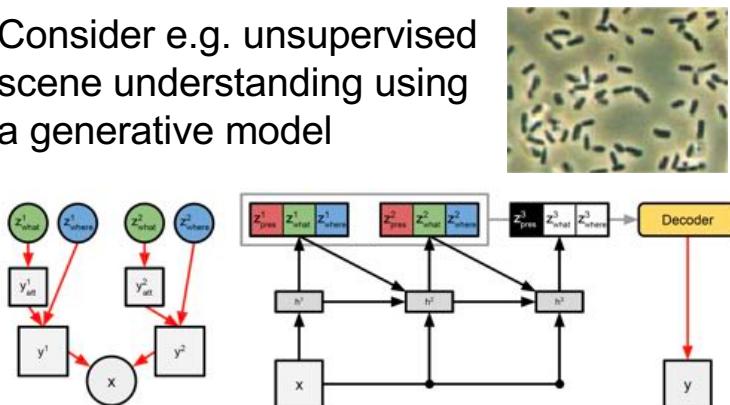


Deep Machines that know when they do not know

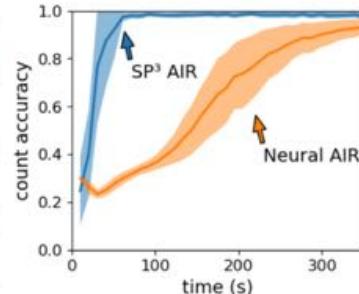


Consider e.g. unsupervised scene understanding using a generative model



[Attend-Infer-Repeat (AIR) model, Hinton et al. NIPS 2016]

Sum-Product Probabilistic Programming:
Making machine learning and data science easier [Stelzner, Molina, Peharz, Vergari, Trapp, Valera, Ghahramani, Kersting ProgProb 2018]



Probabilistic Programming:
Easier modelling by programming generative models in a high-level, prob. language

```
def prior_step(t):
    # Sample object pose. This is a 3-dimensional vector representing x,y position and size.
    z_where = pyro.sample('z_where_{}'.format(t),
                          dist.normal(),
                          z_where_prior_mu, z_where_prior_sigma)

    # Sample object code. This is a 50-dimensional vector.
    z_what = pyro.sample('z_what_{}'.format(t),
                         dist.normal(),
                         z_what_prior_mu, z_what_prior_sigma)

y_att = decode(z_what)
# Map latent code to pixel space using the neural net
```

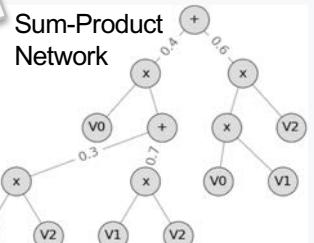
Use deep probabilistic models that feature tractable, deterministic inference

```
from spn.structure.leaves.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
```



Kristian Kersting

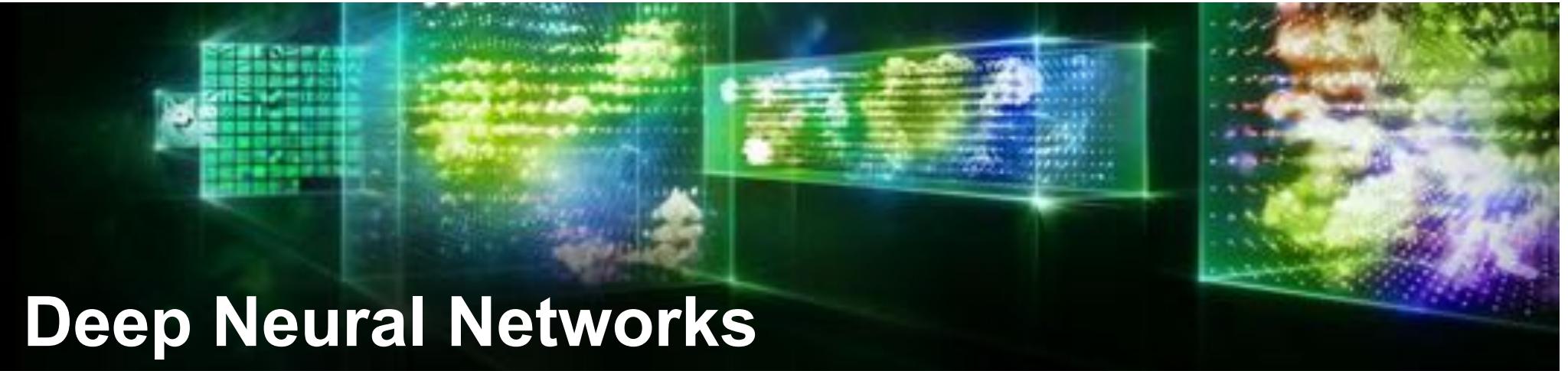
AI and ML have a strong impact



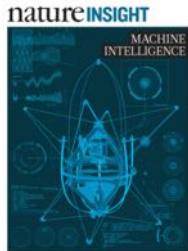
Data are now ubiquitous; there is great value from understanding this data, building models and making predictions

However, there are not enough data scientists, statisticians, machine learning and AI experts

Provide the foundations, algorithms, and tools to develop systems that ease or even automate AI model discovery from data as much as possible

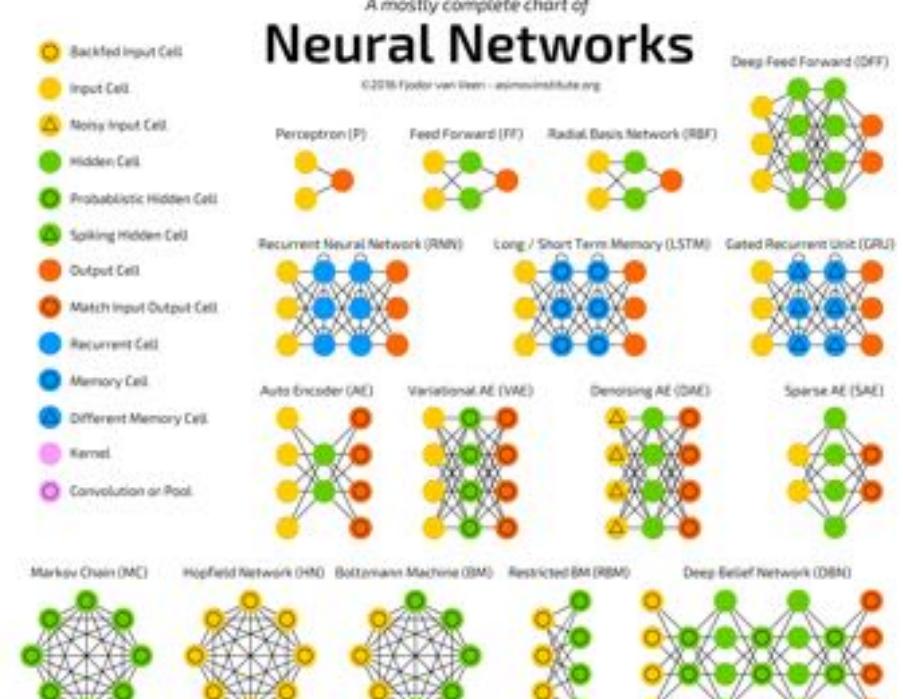
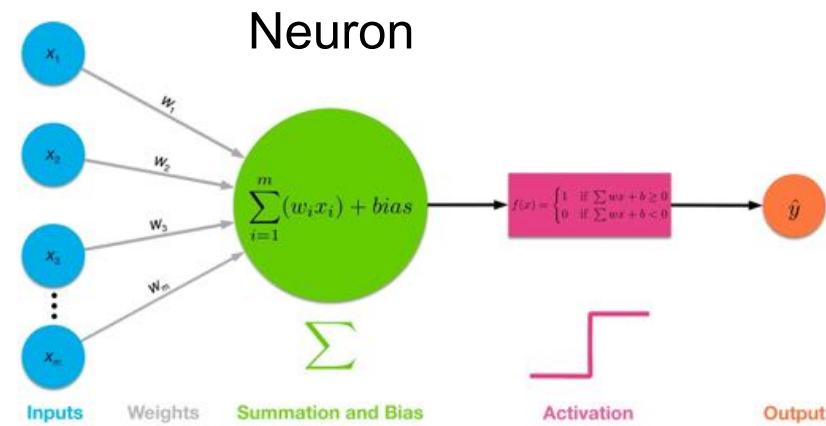


Deep Neural Networks

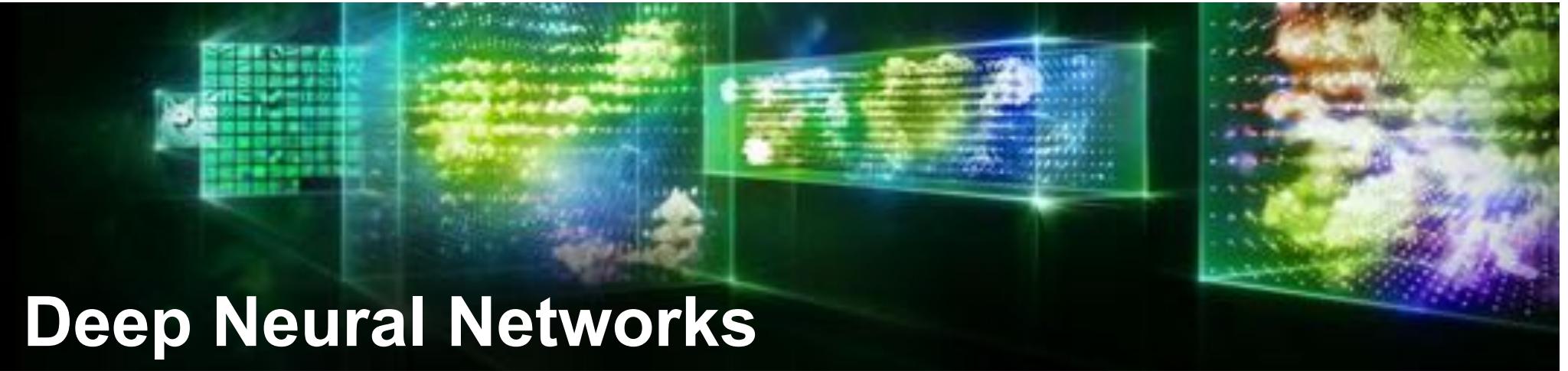


Potentially much more powerful than shallow architectures, represent computations

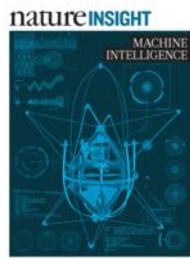
[LeCun, Bengio, Hinton Nature 521, 436–444, 2015]



