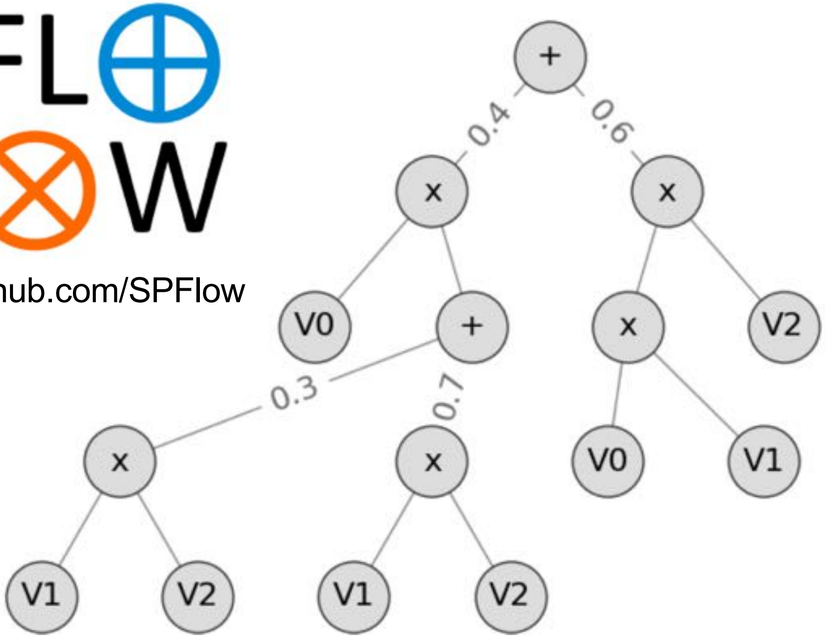
Sum-Product Networks*

The Third Wave of Differentiable Programming

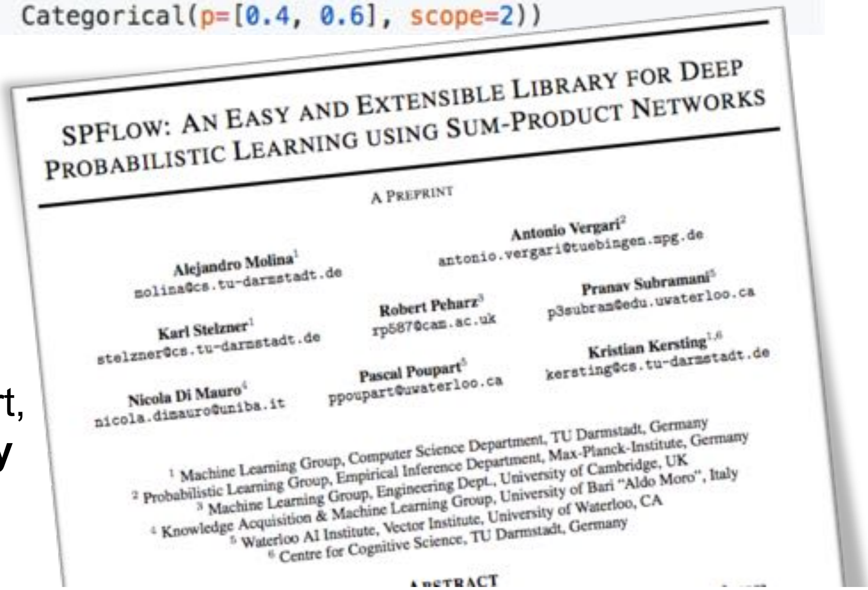
Kristian Kersting

*Thanks for Pedro Domingos for making his slides publically available

github.com/SPFlow




```python
from spn.structure.leaves.parametric.Parametric import Categorical

spn = 0.4 * (Categorical(p=[0.2, 0.8], scope=0) *
             (0.3 * (Categorical(p=[0.3, 0.7], scope=1) *
                     Categorical(p=[0.4, 0.6], scope=2))
            + 0.7 * (Categorical(p=[0.5, 0.5], scope=1) *
                     Categorical(p=[0.6, 0.4], scope=2))))
    + 0.6 * (Categorical(p=[0.2, 0.8], scope=0) *
             Categorical(p=[0.3, 0.7], scope=1) *
             Categorical(p=[0.4, 0.6], scope=2))
```



SPFLOW: AN EASY AND EXTENSIBLE LIBRARY FOR DEEP PROBABILISTIC LEARNING USING SUM-PRODUCT NETWORKS

A PREPRINT

Alejandro Molina[1] molina@cs.tu-darmstadt.de
Antonio Vergari[2] antonio.vergari@tuebingen.mpg.de
Karl Stelzner[1] stelzner@cs.tu-darmstadt.de
Robert Peharz[3] rp587@cam.ac.uk
Pranav Subramani[5] p3subram@edu.uwaterloo.ca
Nicola Di Mauro[4] nicola.dimauro@uniba.it
Pascal Poupart[5] ppoupart@uwaterloo.ca
Kristian Kersting[1,6] kersting@cs.tu-darmstadt.de

[1] Machine Learning Group, Computer Science Department, TU Darmstadt, Germany
[2] Probabilistic Learning Group, Empirical Inference Department, Max-Planck-Institute, Germany
[3] Machine Learning Group, Engineering Dept., University of Cambridge, UK
[4] Knowledge Acquisition & Machine Learning Group, University of Bari "Aldo Moro", Italy
[5] Waterloo AI Institute, Vector Institute, University of Waterloo, CA
[6] Centre for Cognitive Science, TU Darmstadt, Germany

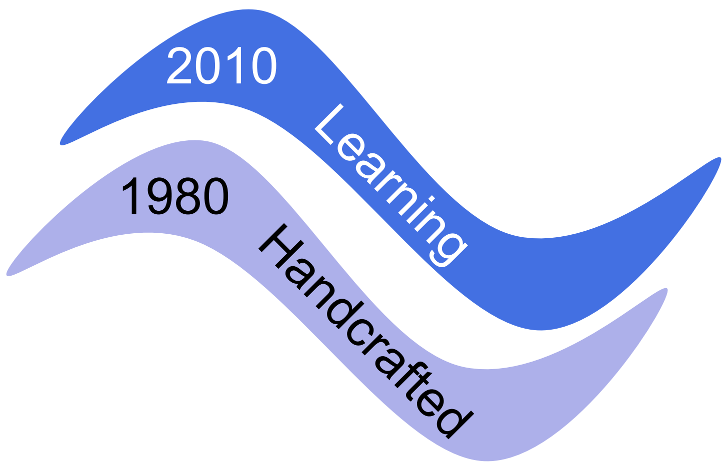
ABSTRACT

Alejandro Molina, Antonio Vergari, Karl Stelzner, Robert Peharz, Pranav Subramani, Nicola Di Mauro, Pascal Poupart, Kristian Kersting: **SPFlow: An Easy and Extensible Library for Deep Probabilistic Learning using Sum-Product Networks**. CoRR abs/1901.03704 (2019)

AI has impact

Data are now ubiquitous; there is great value from understanding this data, building models and making predictions

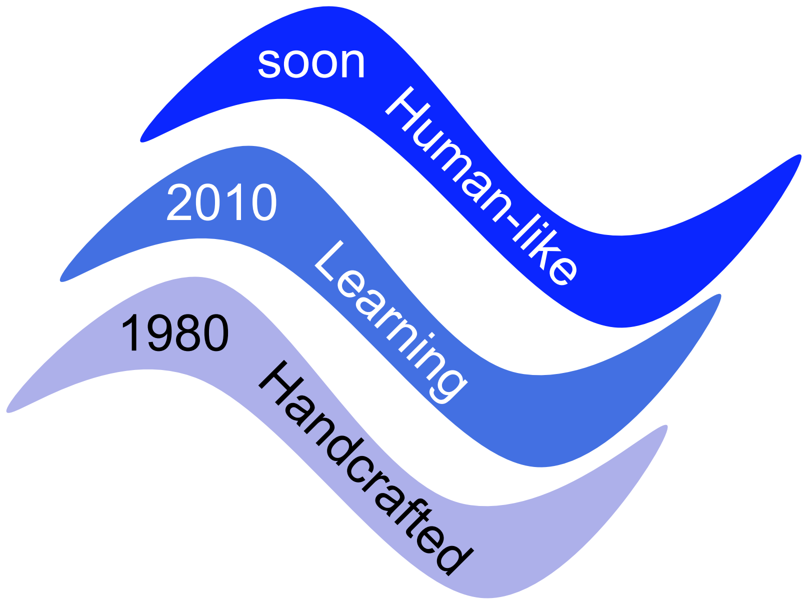
However, data is not everything



The third wave of AI

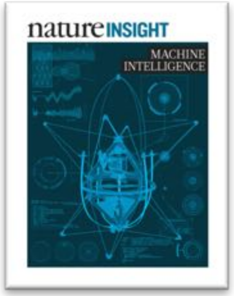
Data are now ubiquitous; there is great value from understanding this data, building models and making predictions

However, data is not everything



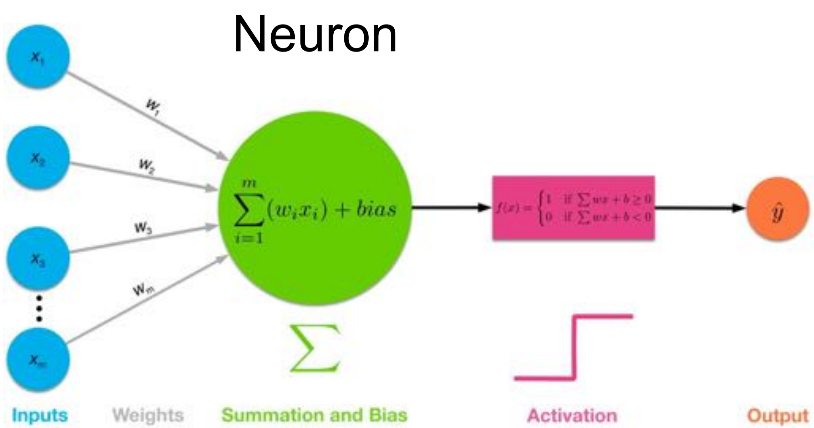
AI systems that can acquire human-like communication and reasoning capabilities, with the ability to recognise new situations and adapt to them.

