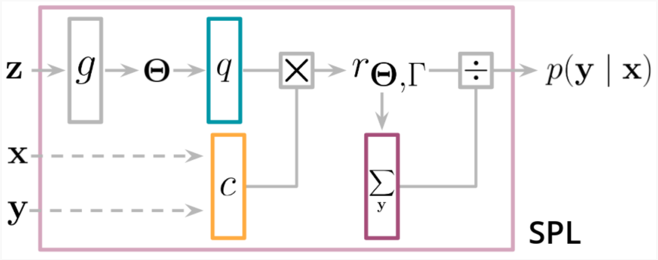
http

IPS'22

Structured Output

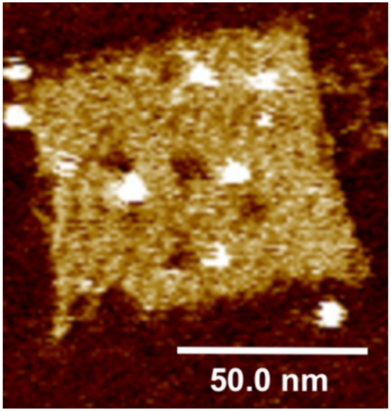


Semantic Probabilistic Layer



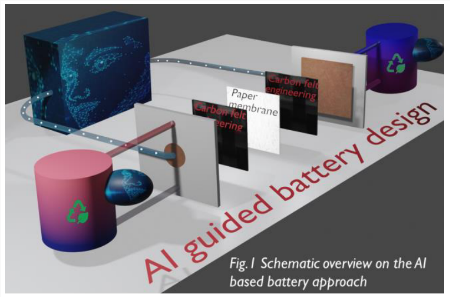
- logical world knowledge encoded in logic circuit c
- unconstrained probability distribution q: PC parametrized by neural net
- with **structured decomposability** (stricter version of decomposability) one can multiply c and q, yielding a new PC of polynomial (quadratic) size
- this yields a predictive $p(\boldsymbol{y} \mid \boldsymbol{x})$ obeying logical constraints!

Advertisement 1



- improving maturity level of **DNA storage**
- 1 PhD position for probabilistic-symbolic ML (with me)
- 1 PhD position for computer vision (with Thomas Pock)

Advertisement 2



European Innovation Council

TU Graz

- developing sustainable **Large-scale Organic Batteries**
- 2 PhD positions for probabilistic ML, Bayesian optimization, AI4Science (with me and Roman Kern)

robert.peharz@tugraz.at

Reading:

- Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Models
- Foundations of Sum-Product Networks for Probabilistic Reasoning

Tutorials:

- NeurIPS'22 Tutorial (video)
- ECML/PKDD'20 Tutorial (video)
- AAAI'20 Tutorial (slides)

https://github.com/SPFlow/SPFlow

https://github.com/Juice-jl/ProbabilisticCircuits.jl