Differentiable Programming

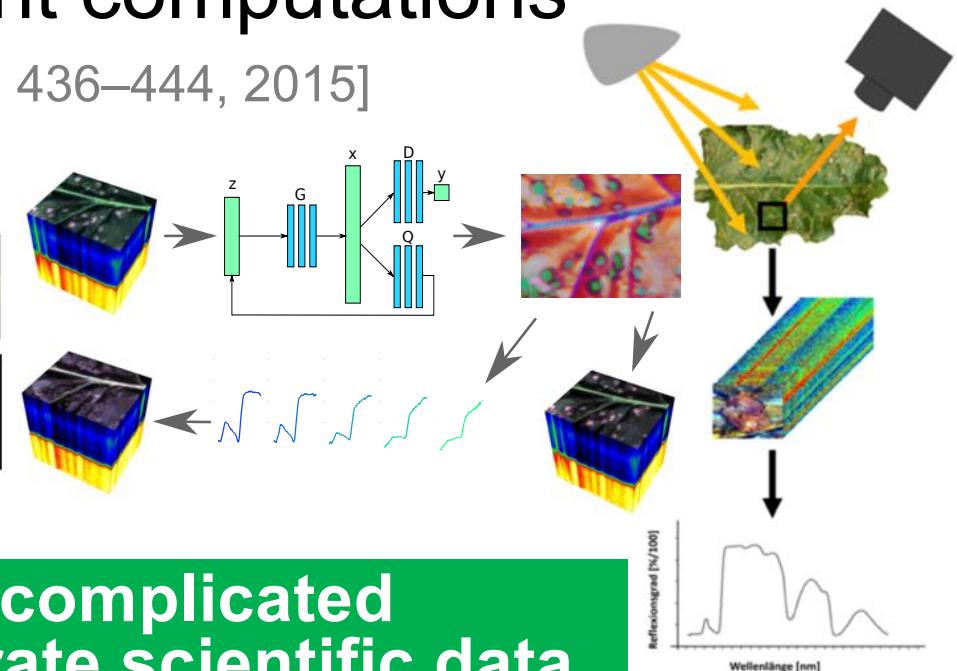
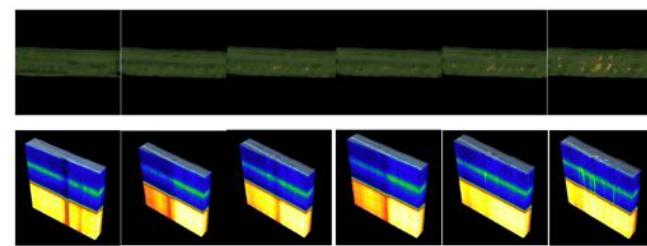
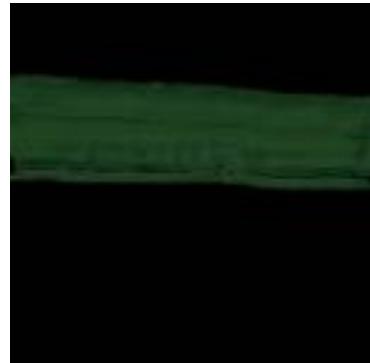


Deep Neural Networks



Potentially much more powerful than shallow architectures, represent computations

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



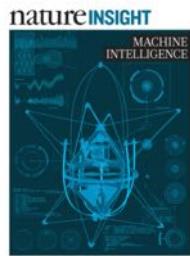
They “develop intuition” about complicated biological processes and generate scientific data

[Schramowski, Brugger, Mahlein, Kersting 2019]

DePhenSe

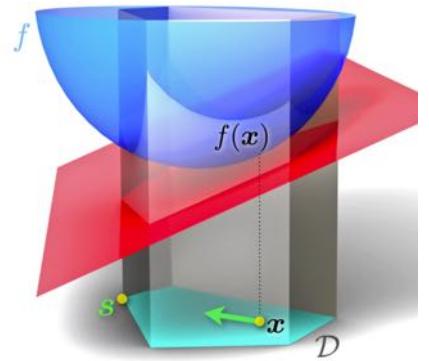
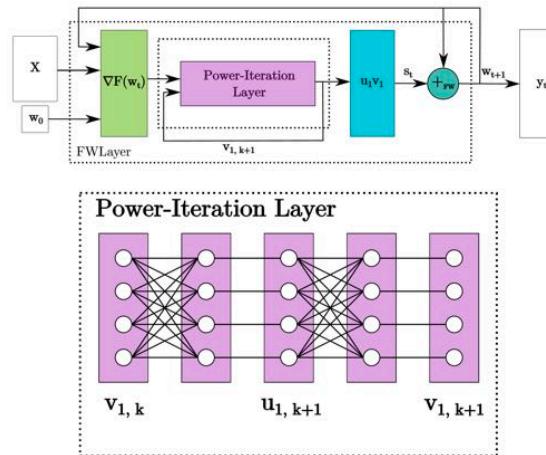
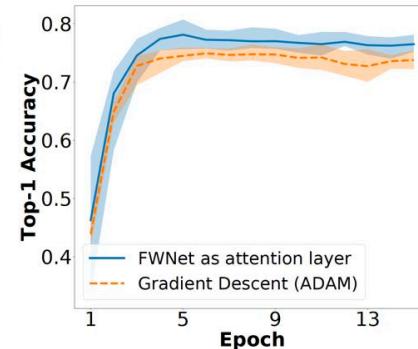
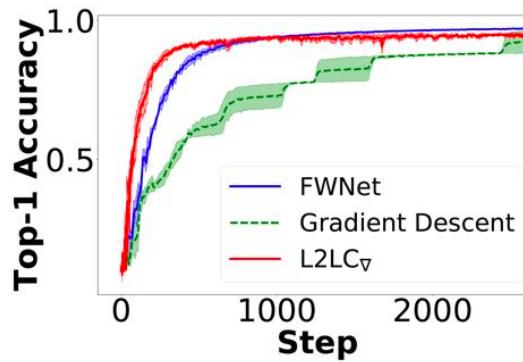


Deep Neural Networks



Potentially much more powerful than shallow architectures, represent computations

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

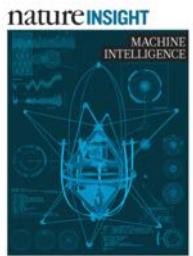


They “invent” constrained optimizers

[Schramowski, Bauckhage, Kersting arXiv:1803.04300, 2018]

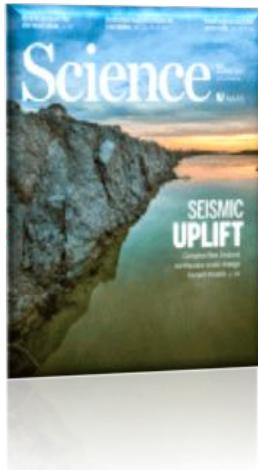


Deep Neural Networks



Potentially much more powerful than shallow architectures, represent computations

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



SHARE

REPORTS

PSYCHOLOGY



1.02k



Semantics derived automatically from language corpora contain human-like biases



0

Aylin Caliskan^{1,*}, Joanna J. Bryson^{1,2,*}, Arvind Narayanan^{1,*}

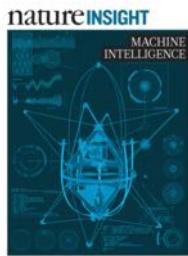
* See all authors and affiliations

Science 14 Apr 2017;
Vol. 356, Issue 6334, pp. 183-186
DOI: 10.1126/science.aal4230

They “capture” stereotypes from human language



Deep Neural Networks

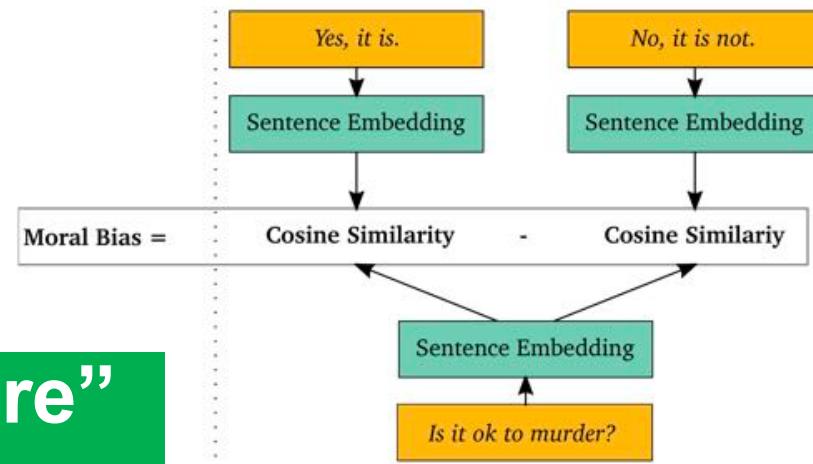


Potentially much more powerful than shallow architectures, represent computations

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

The Moral Choice Machine

Dos	WEAT	Bias	Don'ts	WEAT	Bias
smile	0.116	0.348	rot	-0.099	-1.118
sightsee	0.090	0.281	negative	-0.101	-0.763
cheer	0.094	0.277	harm	-0.110	-0.730
celebrate	0.114	0.264	damage	-0.105	-0.664
picnic	0.093	0.260	slander	-0.108	-0.600
snuggle	0.108	0.238	slur	-0.109	-0.569



But lucky they also “capture” our moral choices

[Jentzsch, Schramowski, Rothkopf, Kersting AIES 2019]

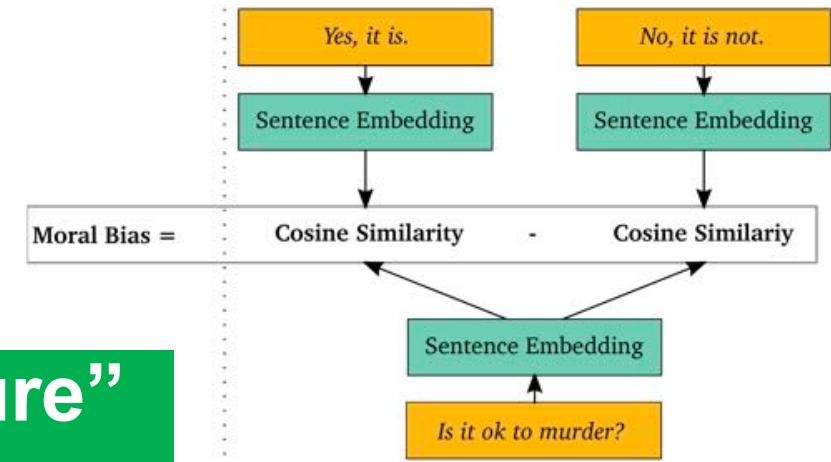


AAAI / ACM conference on
ARTIFICIAL INTELLIGENCE,
ETHICS, AND SOCIETY



The Moral Choice Machine

Dos	WEAT	Bias	Don'ts	WEAT	Bias
smile	0.116	0.348	rot	-0.099	-1.118
sightsee	0.090	0.281	negative	-0.101	-0.763
cheer	0.094	0.277	harm	-0.110	-0.730
celebrate	0.114	0.264	damage	-0.105	-0.664
picnic	0.093	0.260	slander	-0.108	-0.600
snuggle	0.108	0.238	slur	-0.109	-0.569



But lucky they also “capture” our moral choices

[Jentzsch, Schramowski, Rothkopf, Kersting AIES 2019]



AAAI / ACM conference on
ARTIFICIAL INTELLIGENCE,
ETHICS, AND SOCIETY

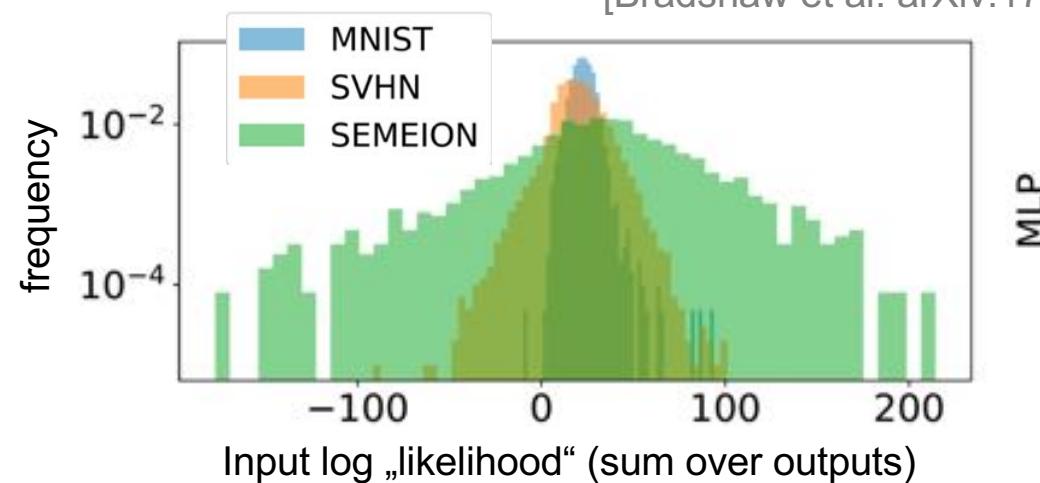
Deep neural networks do not quantify their uncertainty They are not calibrated probabilistic models