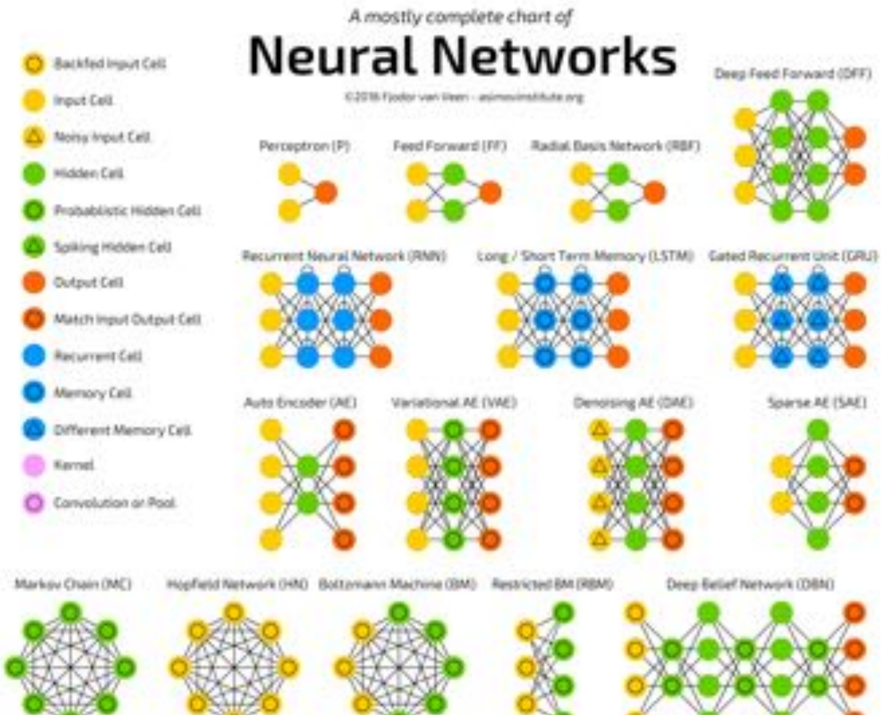
Deep Neural Networks



Potentially much more powerful than shallow architectures, represent computations

[LeCun, Bengio, Hinton Nature 521, 436–444, 2015]





Differentiable Programming

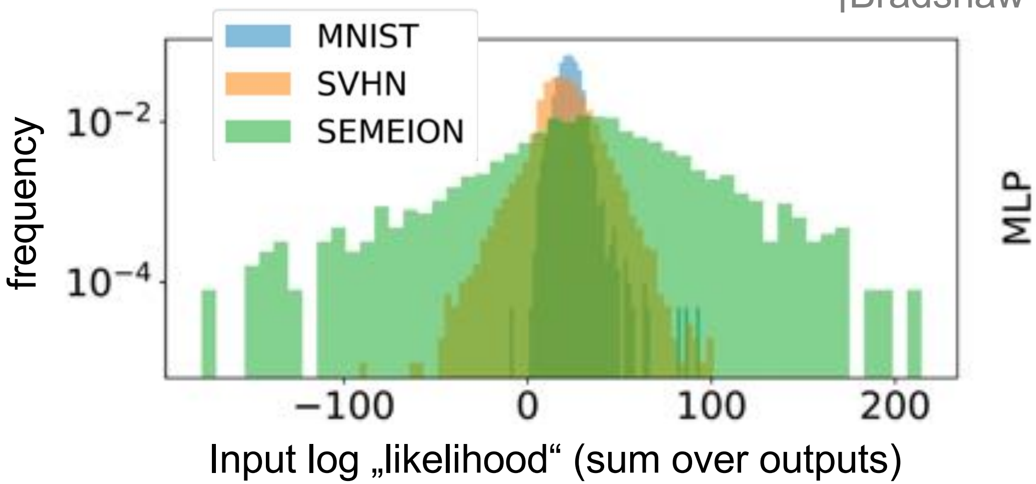
DNNs often have no probabilistic semantics. They are not calibrated joint distributions.

$$P(Y|X) \neq P(Y,X)$$

MNIST | **SVHN** | **SEMEION**

Train & Evaluate | **Transfer Testing**

[Bradshaw et al. arXiv:1707.02476 2017]

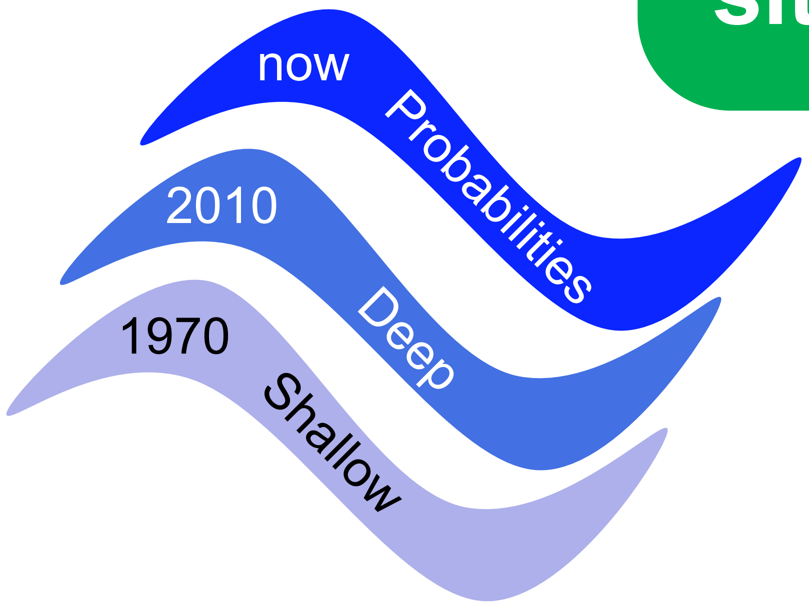


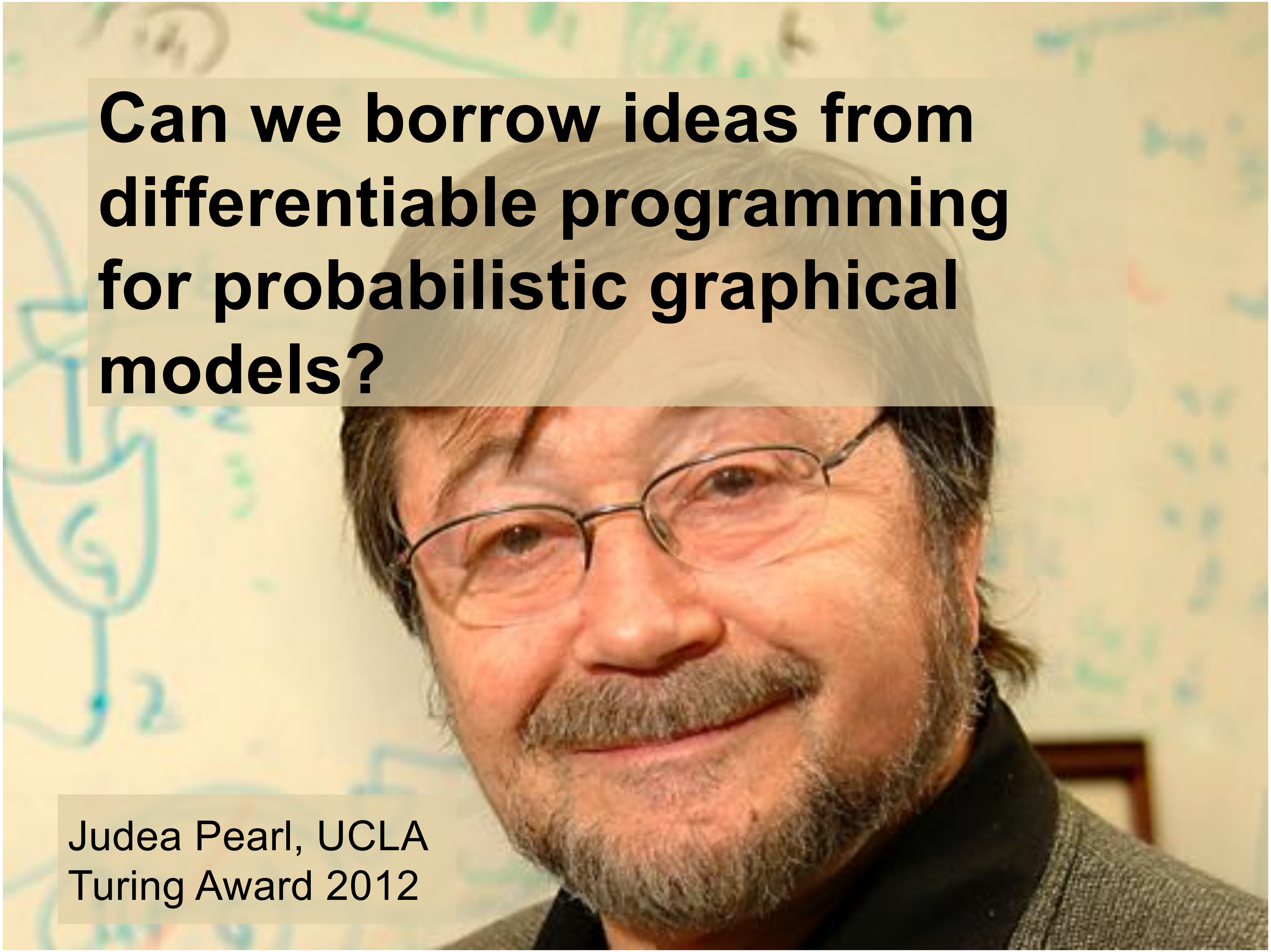
Many DNNs cannot distinguish the datasets

[Peharz, Vergari, Molina, Stelzner, Trapp, Kersting, Ghahramani UDL@UAI 2018]

The third wave of differentiable programming

Getting deep systems that know when they do not know and, hence, recognise new situations and adapt to them





Can we borrow ideas from differentiable programming for probabilistic graphical models?