Train & Evaluate

Transfer Testing

[Bradshaw et al. arXiv:1707.02476 2017]



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

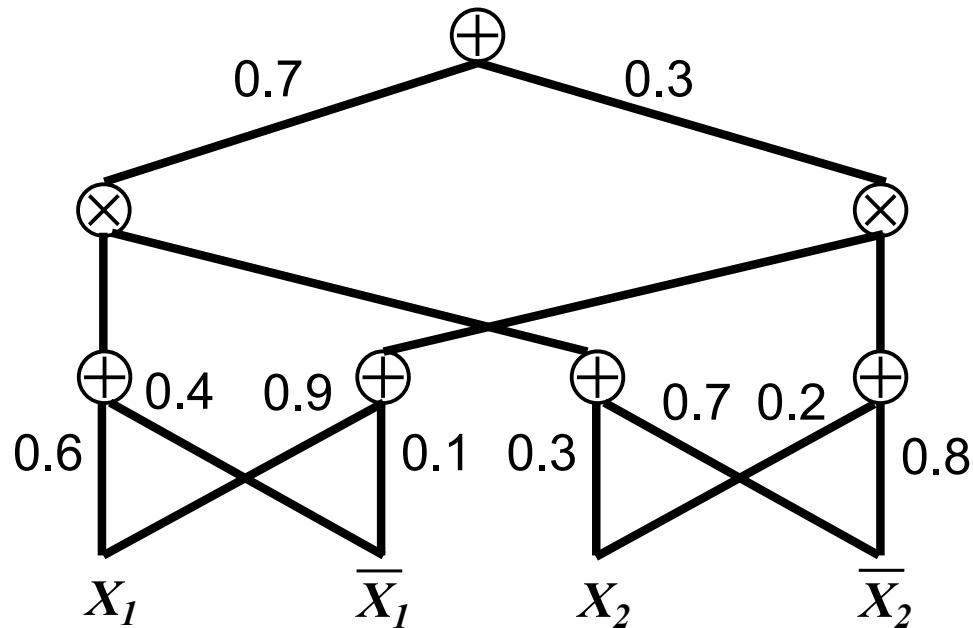
**Getting deep systems that know
when they don't know.**

Can we borrow ideas from deep learning for probabilistic graphical models?



Judea Pearl, UCLA
Turing Award 2012

This results in Sum-Product Networks, a deep probabilistic learning framework

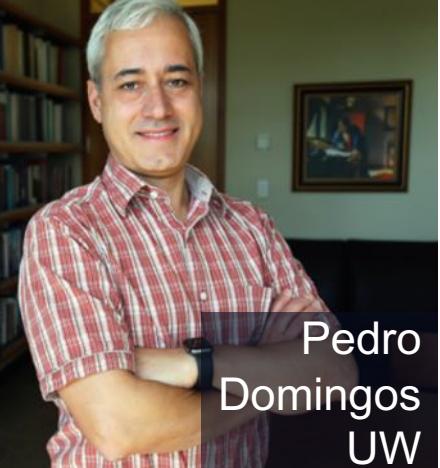


Computational graph
(kind of TensorFlow graphs) that encodes
how to compute
probabilities

Inference is linear in size of network



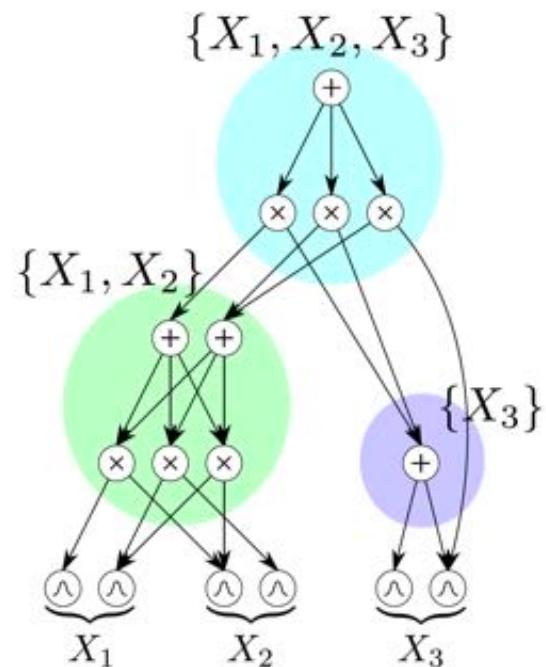
This results in Sum-Product Networks, a deep probabilistic learning framework



- (+) ... convex sum
- (\times) ... product
- (\wedge) ... distribution

completeness
sum children: same scope

decomposability
product children:
non-overlapping scope



Computational graph
(kind of TensorFlow
graphs) that encodes
how to compute
probabilities

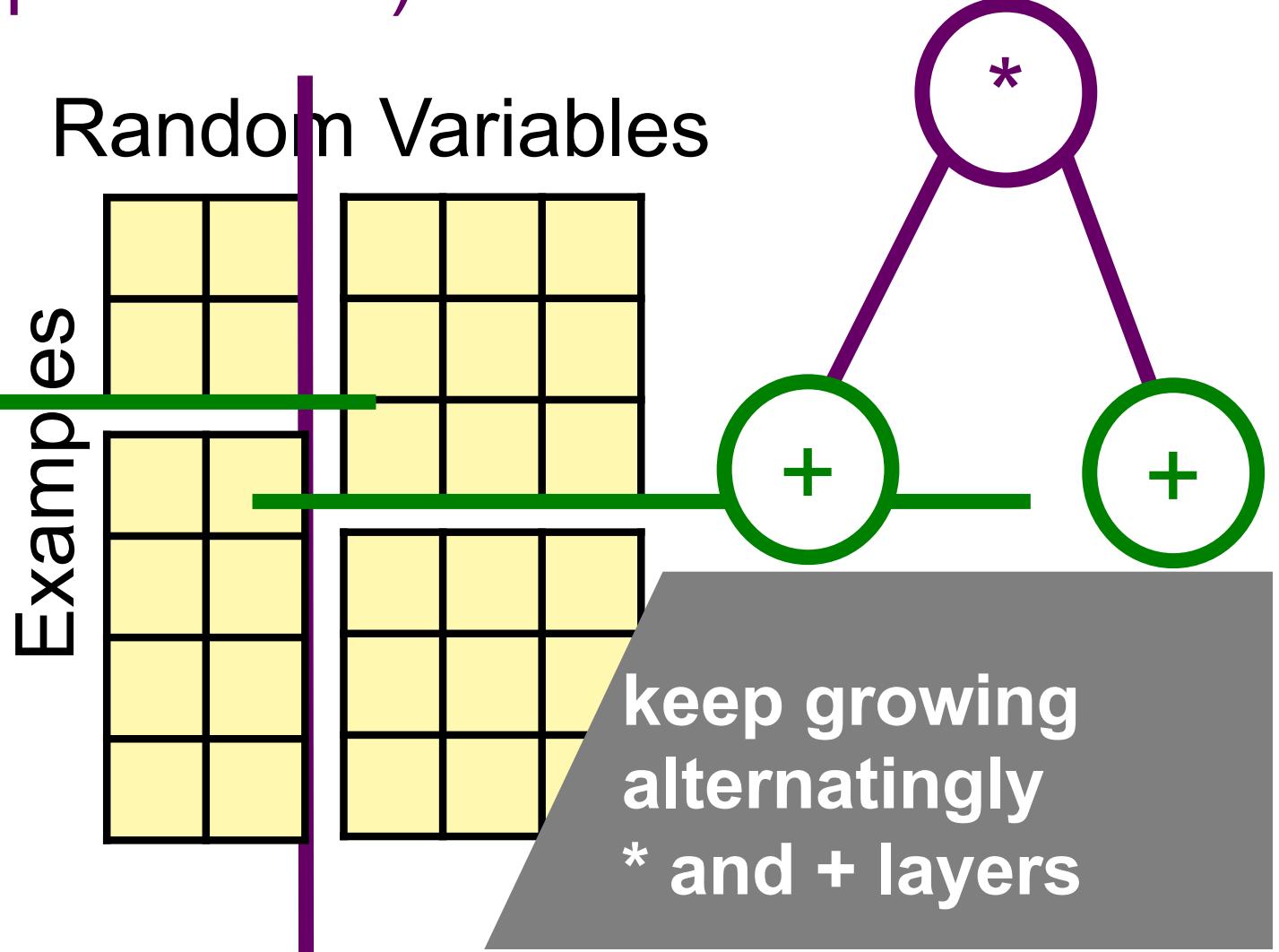
Inference is linear in size of network



And there is a way to select models

Testing independence of random variables
using e.g. (nonparametric) tests

Conditioning,
e.g., via
clustering



[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]

FL⁺ SPFlow: An Easy and Extensible Library ⊗W for Sum-Product Networks



UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO



Max Planck Institute for
Intelligent Systems



UNIVERSITY OF
CAMBRIDGE



VECTOR
INSTITUTE

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



Federal Ministry
of Education
and Research

195 commits

2 branches

0 releases

All 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 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)

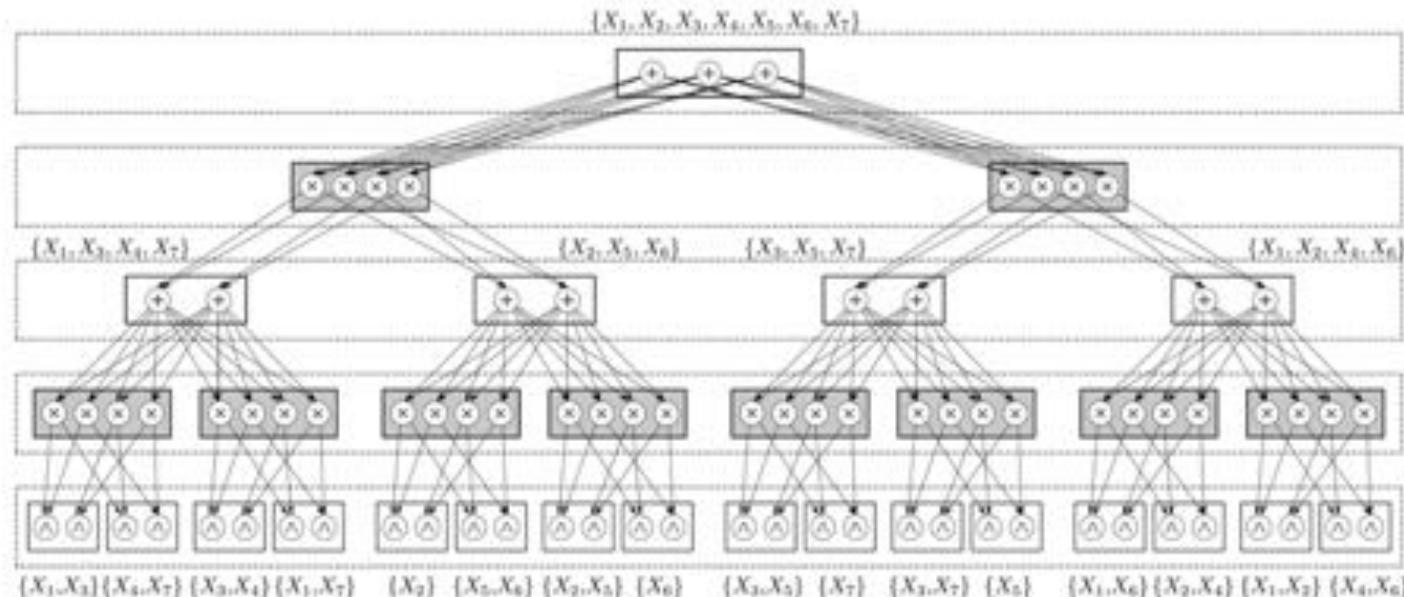
return spn
```

**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 engines like message-passing, conditionals and incremental most probable explanations (IMEs) along with common

Random sum-product networks

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



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

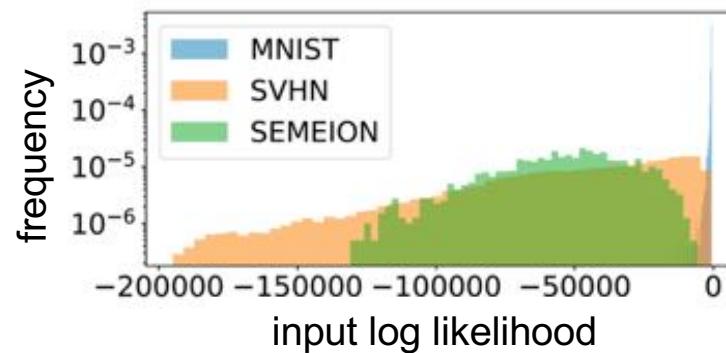
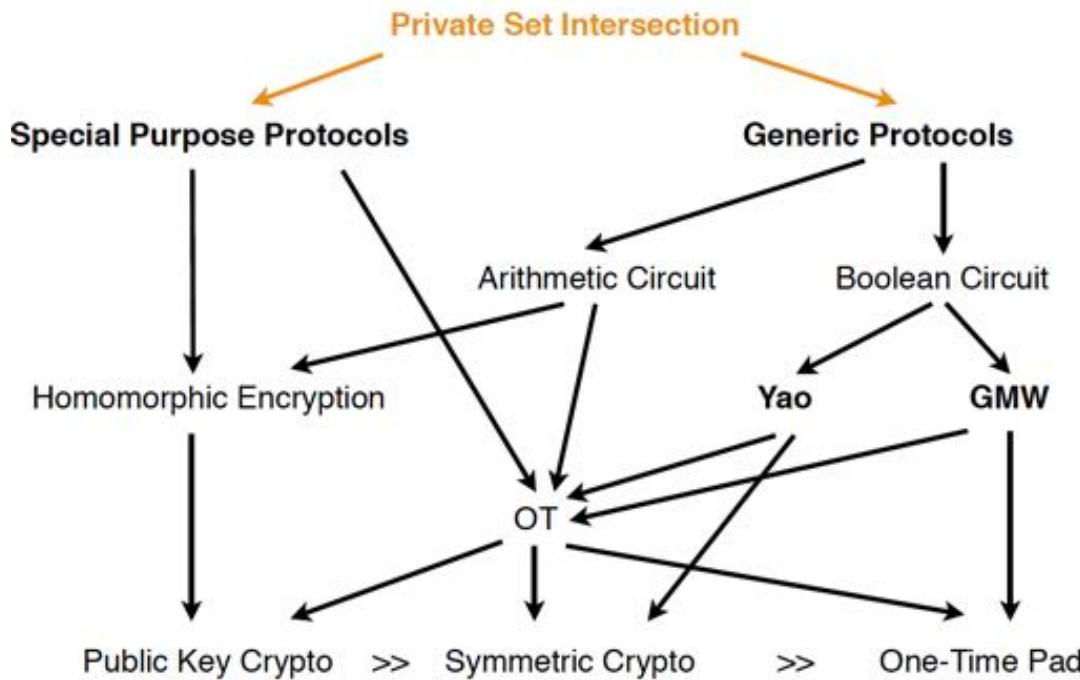


TABLE II
PERFORMANCE COMPARISON. BEST END-TO-END THROUGHPUTS (T), EXCLUDING THE CYCLE COUNTER MEASUREMENTS, ARE DENOTED BOLD.

Dataset	Rows	CPU (μs)	T-CPU (rows/ μs)	CPUF (μs)	T-CPUF (rows/ μs)	GPU (μs)	T-GPU (rows/ μs)	FPGA Cycle Counter	FPGAC (μs)	T-FPGAC (rows/ μs)	FPGA (μs)	T-FPGA (rows/ μs)
Accidents	17009	2798.27				7.87	63090.94	0.27			696.00	24.44
Audio	20000	4271.78				5.4			20317	1	761.00	26.28
Netflix	20000	4892.22				4.8			20322	1	654.00	30.58
MSNBC200	388434	15476.05				30.5			388900	19	608.00	77.56
MSNBC300	388434	10060.78				41.2			388810	19	933.00	78.74
NLTCS	21574	791.80				31.3			21904	1	566.00	38.12
Plants	23215	3621.71	6.41	3521.04		6.59	67004.41	0.35			778.00	29.84
NIPS5	10000	25.11	398.31	26.37		379.23	8210.32	1.22			337.30	29.65
NIPS10	10000	83.60	119.61	84.39		118.49	11550.82	0.87			464.30	21.54
NIPS20	10000	191.30	52.27	182.73	54.72	18689.04	0.54				543.60	18.40
NIPS30	10000	387.61	25.80	349.84	28.58	25355.93	0.39				592.30	16.88
NIPS40	10000	551.64	18.13	471.26	21.22	30820.49	0.32				632.20	15.82
NIPS50	10000	812.44	12.31	792.13	12.62	36355.60	0.28				720.60	13.88
NIPS60	10000	1046.38	9.56	662.53	15.09	40778.36	0.25				799.20	12.51
NIPS70	10000	1148.17	8.71	1134.80		8.81	46759.26	0.21			858.60	11.65
NIPS80	10000	1556.99	6.42	1277.81		7.83	63217.99	0.16			961.80	10.40

How do we do data science offshore?





There are generic protocols to validate computations on authenticated data without knowledge of the secret key

DNA MSPN ####
 Gates: 298208 Yao Bytes: 9542656 Depth: 615

DNA PSPN ####
 Gates: 228272 Yao Bytes: 7304704 Depth: 589

NIPS MSPN ####
 Gates: 1001477 Yao Bytes: 32047264 Depth: 970

Homomorphic sum-product network

[Molina, Weinert, Treiber, Schneider, Kersting 2019]

Learning the Structure of Autoregressive Deep Models such as PixelCNNs

[van den Oord et al. NIPS 2016]



Learn Conditional SPN by testing conditional independence and using conditional clustering, using e.g.
[Zhang et al. UAI 2011; Lee, Honavar UAI 2017; He et al. ICDM 2017; Zhang et al. AAAI 2018; Runge AISTATS 2018]

Conditional SPNs

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

Conditioning
Result



Learn Conditional SPN by testing conditional independence and using conditional clustering, using e.g.
[Zhang et al. UAI 2011; Lee, Honavar UAI 2017; He et al. ICDM 2017; Zhang et al. AAAI 2018; Runge AISTATS 2018]

Conditional SPNs

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

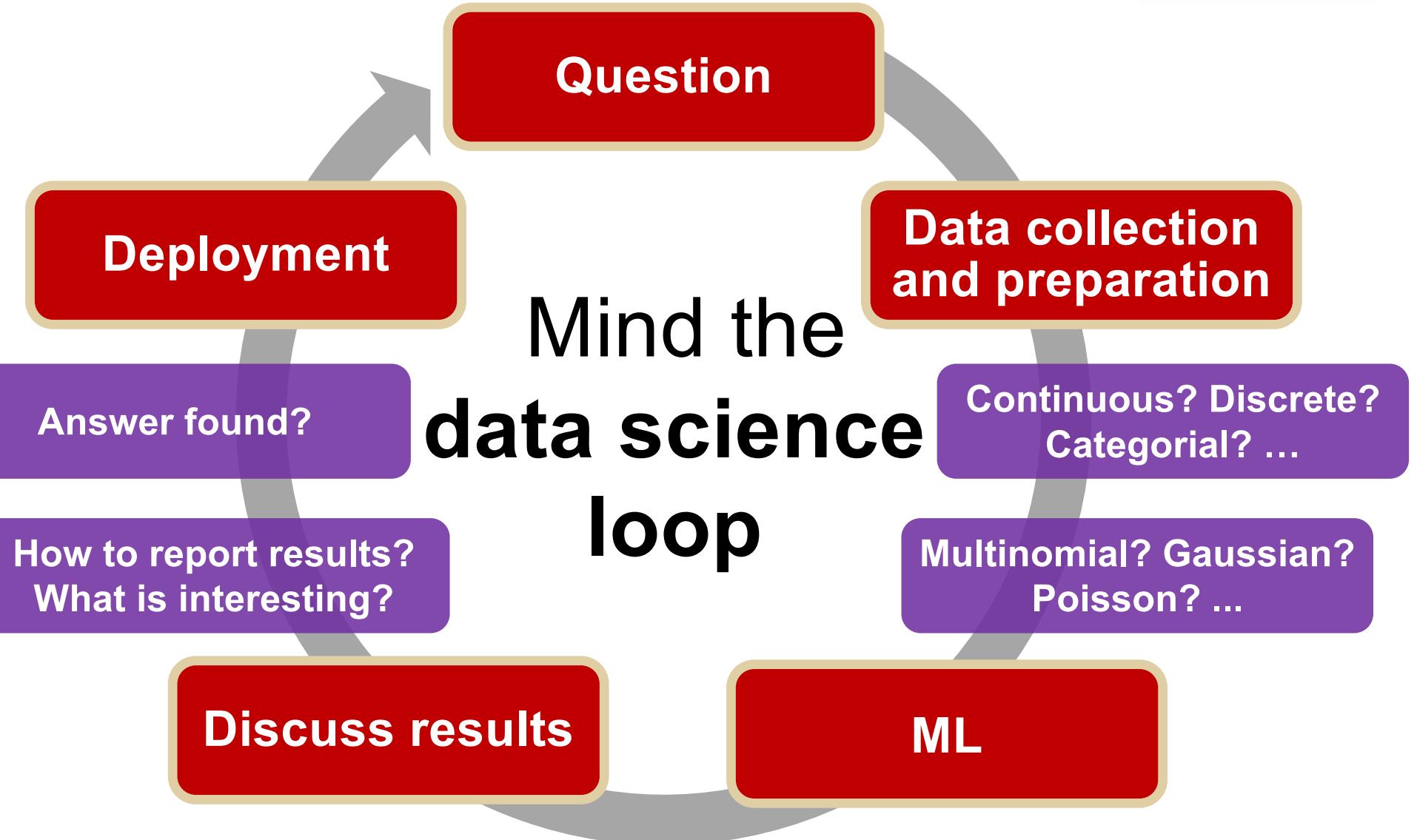
Functional weights realized as neural network



**Learn Conditional SPN by testing conditional independence and using conditional clustering, using e.g.
[Zhang et al. UAI 2011; Lee, Honavar UAI 2017; He et al. ICDM 2017; Zhang et al. AAAI 2018; Runge AISTATS 2018]**

Conditional SPNs

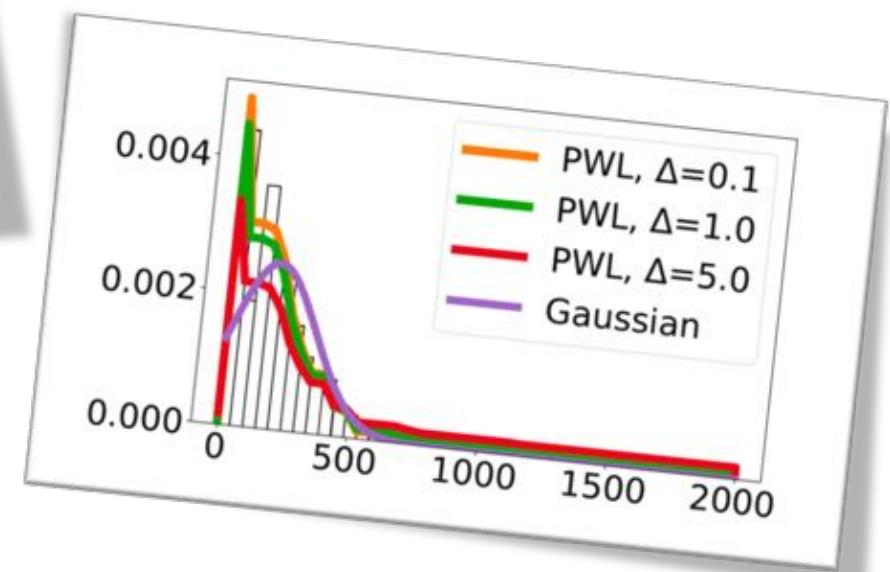
[Shao, Molina, Vergari, Peharz, Kersting 2019]



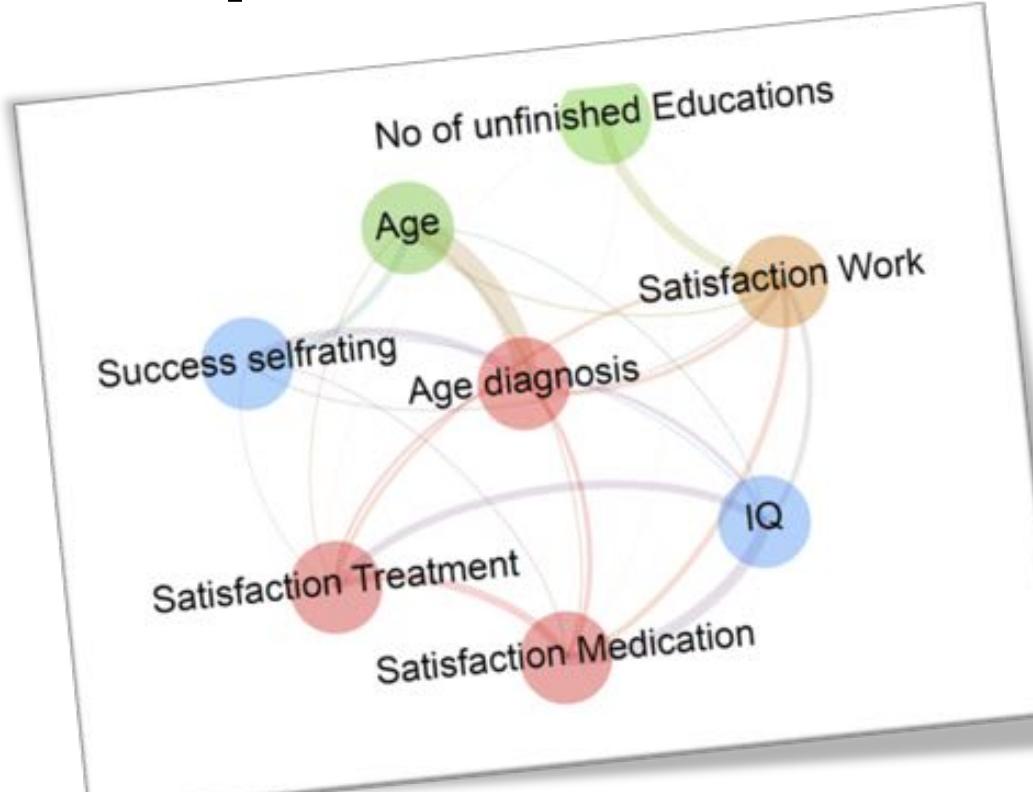
Distribution-agnostic Deep Probabilistic Learning



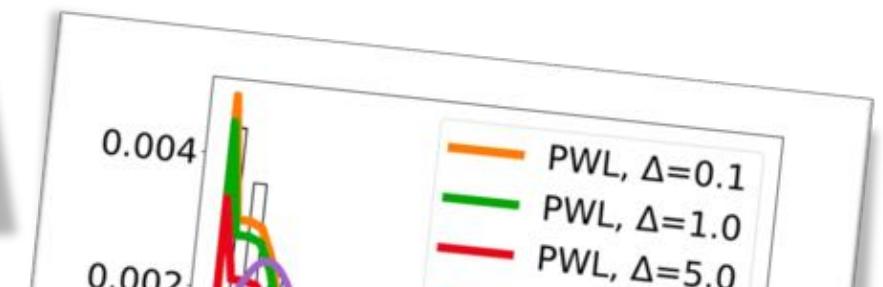
Use nonparametric independency tests and piece-wise linear approximations



Distribution-agnostic Deep Probabilistic Learning



Use nonparametric independency tests and piece-wise linear approximations



However, we have to provide the statistical types and do not gain insights into the parametric forms of the variables.
Are they Gaussians? Gammas? ...

The Explorative Automatic Statistician



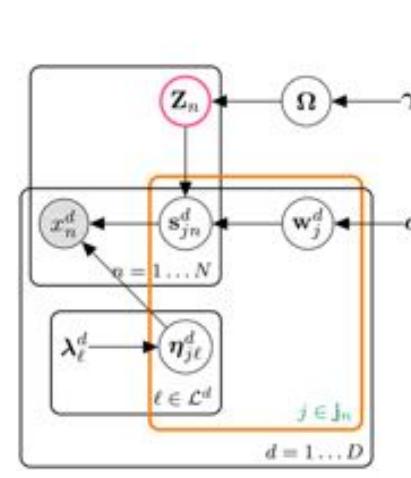
UNIVERSITY OF
CAMBRIDGE



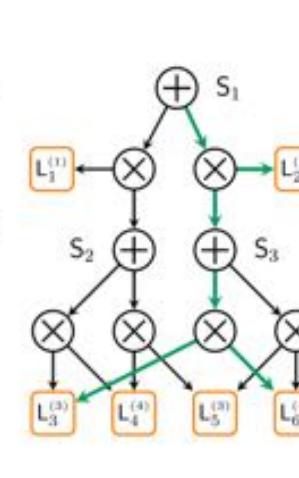
TECHNISCHE
UNIVERSITÄT
DARMSTADT