Judea Pearl, UCLA
Turing Award 2012

Alternative Representation: Graphical Models as (Deep) Networks

X_1	X_2	$P(X)$
1	1	0.4
1	0	0.2
0	1	0.1
0	0	0.3

$$
\begin{aligned}
P(X) = {} & 0.4 \cdot \mathrm{I}[X_1{=}1] \cdot \mathrm{I}[X_2{=}1] \\
& + 0.2 \cdot \mathrm{I}[X_1{=}1] \cdot \mathrm{I}[X_2{=}0] \\
& + 0.1 \cdot \mathrm{I}[X_1{=}0] \cdot \mathrm{I}[X_2{=}1] \\
& + 0.3 \cdot \mathrm{I}[X_1{=}0] \cdot \mathrm{I}[X_2{=}0]
\end{aligned}
$$

Alternative Representation: Graphical Models as (Deep) Networks

X_1	X_2	$P(X)$
1	**1**	**0.4**
1	0	0.2
0	1	0.1
0	0	0.3

$$P(X) = \mathbf{0.4 \cdot I[X_1{=}1] \cdot I[X_2{=}1]} + 0.2 \cdot I[X_1{=}1] \cdot I[X_2{=}0] + 0.1 \cdot I[X_1{=}0] \cdot I[X_2{=}1] + 0.3 \cdot I[X_1{=}0] \cdot I[X_2{=}0]$$

Shorthand using Indicators

X_1	X_2	$P(X)$
1	1	0.4
1	0	0.2
0	1	0.1
0	0	0.3

$$\begin{aligned} P(X) = {} & 0.4 \cdot X_1 \cdot X_2 \\ & + 0.2 \cdot X_1 \cdot \overline{X}_2 \\ & + 0.1 \cdot \overline{X}_1 \cdot X_2 \\ & + 0.3 \cdot \overline{X}_1 \cdot \overline{X}_2 \end{aligned}$$

Summing Out Variables

Let us say, we want to compute $P(X_1 = 1)$

X_1	X_2	$P(X)$
1	**1**	**0.4**
1	**0**	**0.2**
0	1	0.1
0	0	0.3

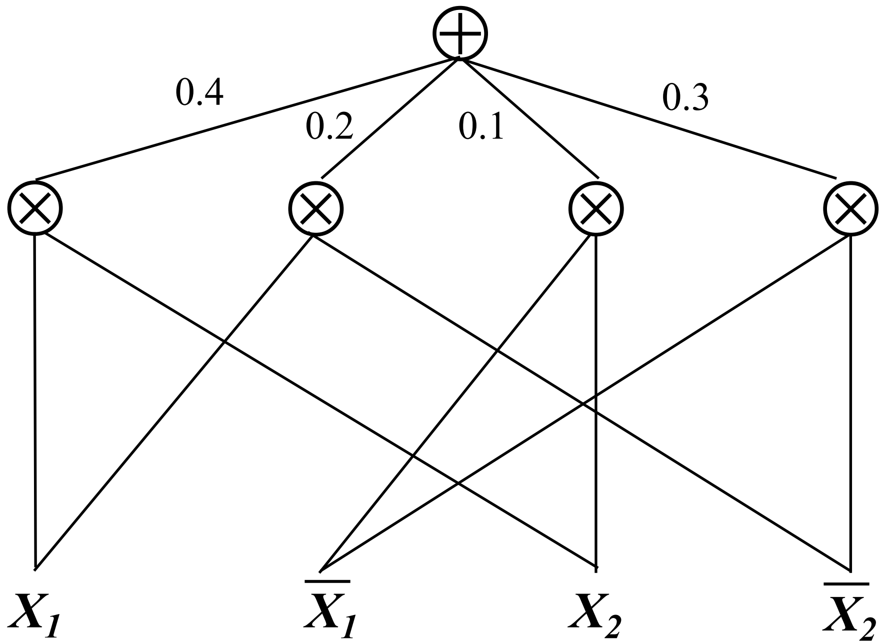
$$
\begin{aligned}
P(e) = &\ \mathbf{0.4 \cdot X_1 \cdot X_2} \\
&+ \mathbf{0.2 \cdot X_1 \cdot \overline{X}_2} \\
&+ 0.1 \cdot \overline{X}_1 \cdot X_2 \\
&+ 0.3 \cdot \overline{X}_1 \cdot \overline{X}_2
\end{aligned}
$$

Set $X_1 = 1, \overline{X}_1 = 0, \boxed{X_2 = 1, \overline{X}_2 = 1}$

Easy: Set both indicators of X2 to 1

This can be represented as a computational graph

X_1	X_2	$P(X)$
1	1	0.4
1	0	0.2
0	1	0.1
0	0	0.3

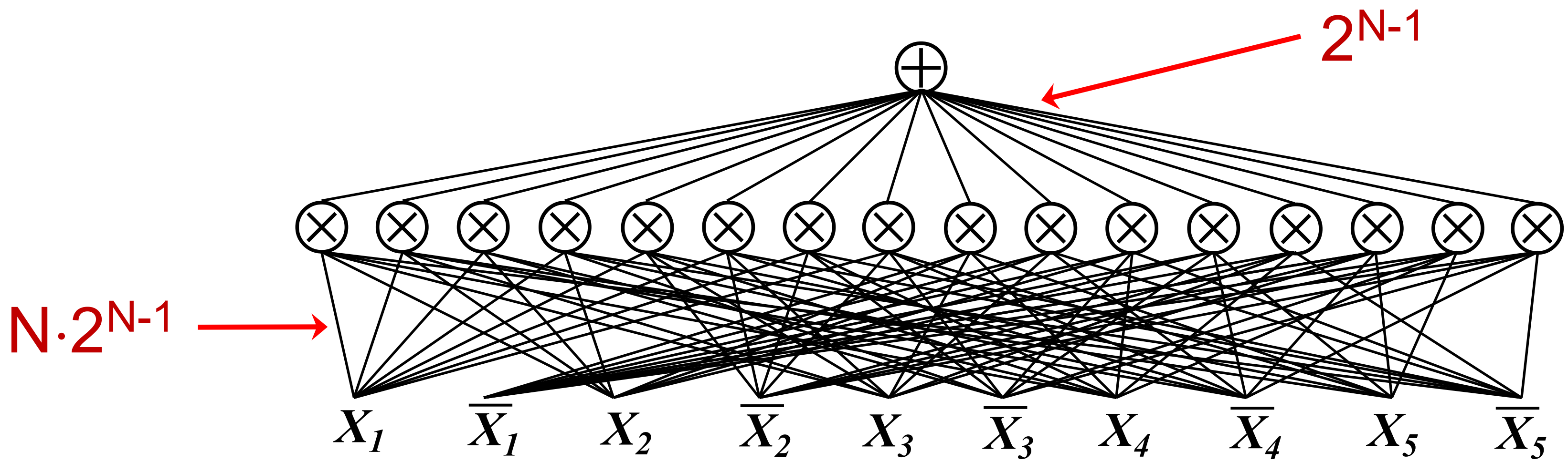


network polynomial

However, the network polynomial of a distribution might be exponentially large

Example: Parity

Uniform distribution over states with even number of 1's



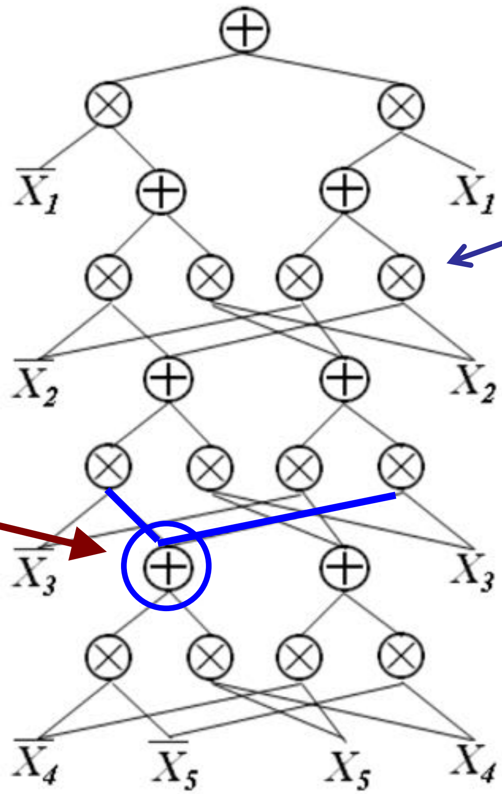
Make the computational graphs deep

Example: Parity

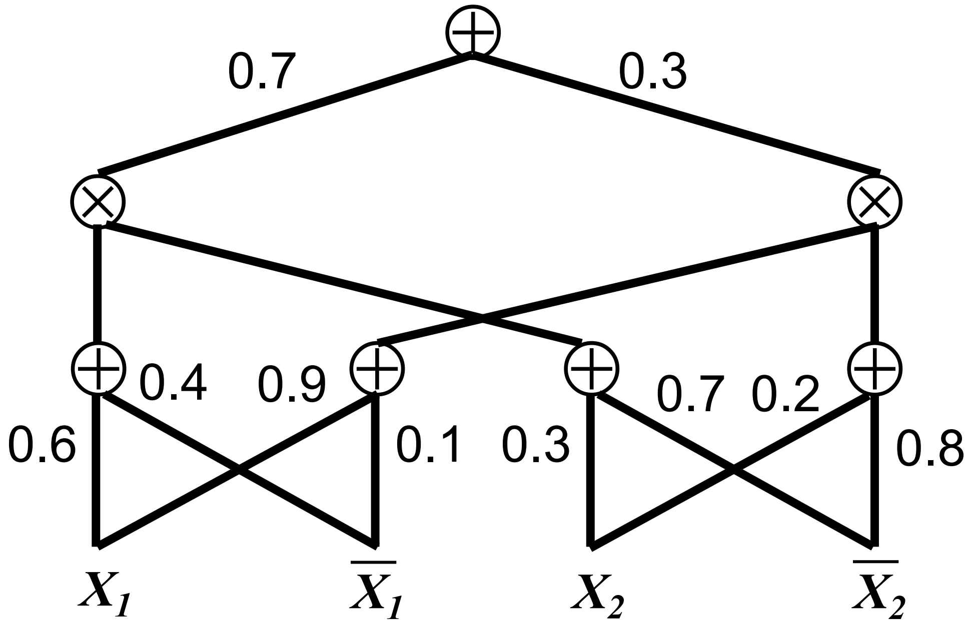
Uniform distribution over states with even number of 1's

Induce many hidden layers

Reuse partial computation



Sum-Product Networks*
a deep probabilistic learning framework [Poon, Domingos UAI 2011]



A SPN S is a rooted DAG where **nodes** are sum, product, input indicator and **weights** are on edges from the sums of children

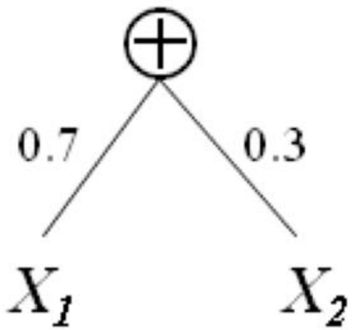
*SPNs are an instance of Arithmetic Circuits (ACs). ACs have been introduced into the AI literature more than15 years ago as a tractable representation of probability distributions
[Darwiche CACM 48(4):608-647 2001] c

Valid SPN: General Conditions

SPN is valid if $S(e) = \Sigma_{X\sim e} S(X)$ for all e. If so, we can compute (conditional) marginals efficiently since the partition function Z can be computed by setting all indicators to 1