	X^1	X^2	X^3	X^4	X^5
x_8					
x_7			?		
x_6					
missing value	x_5	?			
x_4			?		
x_3					
x_2		?			
x_1					

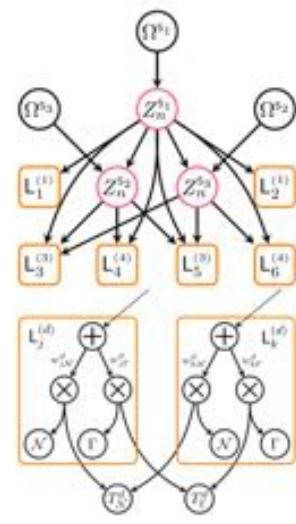
We can even automatically discovers the statistical types and parametric forms of the variables



Bayesian Type Discovery



Mixed Sum-Product Network



Automatic Statistician

That is, the machine understands the data with few expert input ...

The screenshot shows a user interface for a DeepNotebook. At the top, there are three buttons: 'Toggle Introduction', 'Toggle explanations', and 'Toggle Code'. Below these, the title 'Exploring the Titanic dataset' is displayed in a large, bold font. A detailed description of the dataset follows:

This report describes the dataset Titanic and contains general statistical information and an analysis on the influence different features and subgroups of the data have on each other. The first part of the report contains general statistical information about the dataset and an analysis of the variables and probability distributions. The second part focusses on a subgroup analysis of the data. Different clusters identified by the network are analyzed and compared to give an insight into the structure of the data. Finally the influence different variables have on the predictive capabilities of the model are analyzed. The whole report is generated by fitting a sum product network to the data and extracting all information from this model.

Völker: "DeepNotebooks – Interactive data analysis using Sum-Product Networks." MSc Thesis, TU Darmstadt, 2018

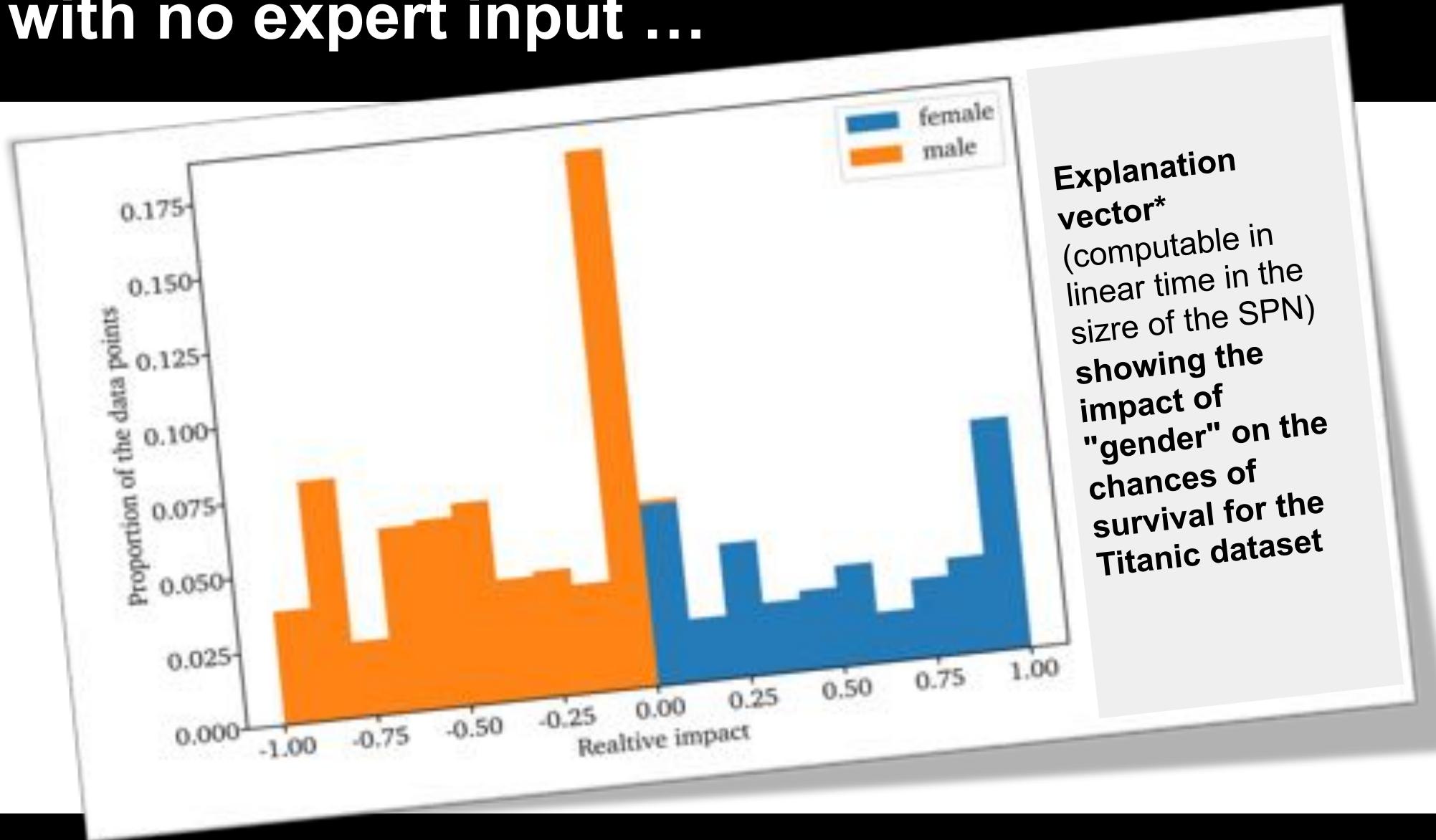


TECHNISCHE
UNIVERSITÄT
DARMSTADT

Report framework created @ TU Darmstadt

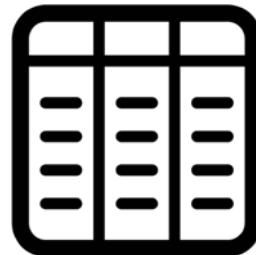
...and can compile data reports automatically

The machine understands the data with no expert input ...



...and can compile data reports automatically

P(heart attack |)?



The New York Times

f t e ↗ 📒

Opinion

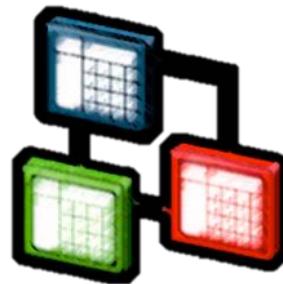
A.I. Is Harder Than You Think and Data Science

By Gary Marcus and Ernest Davis

Mr. Marcus is a professor of psychology and neural science. Mr. Davis is a professor of computer science.

May 18, 2018

P(heart attack |)?



The New York Times

f t e ↗ 📖

Opinion

A.I. Is Harder Than You Think and Data Science

By Gary Marcus and Ernest Davis

Mr. Marcus is a professor of psychology and neural science. Mr. Davis is a professor of computer science.

May 18, 2018

This image shows a screenshot of a New York Times Opinion article. The title of the article is "A.I. Is Harder Than You Think and Data Science". It is written by Gary Marcus and Ernest Davis. The article discusses the relationship between Artificial Intelligence and Data Science. The screenshot includes social media sharing icons for Facebook, Twitter, and Email, as well as a bookmark icon. The date of publication is May 18, 2018.

P(heart
attack |)?



The New York Times

Opinion

A.I. Is Harder Than You Think and Data Science

By Gary Marcus and Ernest Davis

Mr. Marcus is a professor of psychology and neural science. Mr. Davis is a professor of computer science.

May 18, 2018

f t e ↗ 📖

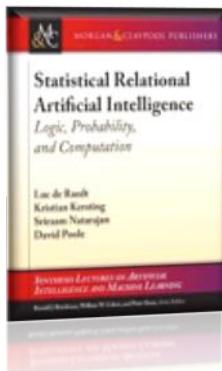
This image shows a screenshot of a New York Times opinion article. The title is "A.I. Is Harder Than You Think and Data Science". It's written by Gary Marcus and Ernest Davis. The author's bio notes that Mr. Marcus is a professor of psychology and neural science, and Mr. Davis is a professor of computer science. The date of publication is May 18, 2018. The article has social sharing icons for Facebook, Twitter, and Email, along with a 'Bookmark' icon. The overall layout is that of a digital newspaper article.

P(heart attack |)?



Crossover of ML and DS with data & programming abstractions

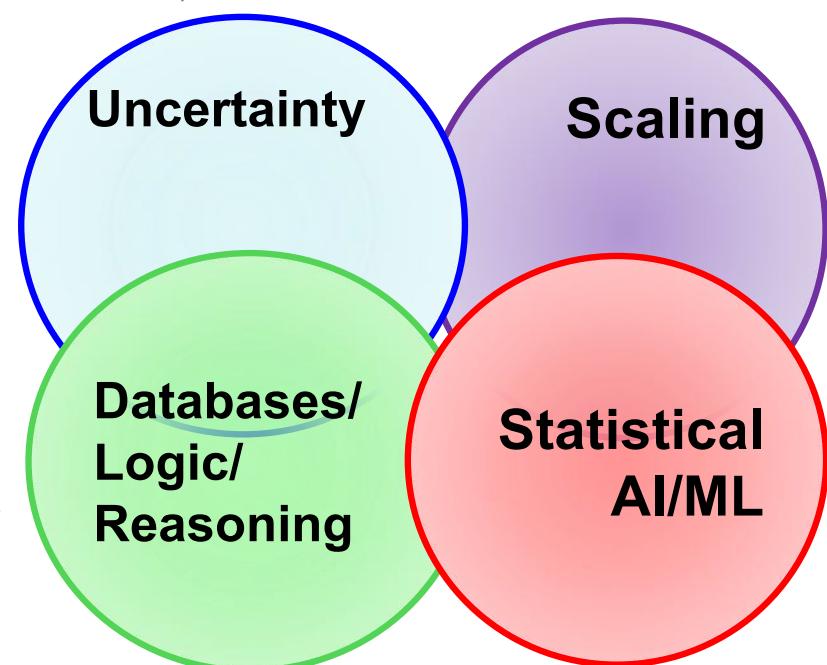
De Raedt, Kersting, Natarajan, Poole: Statistical Relational Artificial Intelligence: Logic, Probability, and Computation. Morgan and Claypool Publishers, ISBN: 9781627058414, 2016.