Theorem: *SPN is valid if it is complete & consistent*

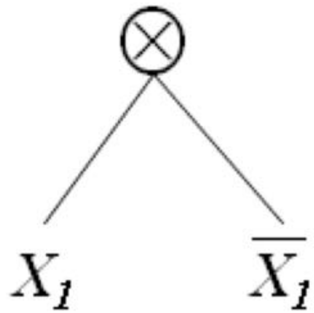
Complete: Under sum, children cover the same set of variables



Incomplete

$S(e) \leq \Sigma_{X\sim e} S(X)$

Consistent: Under product, no variable in one child and negation in another



Inconsistent

$S(e) \geq \Sigma_{X\sim e} S(X)$

Inference: Linear in Size of Network

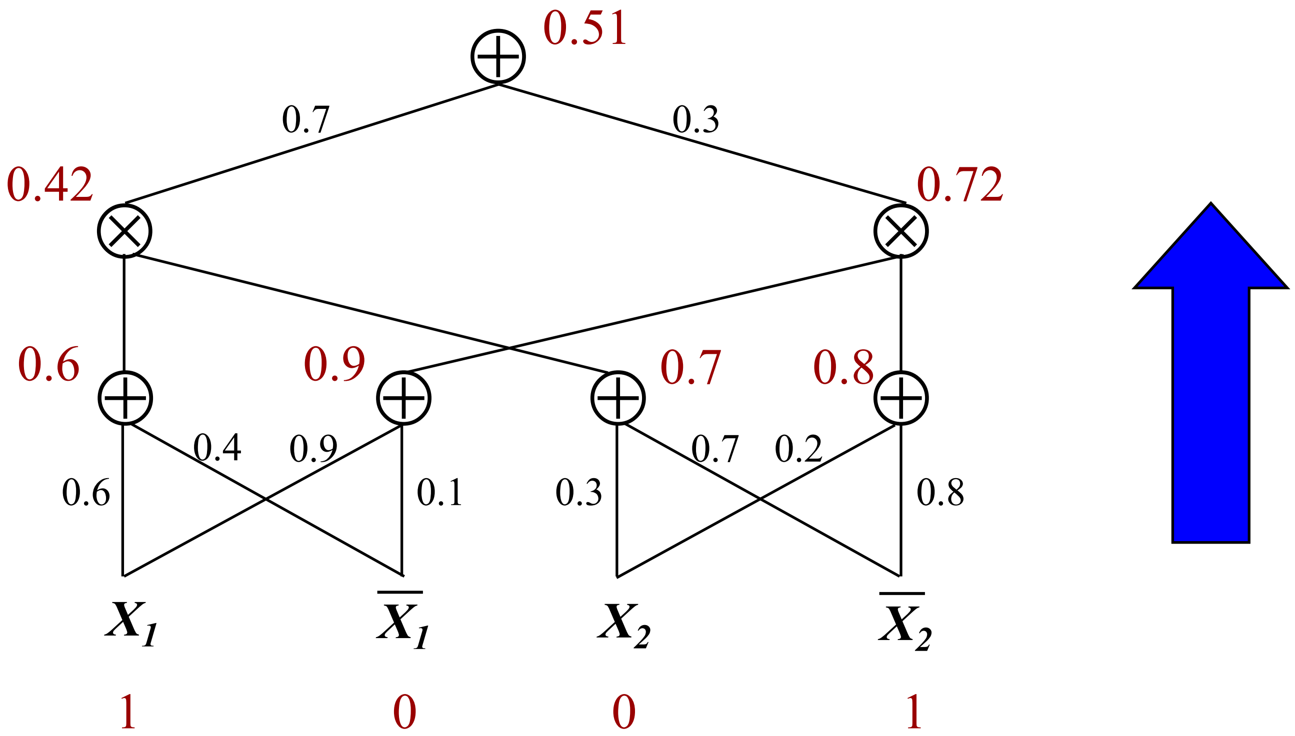
As long as weights sum to 1
at each sum node

$$P(X) = S(X)$$

X: $X_1 = 1, X_2 = 0$

X_1	1
$\overline{X}_1$	0
X_2	0
$\overline{X}_2$	1

How to set the indicator variables



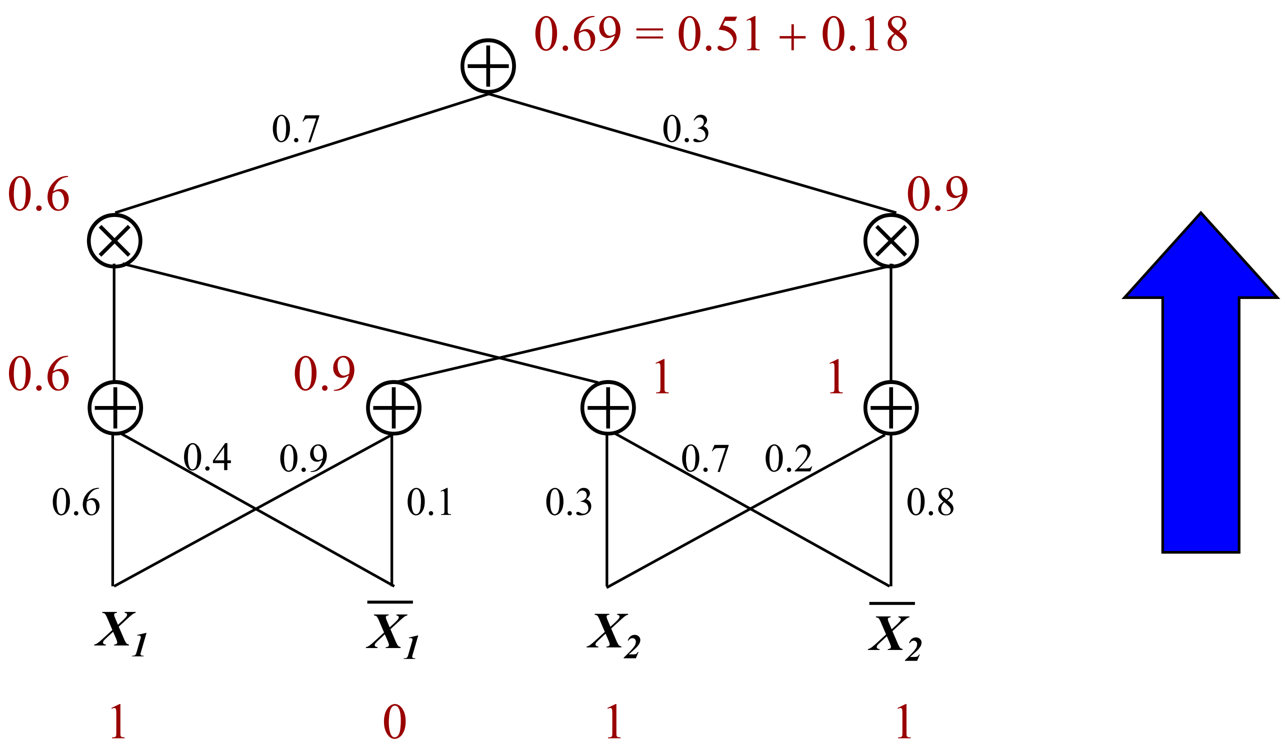
Inference: Linear in Size of Network

Marginal: $P(e) = S(e)$

e: $X_1 = 1$

X_1	1
$\overline{X}_1$	0
X_2	1
$\overline{X}_2$	1

How to set the indicator variables



Inference: Linear in Size of Network

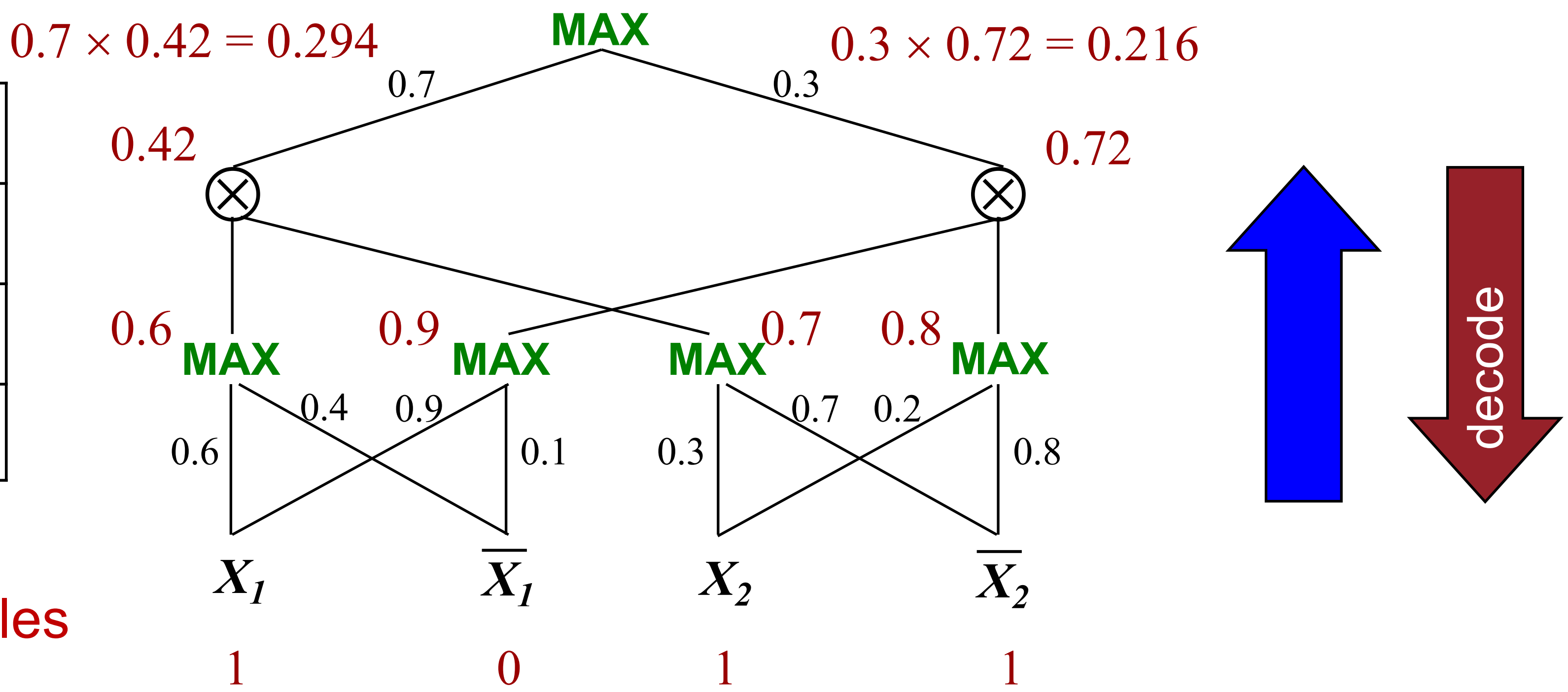
TECHNISCHE UNIVERSITÄT DARMSTADT

MAP: Replace sums with maxs

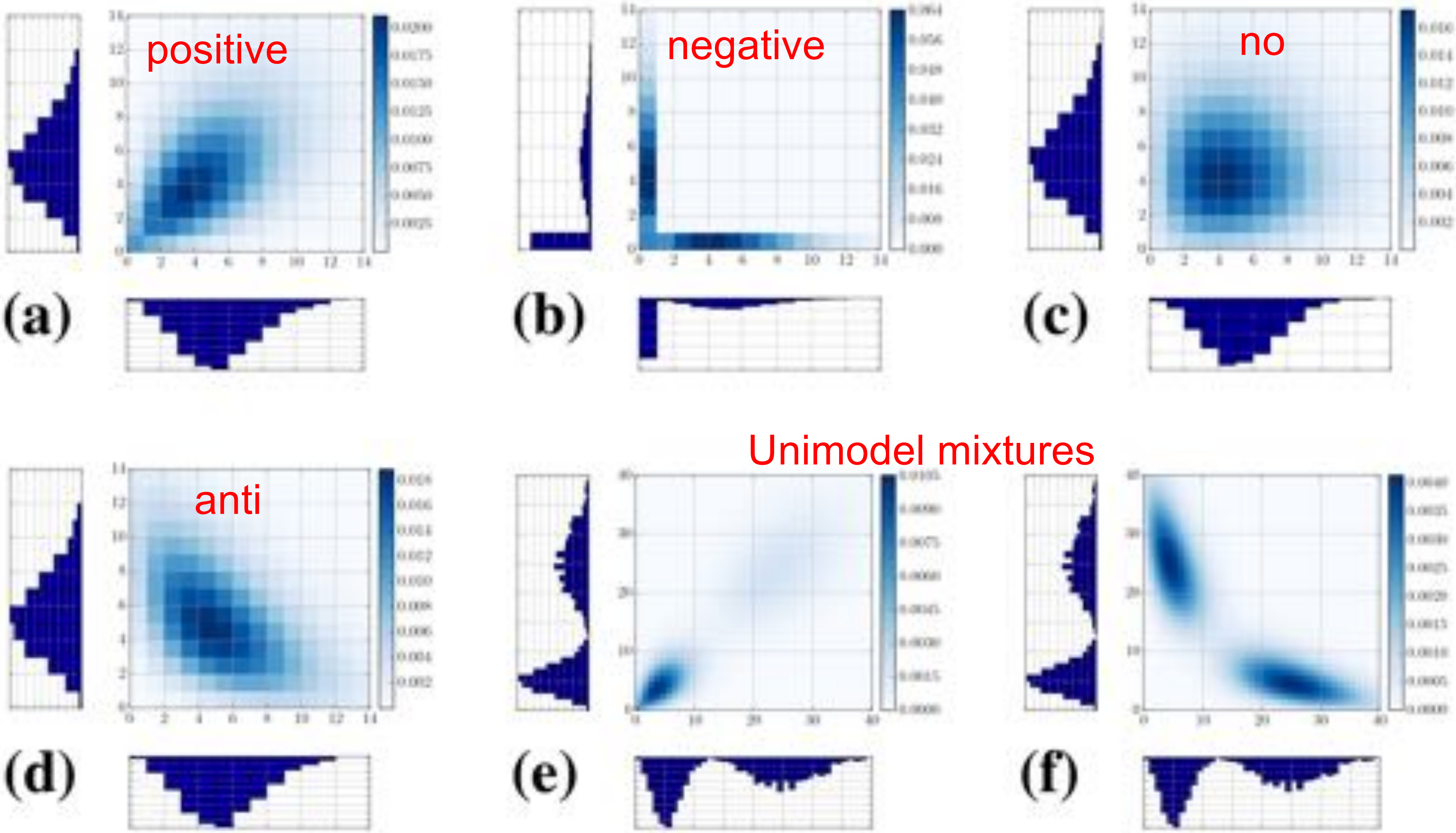
e: $X_1 = 1$

X_1	1
$\overline{X}_1$	0
X_2	1
$\overline{X}_2$	1

How to set the indicator variables



Building challenging multivariate distributions from well-known univariate distributions with flexible correlations, here multivariate Poisson distribution



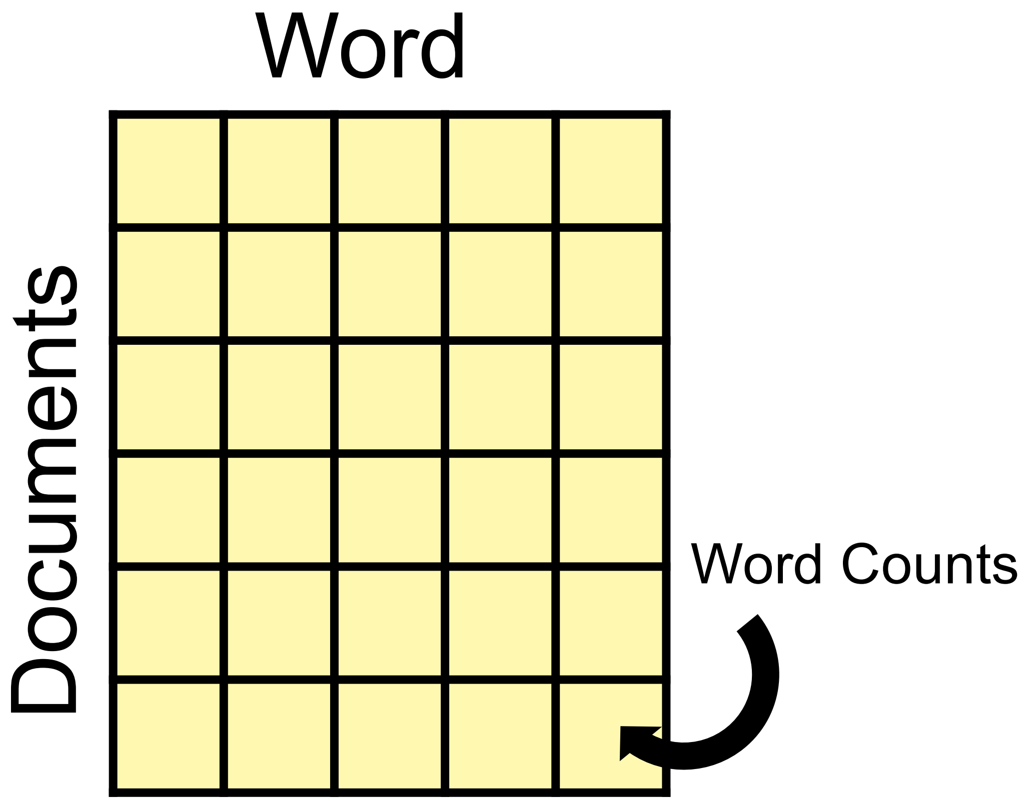
And also learning is simple. E.g. we can learn (the structure) via parameter estimation assuming a fixed network (like in Deep Neural Learning)

- Start with a dense SPN
- Find the structure by (online) learning weights
  Zero weights signify absence of connections
- (Hard) EM beneficial to avoid gradient vanishing
  Each sum node is a mixture over children

In principle you can turn a given SPN into, say, a TensorFlow computation graph and apply any known algorithm from there

Or we learn directly (Tree-)SPNs

Testing independence using a (non-parametric) independency test



[Poon, Domingos UAI'11; Molina, Natarajan, Kersting AAAI´17]

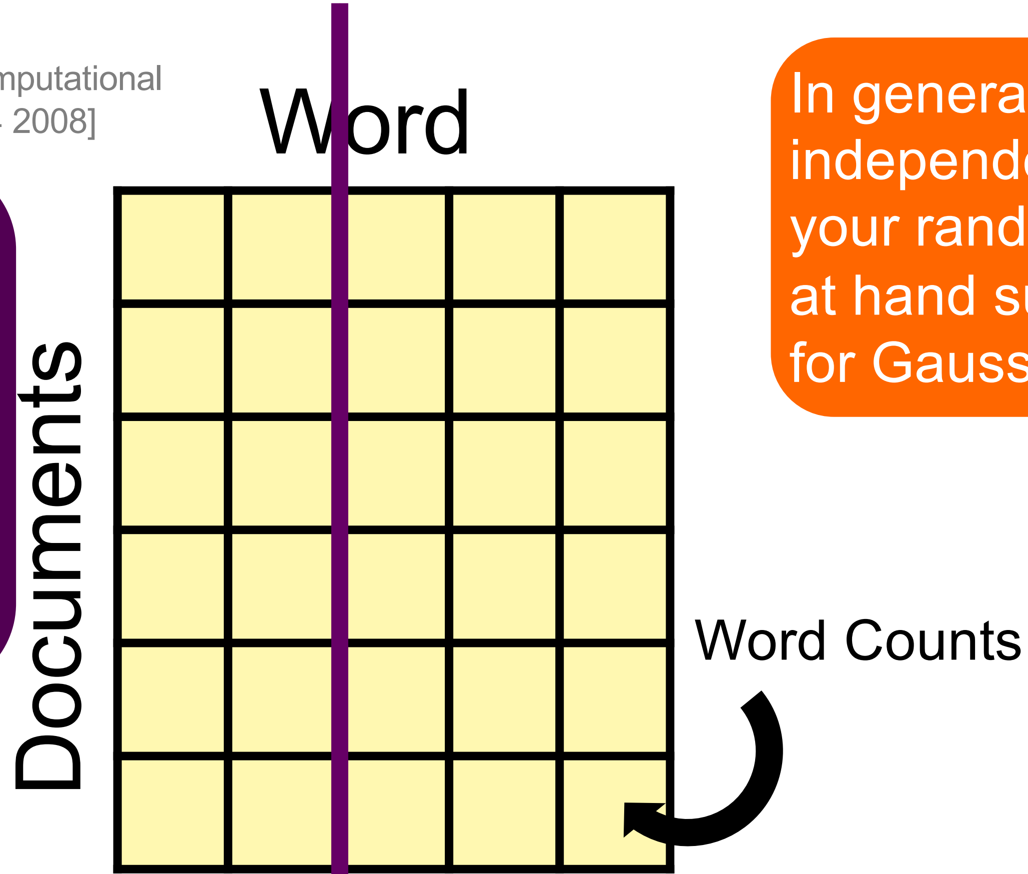
Or we learn directly (Tree-)SPNs

TECHNISCHE UNIVERSITÄT DARMSTADT

Testing independence using a (non-parametric) independency test

[Zeileis, Hothorn, Hornik Journal of Computational And Graphical Statistics 17(2):492–514 2008]