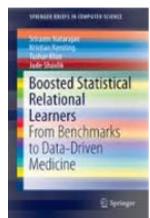


**building general-purpose
data science and ML
machines**

**make the ML/DS expert
more effective**

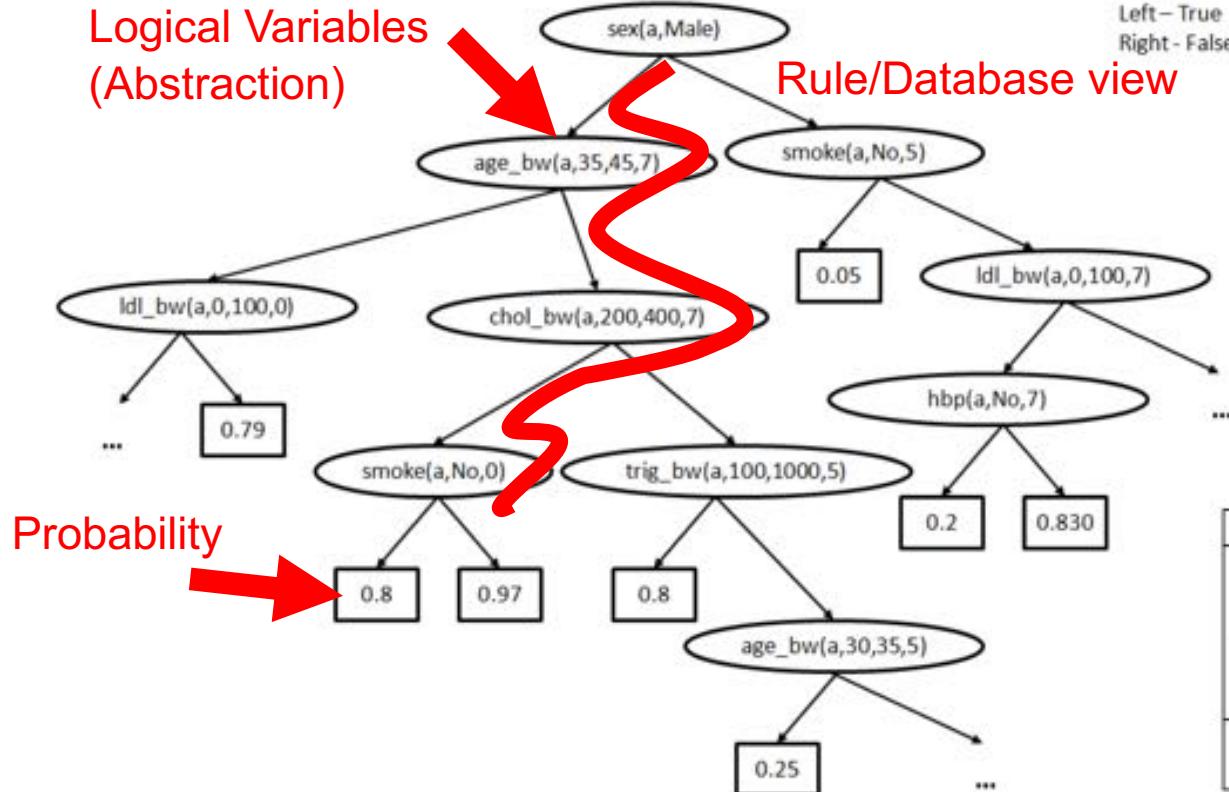
**increases the number of
people who can
successfully build ML/DS
applications**





Understanding Electronic Health Records

Atherosclerosis is the cause of the majority of Acute Myocardial Infarctions (heart attacks)



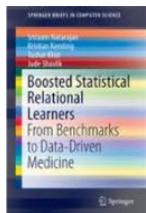
[Circulation; 92(8), 2157-62, 1995;
JACC; 43, 842-7, 2004]

Algorithm	Accuracy	AUC-ROC	The higher, the better
J48	0.667	0.607	
SVM	0.667	0.5	
AdaBoost	0.667	0.608	
Bagging	0.677	0.613	
NB	0.75	0.653	
RPT	0.669*	0.778	
RFGB	0.667*	0.819	

25%

Algorithm for Mining Markov Logic Networks	Likelihood The higher, the better	AUC-ROC The higher, the better	AUC-PR The higher, the better	Time The lower, the better	state-of-the-art
Boosting	0.81 11%	0.96 78%	0.93 50%	9s 37200x	
LSM	0.73 	0.54 	0.62 	93 hrs faster	

[Kersting, Driessens ICML'08; Karwath, Kersting, Landwehr ICDM'08; Natarajan, Joshi, Tadepelli, Kersting, Shavlik. IJCAI'11; Natarajan, Kersting, Ip, Jacobs, Carr IAAI '13; Yang, Kersting, Terry, Carr, Natarajan AIME '15; Khot, Natarajan, Kersting, Shavlik ICDM'13, MLJ'12, MLJ'15, Yang, Kersting, Natarajan BIBM'17]



<https://starling.utdallas.edu/software/boostsrl/wiki/>



People

Publications

Projects

Software

Datasets

Blog



BOOSTSRL BASICS

Getting Started

File Structure

Basic Parameters

Advanced Parameters

Basic Modes

Advanced Modes

ADVANCED BOOSTSRL

Default (RDN-Boost)

MLN-Boost

Regression

One-Class Classification

Cost-Sensitive SRL

Learning with Advice

Approximate Counting

Discretization of Continuous-Valued Attributes

Lifted Relational Random Walks

Grounded Relational Random Walks

APPLICATIONS

Natural Language Processing

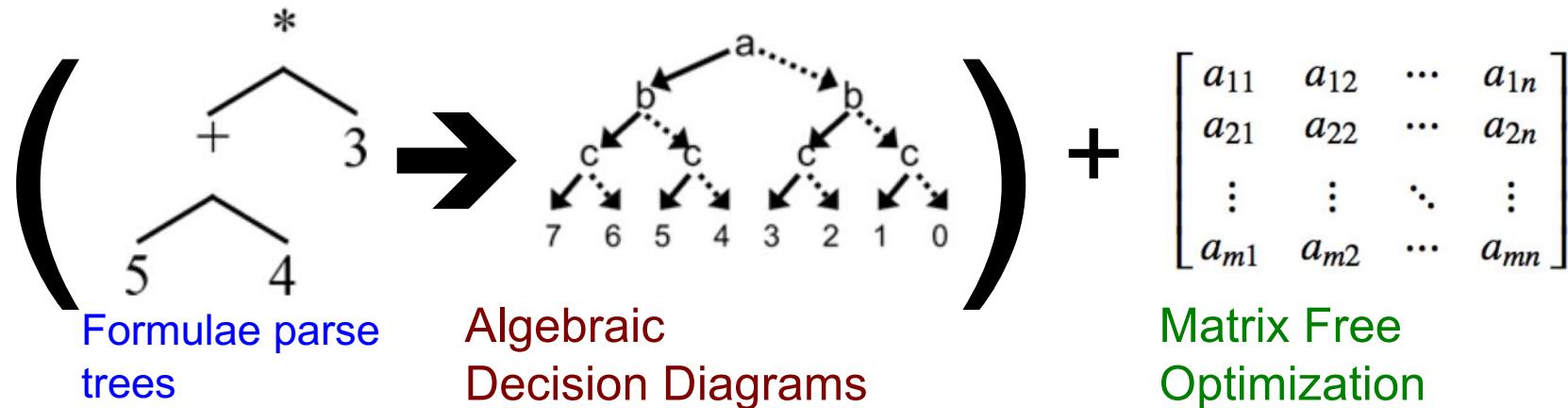
BoostSRL Wiki

BoostSRL (Boosting for Statistical Relational Learning) is a gradient-boosting based approach to learning different types of SRL models. As with the standard gradient-boosting approach, our approach turns the model learning problem to learning a sequence of regression models. The key difference to the standard approaches is that we learn relational regression models i.e., regression models that operate on relational data. We assume the data in a predicate logic format and the output are essentially first-order regression trees where the inner nodes contain conjunctions of logical predicates. For more details on the models and the algorithm, we refer to our book on this topic.

Sriram Natarajan, Tushar Khot, Kristian Kersting and Jude Shavlik, Boosted Statistical Relational Learners: From Benchmarks to Data-Driven Medicine . SpringerBriefs in Computer Science, ISBN: 978-3-319-13643-1, 2015

Human-in-the-loop learning

New field: Probabilistic Programming



name	Problem Statistics			Symbolic IPM ADDI	time[s]	Ground IPM time[s]
	#vars	#constr	$nnz(A)$			
factory	131.072	688.128	4.000.000	1819	6899	516
factory0	524.288	2.752.510	15.510.000	1895	6544	7920
factory1	2.097.150	11.000.000	59.549.700	2406	34749	159730
factory2	4.194.300	22.020.100	119.099.000	2504	36248	$\geq 48\text{hrs.}$

>4.8x faster

Applies to QPs but here illustrated on MDPs for a factory agent which must paint two objects and connect them. The objects must be smoothed, shaped and polished and possibly drilled before painting, each of which actions require a number of tools which are possibly available. Various painting and connection methods are represented, each having an effect on the quality of the job, and each requiring tools. Rewards (required quality) range from 0 to 10 and a discounting factor of 0.9 was used.

In general, computing the exact posterior is intractable, i.e., inverting the generative process to determine the state of latent variables corresponding to an input is time-consuming and error-prone.

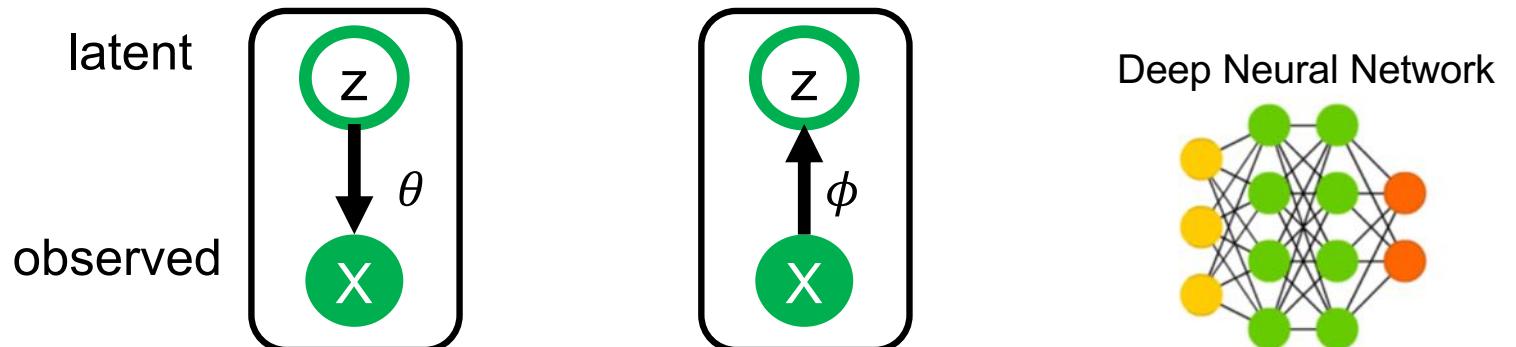
Deep Probabilistic Programming

```
import pyro.distributions as dist

def model(data):
    # define the hyperparameters that control the beta prior
    alpha0 = torch.tensor(10.0)
    beta0 = torch.tensor(10.0)
    # sample f from the beta prior
    f = pyro.sample("latent_fairness", dist.Beta(alpha0, beta0))
    # loop over the observed data
    for i in range(len(data)):
        # observe datapoint i using the bernoulli
        # likelihood Bernoulli(f)
        pyro.sample("obs_{}".format(i), dist.Bernoulli(f), obs=data[i])
```

```
def guide(data):
    # register the two variational parameters with Pyro.
    alpha_q = pyro.param("alpha_q", torch.tensor(15.0),
                         constraint=constraints.positive)
    beta_q = pyro.param("beta_q", torch.tensor(15.0),
                         constraint=constraints.positive)
    # sample latent_fairness from the distribution Beta(alpha_q, beta_q)
    pyro.sample("latent_fairness", dist.Beta(alpha_q, beta_q))
```

(2) Ease the implementation by some high-level, probabilistic programming language



(1) Instead of optimizing variational parameters for every new data point, use a deep network to predict the posterior given X [Kingma, Welling 2013, Rezende et al. 2014]

[Stelzner, Molina, Peharz, Vergari, Trapp, Valera, Ghahramani, Kersting ProgProb 2018]

Sum-Product Probabilistic Programming

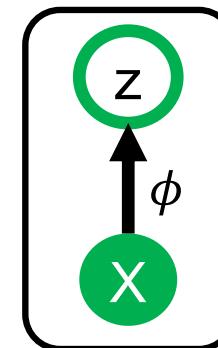
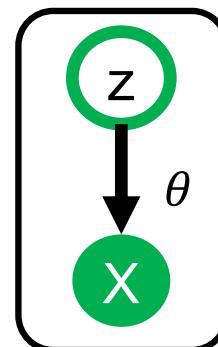
```
import pyro.distributions as dist

def model(data):
    # define the hyperparameters that control the beta prior
    alpha0 = torch.tensor(10.0)
    beta0 = torch.tensor(10.0)
    # sample f from the beta prior
    f = pyro.sample("latent_fairness", dist.Beta(alpha0, beta0))
    # loop over the observed data
    for i in range(len(data)):
        # observe datapoint i using the bernoulli
        # likelihood Bernoulli(f)
        pyro.sample("obs_{}".format(i), dist.Bernoulli(f), obs=data[i])
```

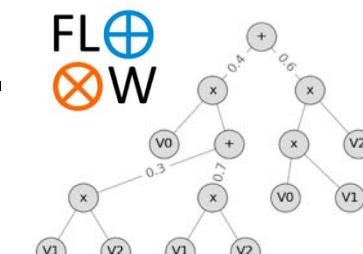
```
def guide(data):
    # register the two variational parameters with Pyro.
    alpha_q = pyro.param("alpha_q", torch.tensor(15.0),
                         constraint=constraints.positive)
    beta_q = pyro.param("beta_q", torch.tensor(15.0),
                         constraint=constraints.positive)
    # sample latent_fairness from the distribution Beta(alpha_q, beta_q)
    pyro.sample("latent_fairness", dist.Beta(alpha_q, beta_q))
```

(2) Ease the implementation by some high-level, probabilistic programming language

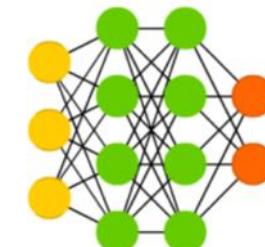
latent
observed



Sum-Product Network



Deep Neural Network

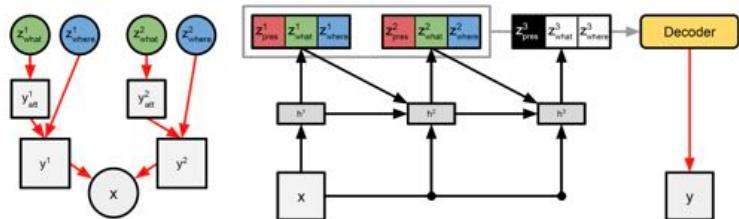


(1) Instead of optimizing variational parameters for every new data point, use a deep network to predict the posterior given X [Kingma, Welling 2013, Rezende et al. 2014]

Unsupervised scene understanding

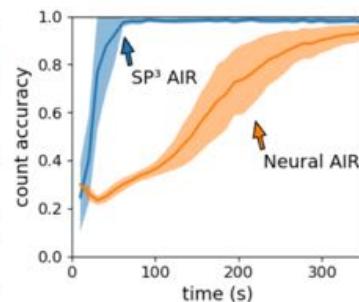
[Stelzner, Peharz, Kersting ICML 2019]

Consider e.g. unsupervised scene understanding using a generative model



[Attend-Infer-Repeat (AIR) model, Hinton et al. NIPS 2016]

Sum-Product Probabilistic Programming:
Making machine learning and data science easier [Stelzner, Molina, Peharz, Vergari, Trapp, Valera, Ghahramani, Kersting ProgProb 2018]



Probabilistic Programming:

Easier modelling by programming generative models in a high-level, prob. language

```
def prior_step(t):
    # Sample object pose. This is a 3-dimensional vector representing x,y position and size.
    z_where = pyro.sample("z_where_{}".format(t),
                          dist.normal,
                          z_where_prior_mu, z_where_prior_sigma)

    # Sample object code. This is a 50-dimensional vector.
    z_what = pyro.sample("z_what_{}".format(t),
                         dist.normal,
                         z_what_prior_mu, z_what_prior_sigma)

y_att = decode(z_what)
```



Map latent code to pixel space using the neural net

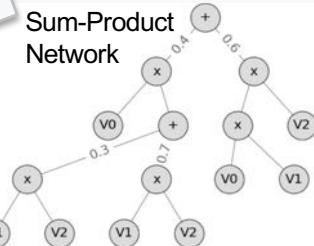
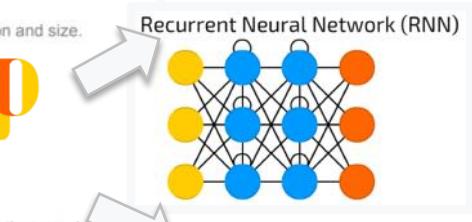
Use deep probabilistic models that feature tractable, deterministic inference

```
from spn.structure.leaves.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])

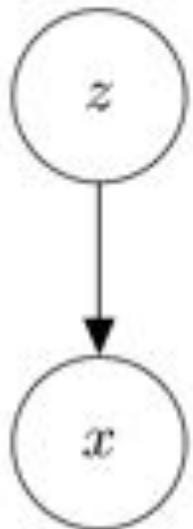
assign_ids(spn)
rebuild_scopes_bottom_up(spn)