E.g. for Poisson RVs: Learn Poisson model trees for P(x|V-x) and P(y|V-y). Check whether X resp. Y is significant in P(y|V-x) resp. P(x|V-y)



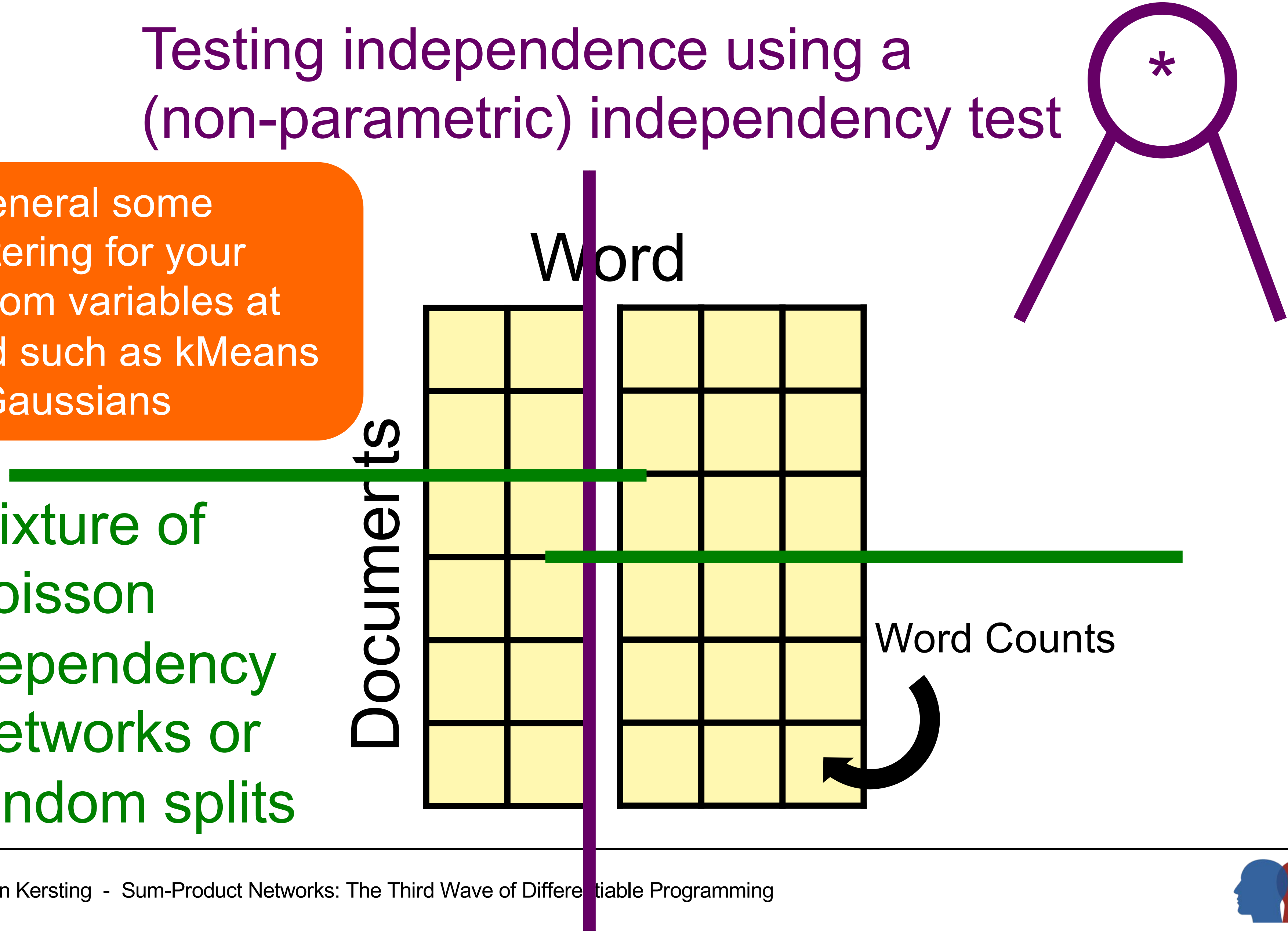
In general use the independency test for your random variables at hand such as g-test for Gaussians

Or we learn directly (Tree-)SPNs

Testing independence using a (non-parametric) independency test

In general some clustering for your random variables at hand such as kMeans for Gaussians

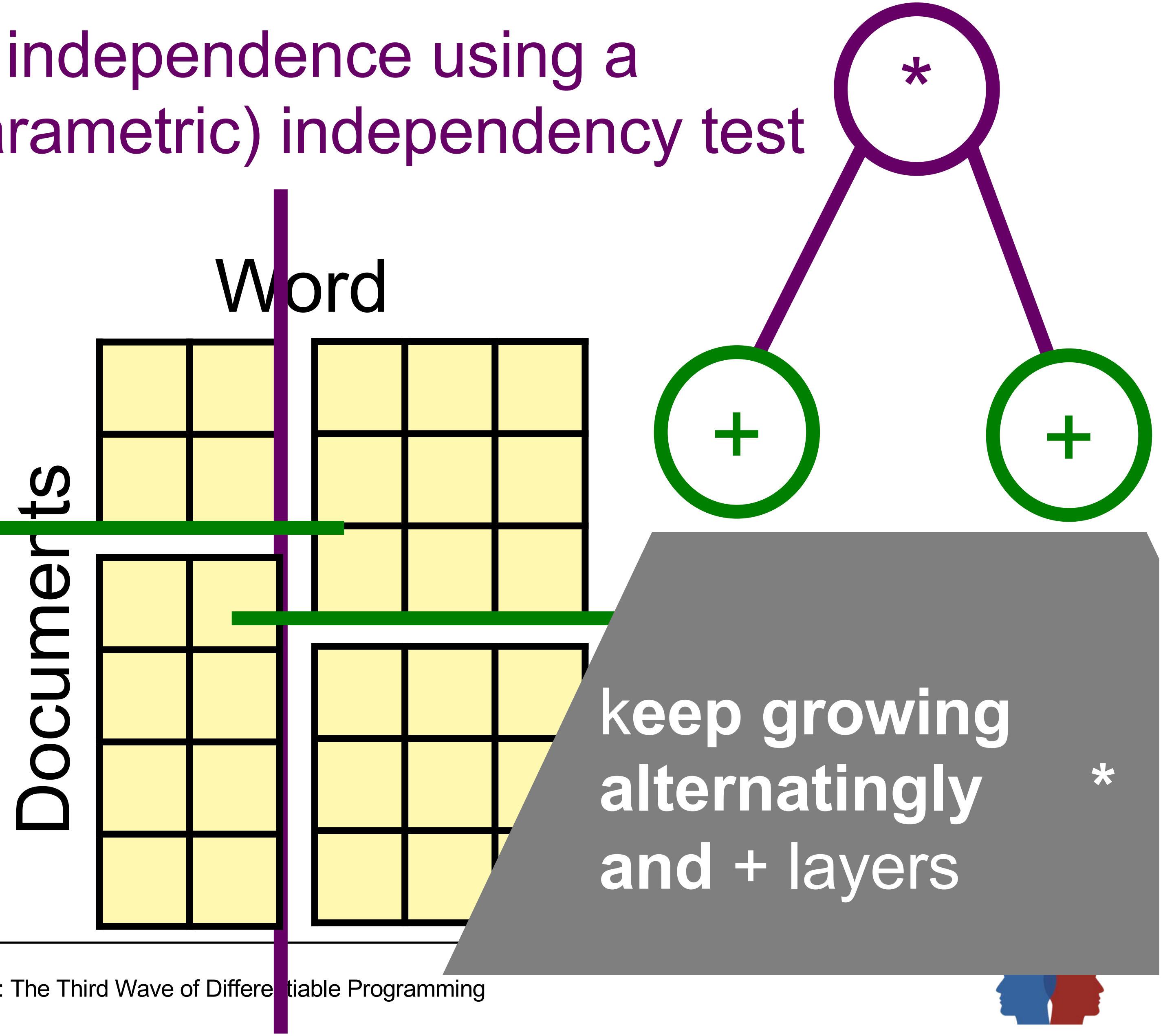
Mixture of Poisson Dependency Networks or random splits



Or we learn directly (Tree-)SPNs

TECHNISCHE UNIVERSITÄT DARMSTADT

Testing independence using a (non-parametric) independency test



Clustering or random splits

keep growing * and + layers alternatingly

[Poon, Domingos UAI'11; Molina, Natarajan, Kersting AAAI'17; Vergari, Peharz, Di Mauro, Molina, Kersting, Esposito AAAI '18; Molina, Vergari, Di Mauro, Esposito, Natarajan, Kersting AAAI '18]

SPFlow: An Easy and Extensible Library for Sum-Product Networks

[Molina, Vergari, Stelzner, Peharz, Subramani, Poupart, Di Mauro, Kersting 2019]

TECHNISCHE UNIVERSITÄT DARMSTADT
UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO
UNIVERSITY OF WATERLOO
CAML
MADESI
Max Planck Institute for Intelligent Systems
UNIVERSITY OF CAMBRIDGE
VECTOR INSTITUTE
DFG
Federal Ministry of Education and Research

195 commits | 2 branches | 0 releases | 6 contrib...

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

https://github.com/SPFlow/SPFlow

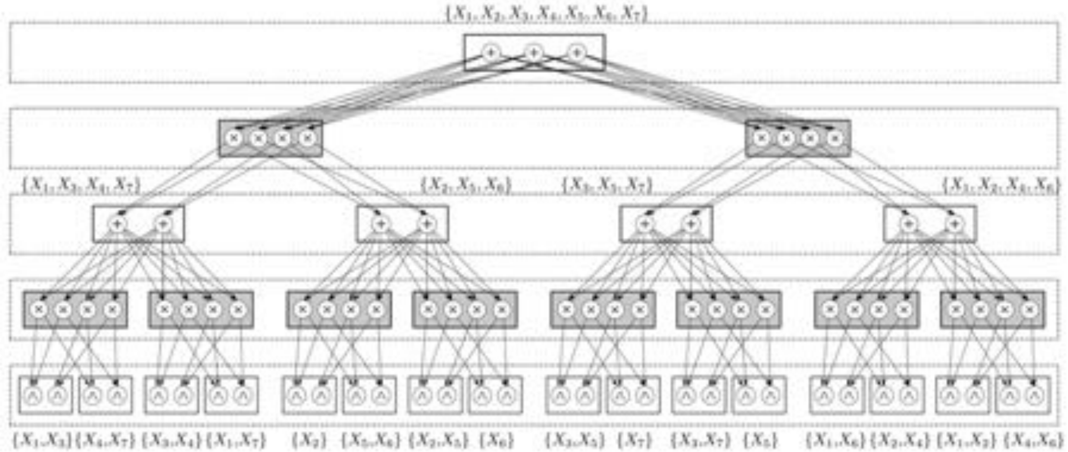
```
from spn.structure.leaves.parametric.Parametric import Categorical

from spn.structure.Base import Sum, Product

from spn.structure.base import assign_ids, rebuild_scopes_bottom_up


p0 = Product(children=[Categorical(p=[0.3, 0.7], scope=1), Categorical(p=[0.4, 0.6], scope=2)])
p1 = Product(children=[Categorical(p=[0.5, 0.5], scope=1), Categorical(p=[0.6, 0.4], scope=2)])
s1 = Sum(weights=[0.3, 0.7], children=[p0, p1])
p2 = Product(children=[Categorical(p=[0.2, 0.8], scope=0), s1])
p3 = Product(children=[Categorical(p=[0.2, 0.8], scope=0), Categorical(p=[0.3, 0.7], scope=1)])
p4 = Product(children=[p3, Categorical(p=[0.4, 0.6], scope=2)])
spn = Sum(weights=[0.4, 0.6], children=[p2, p4])

assign_ids(spn)
rebuild_scopes_bottom_up(spn)

return spn
```

Domain Specific Language, Inference, EM, and Model Selection as well as Compilation of SPNs into TF and PyTorch and also into flat, library-free code even suitable for running on devices: C/C++,GPU, FPGA

SPFlow, an open-source Python library providing a simple interface to inference, learning and manipulation routines for deep and tractable probabilistic models called Sum-Product Networks (SPNs). The library allows one to quickly create SPNs both from data and through a domain specific language (DSL). It efficiently implements several probabilistic inference

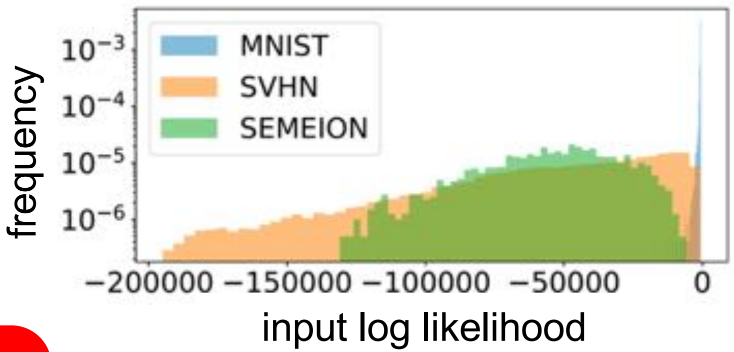
Random sum-product networks

[Peharz, Vergari, Molina, Stelzner, Trapp, Kersting, Ghahramani UDL@UAI 2018]



Build a random SPN structure. This can be done in an informed way or completely at random

		RAT-SPN	MLP	vMLP
Accuracy	MNIST	98.19 (8.5M)	98.32 (2.64M)	98.09 (5.28M)
	F-MNIST	89.52 (0.65M)	90.81 (9.28M)	89.81 (1.07M)
	20-NG	47.8 (0.37M)	49.05 (0.31M)	48.81 (0.16M)
Cross-Entropy	MNIST	0.0852 (17M)	0.0874 (0.82M)	0.0974 (0.22M)
	F-MNIST	0.3525 (0.65M)	0.2965 (0.82M)	0.325 (0.29M)
	20-NG	1.6954 (1.63M)	1.6180 (0.22M)	1.6263 (0.22M)



outliers

prototypes

outliers

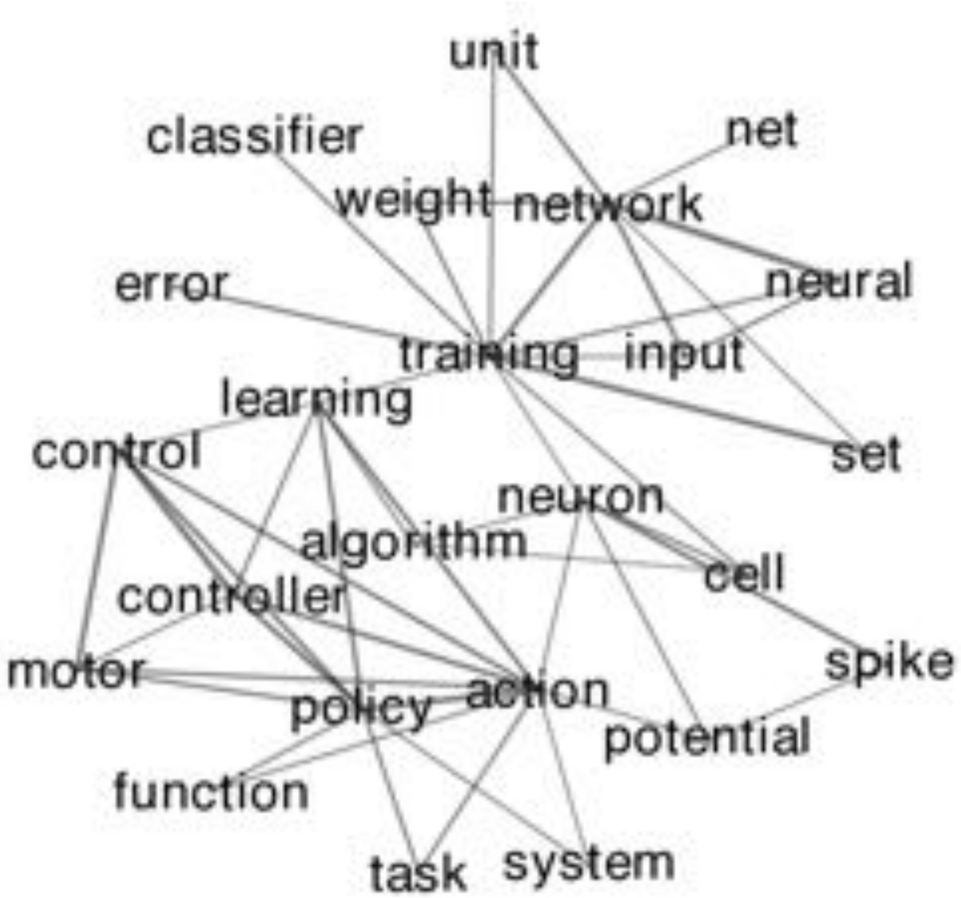
prototypes

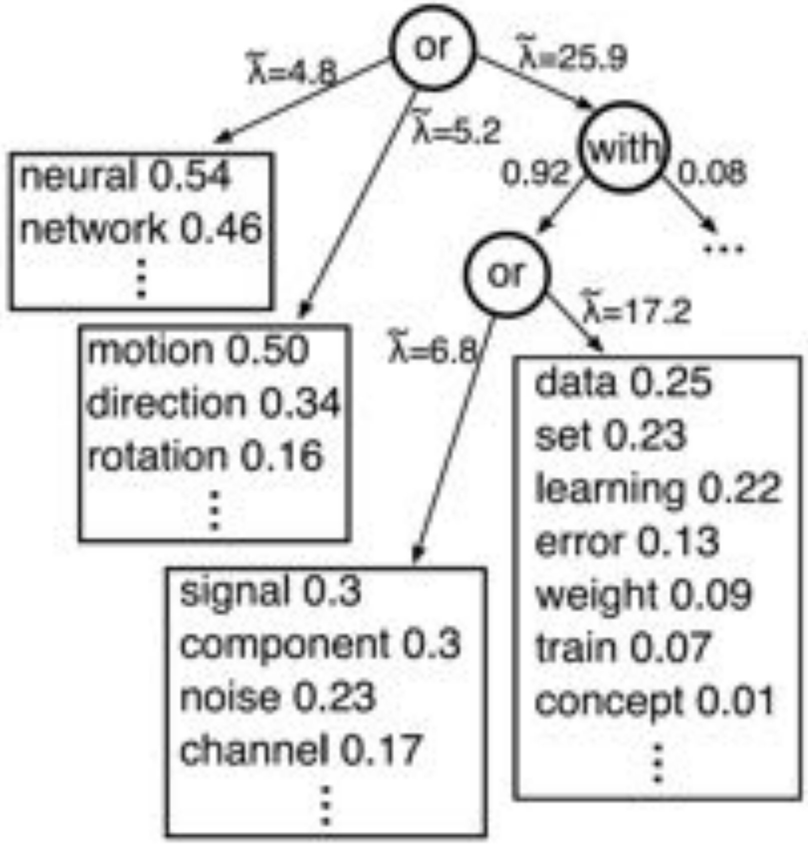
SPNs can have similar predictive performances as (simple) DNNs

SPNs can distinguish the datasets

SPNs know when they do not know by design

SPNs closely related to well known, advanced ML models, e.g. Poisson SPNs = Hierarchical Topic Models