return spn
```

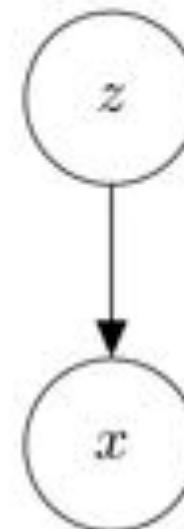


Actually, the main idea is to replace the VAEs within AIR by SPNs

VAE

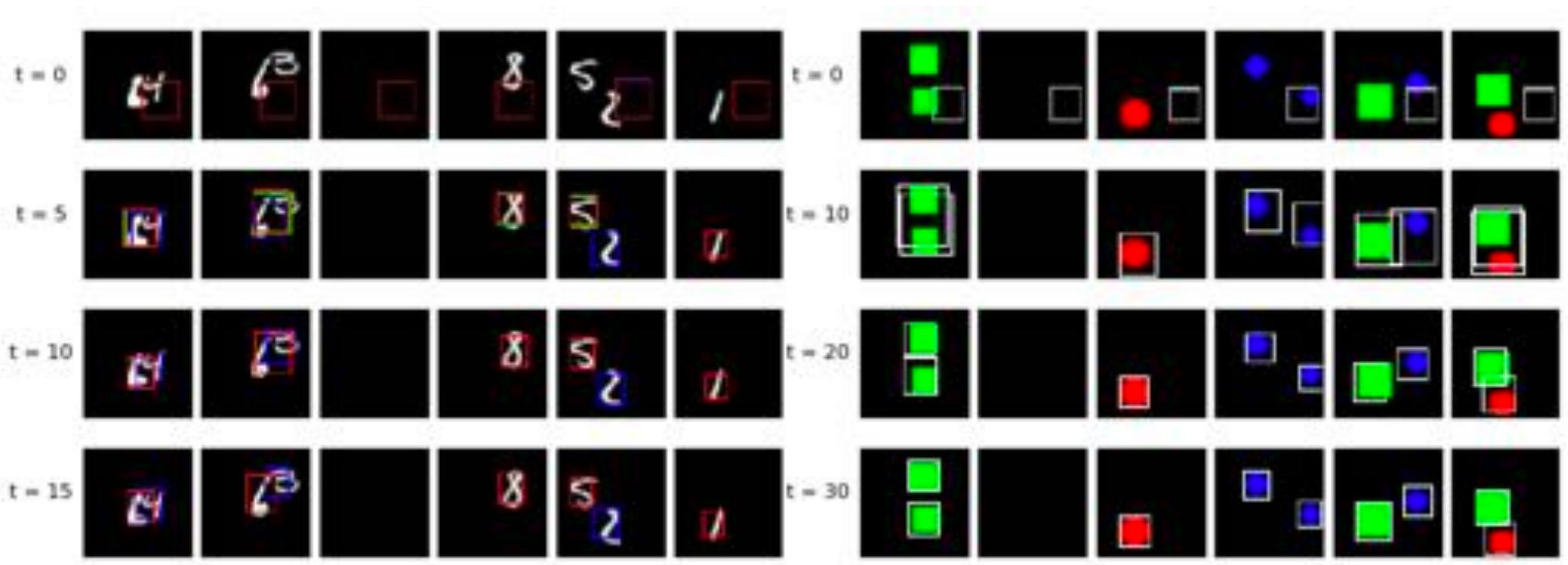


SPN



- infinite mixture model
- intractable density
- intractable posterior
- “large” but finite mixture model
- tractable density
- tractable marginals [Peharz et al., 2015]
- tractable posterior [Vergari et al., 2017]

Sum-Product Attent-Infer Repeat



[Stelzner, Peharz, Kersting ICML 2019]



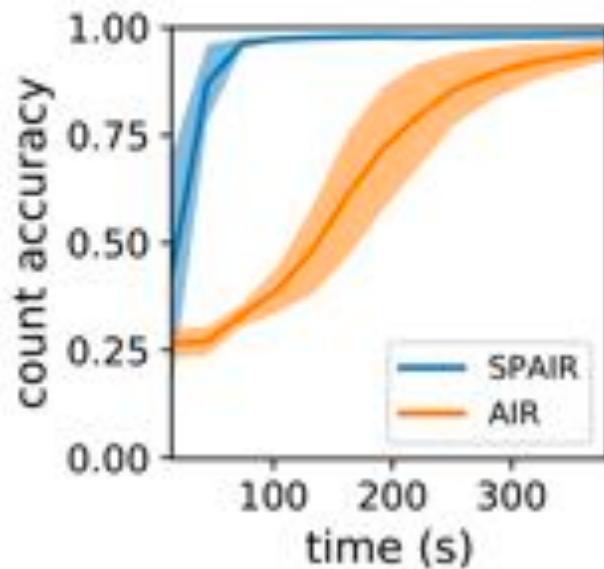
UNIVERSITY OF
CAMBRIDGE



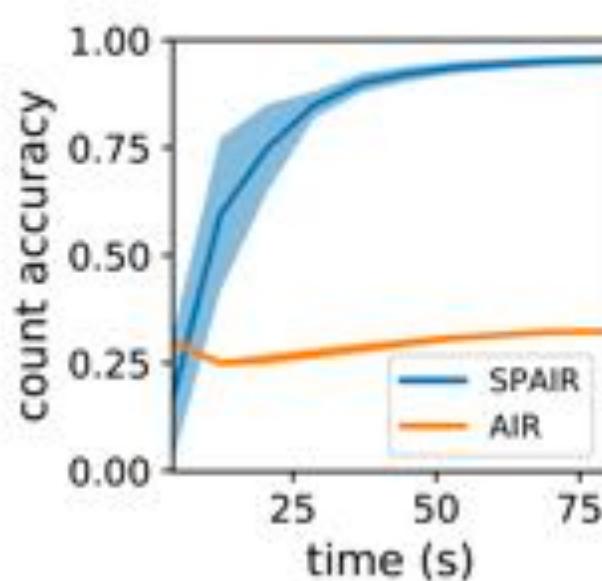
TECHNISCHE
UNIVERSITÄT
DARMSTADT

Sum-Product Attent-Infer Repeat

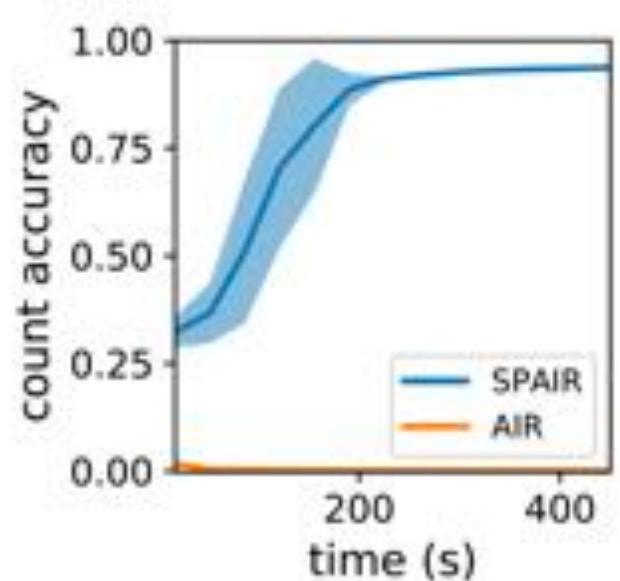
Multi-MNIST



Sprites



Noisy MNIST



[Stelzner, Peharz, Kersting ICML 2019]

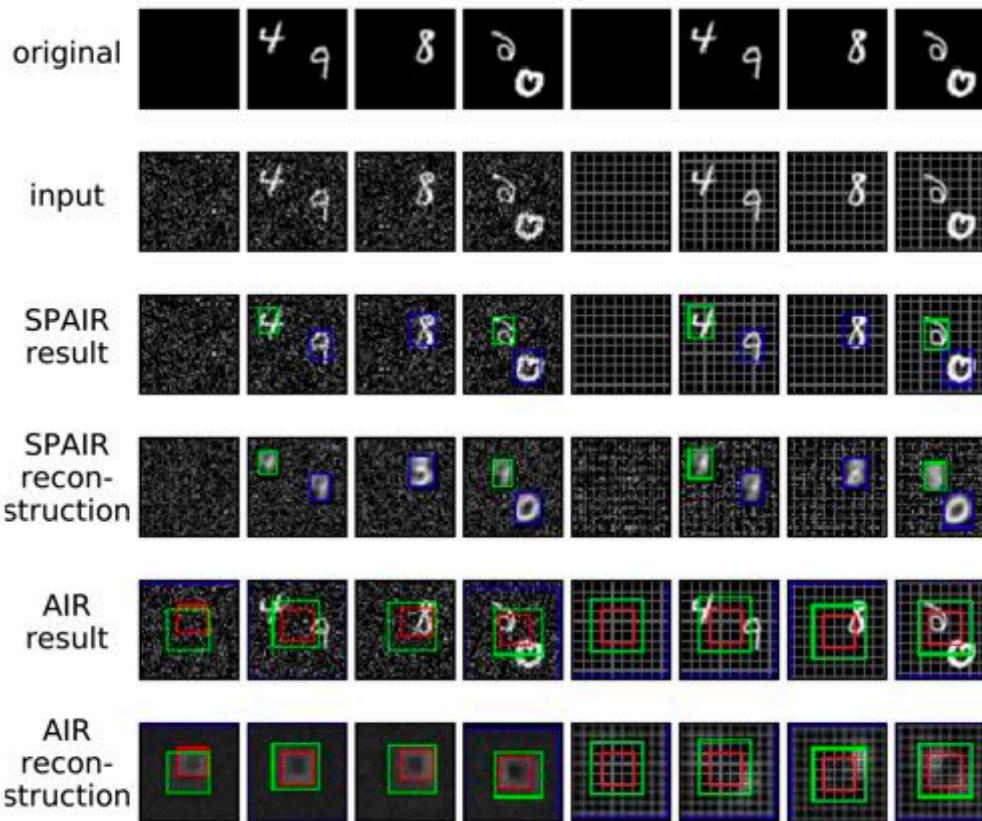


UNIVERSITY OF
CAMBRIDGE



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Sum-Product Attent-Infer Repeat



[Stelzner, Peharz, Kersting ICML 2019]



There are strong investments into (deep) probabilistic programming



RelationalAI, Apple, Microsoft and Uber are investing hundreds of millions of US dollars



Since we need languages for Systems AI,

the computational and mathematical modeling of complex AI systems.

[Laue et al. NeurIPS 2018; Kordjamshidi, Roth, Kersting:
“Systems AI: A Declarative Learning Based Programming
Perspective.” IJCAI-ECAI 2018]



Eric Schmidt, Executive Chairman, Alphabet Inc.: Just Say "Yes", Stanford Graduate School of Business, May 2, 2017. <https://www.youtube.com/watch?v=vbb-AjiXyh0>.

Overall, AI/ML/DS indeed refine “formal” science, but ...

- **AI is more than deep neural networks.** Probabilistic and causal models are whiteboxes that provide insights into applications
- **AI is more than a single table.** Loops, graphs, different data types, relational DBs, ... are central to data science and high-level programming languages for DS help to capture this complexity
- **AI is more than just Machine Learners and Statisticians**

Learning-based programming offers a framework for building systems that help to go beyond, democratize, and even automate traditional AI/ML/DS

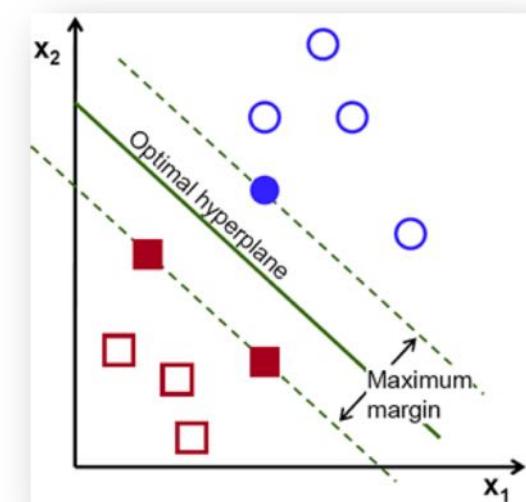
Not every Data Science machine is generative

$$\min_{\mathbf{w}, b, \xi} \mathcal{P}(\mathbf{w}, b, \xi) = \frac{1}{2} \mathbf{w}^2 + C \sum_{i=1}^n \xi_i$$

subject to $\begin{cases} \forall i \quad y_i(\mathbf{w}^\top \Phi(\mathbf{x}_i) + b) \geq 1 - \xi_i \\ \forall i \quad \xi_i \geq 0 \end{cases}$

Not everyone likes to turn math into code

Support Vector Machines
Cortes, Vapnik MLJ 20(3):273-297, 1995



High-level Languages for Mathematical Programs

Write down SVM in „paper form.“ The machine compiles it into solver form.

```
#QUADRATIC OBJECTIVE
minimize: sum{J in feature(I,J)} weight(J)**2 + c1 * slack + c2 * cosslack;

#labeled examples should be on the correct side
subject to forall {I in labeled(I)}: labeled(I)*predict(I) >= 1 - slack(I);

#slacks are positive
subject to forall {I in labeled(I)}: slack(I) >= 0;
```

Embedded within
Python s.t. loops and
rules can be used

reloop

RELOOP: A Toolkit for Relational Convex Optimization

Support Vector Machines
Cortes, Vapnik MLJ 20(3):273-297, 1995