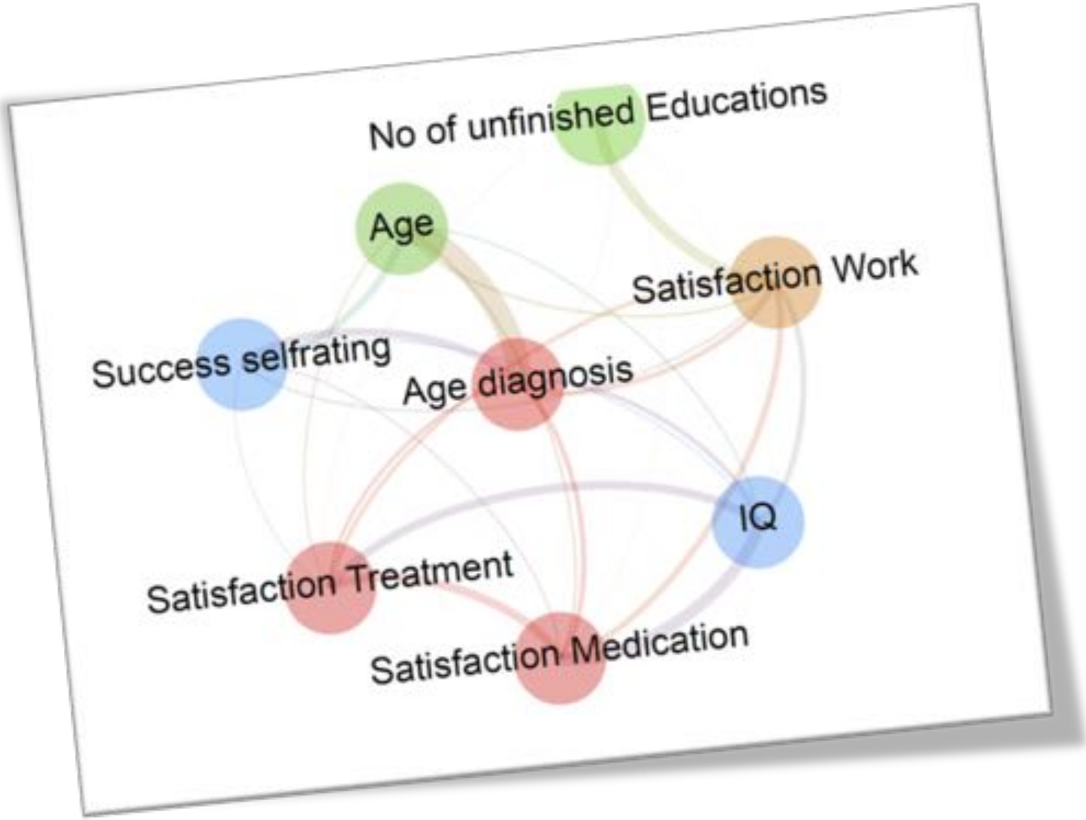




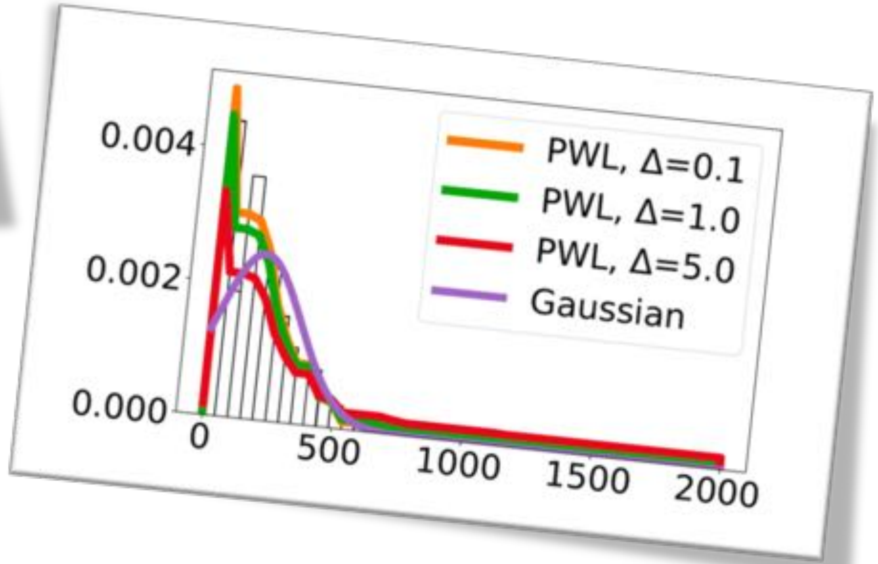
Mutual Information
(NIPS corpus)

Poisson Multinomial SPN
= hierachical topic model

SPNs feature distribution-agnostic deep probabilistic learning

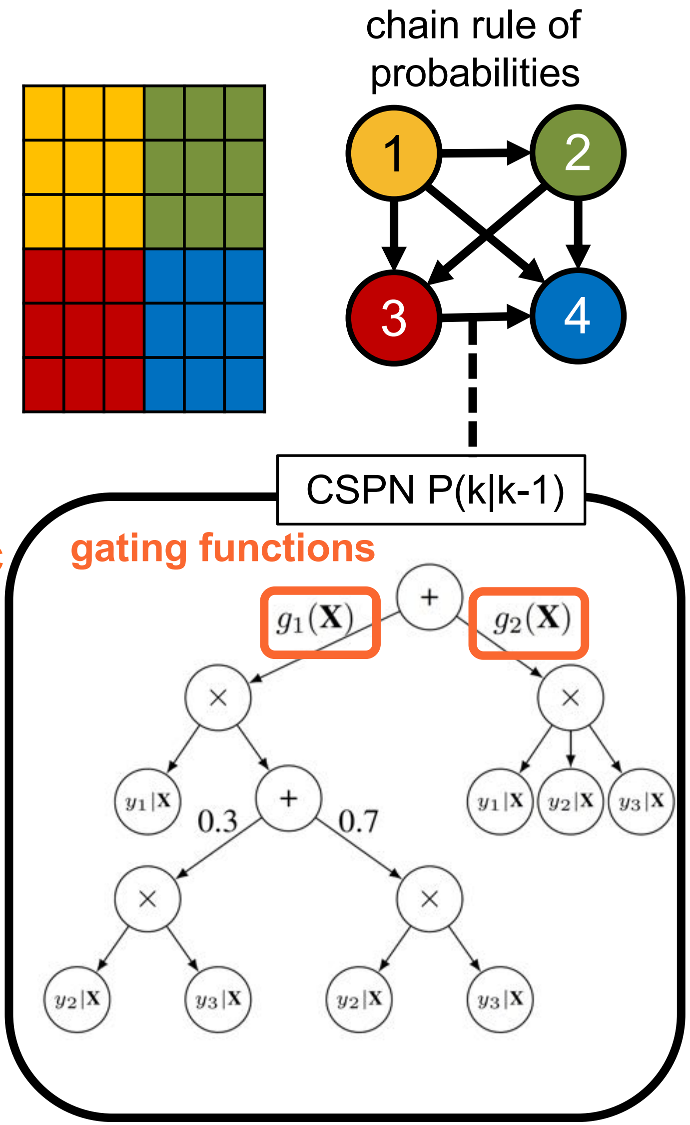


Use nonparametric independency tests and piece-wise linear approximations of the univariate distributions in the leaves



Putting a little bit of structure into SPN models allows one to realize autoregressive deep models akin to PixelCNNs [van den Oord et al. NIPS 2016]

CSPNs
PixelCNNs



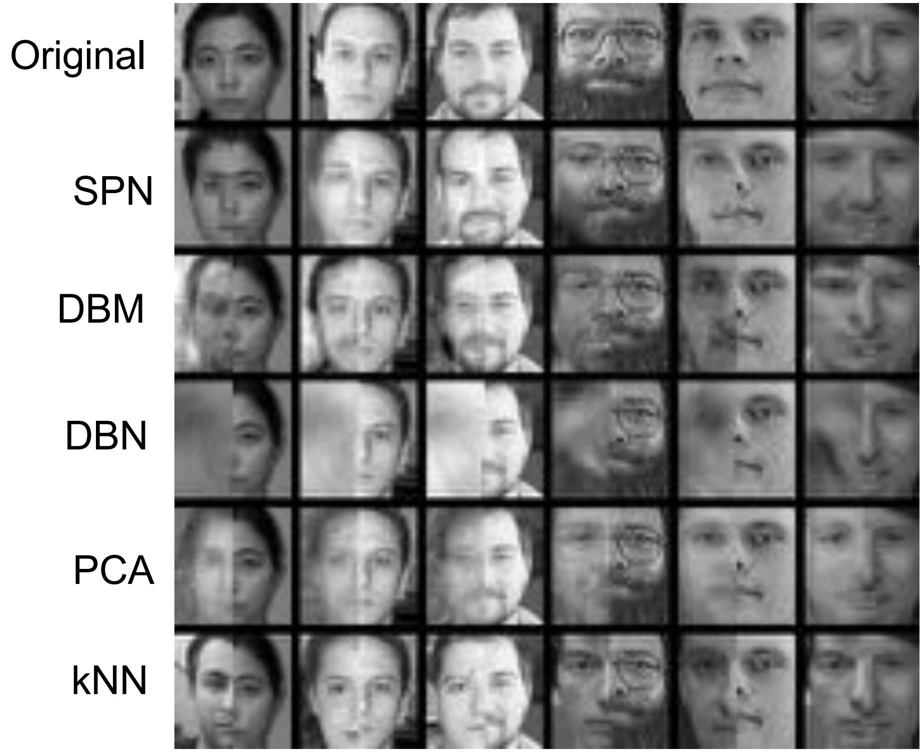
Learn Conditional SPN (CSPNs) by non-parametric conditional independence testing and conditional clustering [Zhang et al. UAI 2011; Lee, Honovar UAI 2017; He et al. ICDM 2017; Zhang et al. AAAI 2018; Runge AISTATS 2018] **encoded using gating functions**

Conditional SPNs

[Shao, Molina, Vergari, Peharz, Kersting 2019]

CAML DFG

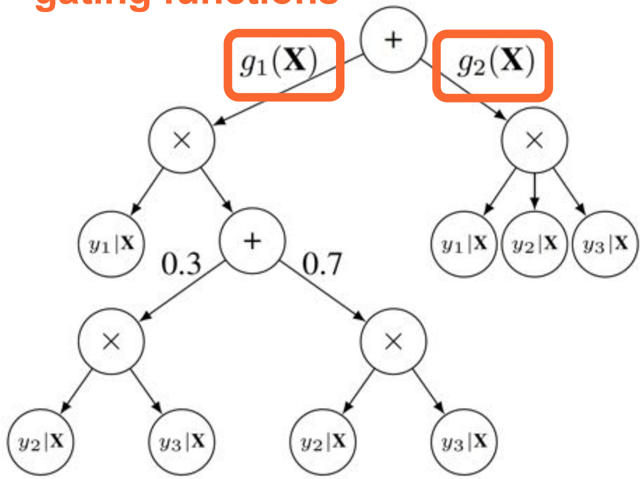
[Poon, Domingos UAI’11]



Gating functions encoded as deep network

Learn Conditional SPN (CSPNs) by non-parametric conditional independence testing and conditional clustering [Zhang et al. UAI 2011; Lee, Honovar UAI 2017; He et al. ICDM 2017; Zhang et al. AAAI 2018; Runge AISTATS 2018] **encoded using gating functions**

gating functions



Conditional SPNs

[Shao, Molina, Vergari, Peharz, Kersting 2019]

CAML DFG

What have we learnt about SPNs?

TECHNISCHE UNIVERSITÄT DARMSTADT

Sum-product networks (SPNs)

- DAG of sums and products
- They are instances of Arithmetic Circuits (ACs)
- Compactly represent partition function
- Learn many layers of hidden variables

Efficient marginal inference

Easy learning

Can outperform well-known alternatives e.g. faster Attend-Infer-Repeat models [Stelzner, Peharz, Kersting ICML 2019]