UCLA Computer Science

STAR AI RESEARCH LAB UCLA

From Probabilistic Circuits to Probabilistic Programs and Back

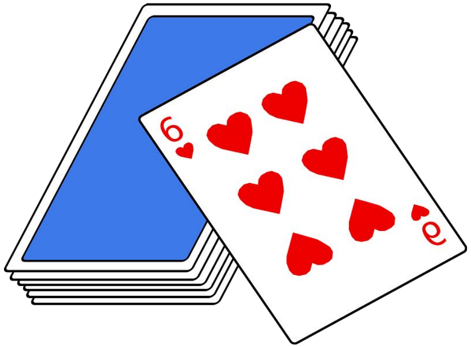
Guy Van den Broeck

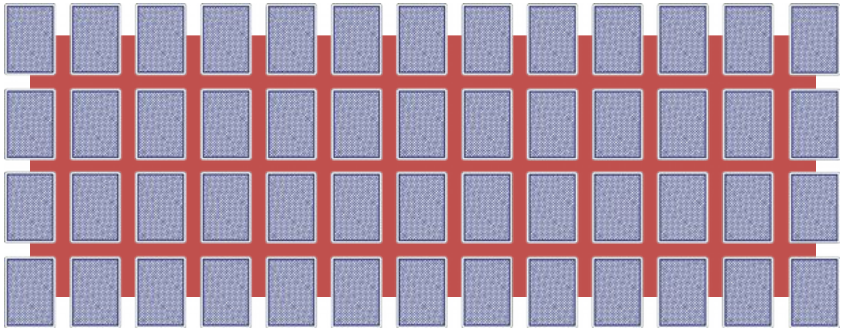
Los Alamos National Laboratory - Mar 16, 2021

Trying to be provocative

Probabilistic graphical models is how we do probabilistic AI!

Graphical models of variable-level (in)dependence
are a broken abstraction.





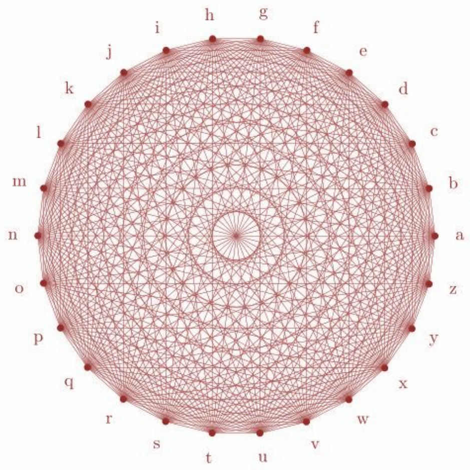
[VdB KRR15]

Trying to be provocative

Probabilistic graphical models is how we do probabilistic AI!

Graphical models of variable-level (in)dependence are a broken abstraction.

3.14 Smokes(x) ∧ Friends(x,y) ⇒ Smokes(y)



[VdB KRR15]

Trying to be provocative

Probabilistic graphical models is how we do probabilistic AI!

Graphical models of variable-level (in)dependence are a broken abstraction.

Bean Machine

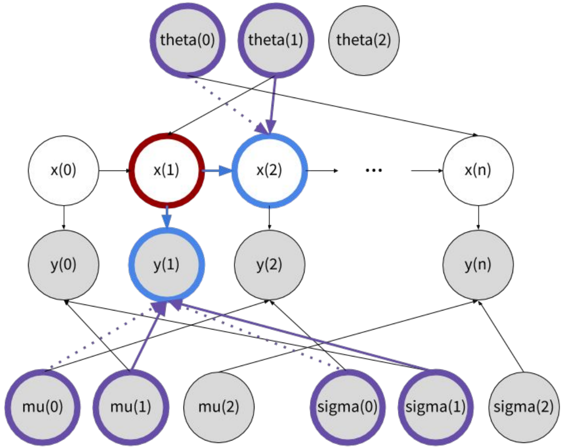
$$\mu_k \sim \mathrm{Normal}(\alpha, \beta)$$

$$\sigma_k \sim \mathrm{Gamma}(\nu, \rho)$$

$$\theta_k \sim \mathrm{Dirichlet}(\kappa)$$

$$x_i \sim \begin{cases} \mathrm{Categorical}(init) & \text{if } i = 0 \\ \mathrm{Categorical}(\theta_{x_{i-1}}) & \text{if } i > 0 \end{cases}$$

$$y_i \sim \mathrm{Normal}(\mu_{x_i}, \sigma_{x_i})$$



[Tehrani et al. PGM20]

Computational Abstractions

Let us think of probability distributions as objects that are computed.

Abstraction = Structure of Computation

Two examples:

1. Probabilistic Circuits
2. Probabilistic Programs

Probabilistic Circuits

Fully factorized

NaiveBayes AndOrGraphs PDGs

Trees PSDDs CNets LTMs SPNs NADEs

Thin Junction Trees ACs MADEs MAFs VAEs

DPPs FVSBNs TACs IAFs NAFs RAEs

Mixtures BNs NICE FGs GANs

RealNVP MNs

The Alphabet Soup of probabilistic models

Fully factorized

NaiveBayes AndOrGraphs PDGs

Trees PSDDs CNets LTMs SPNs NADEs

Thin Junction Trees ACs MADEs MAFs VAEs

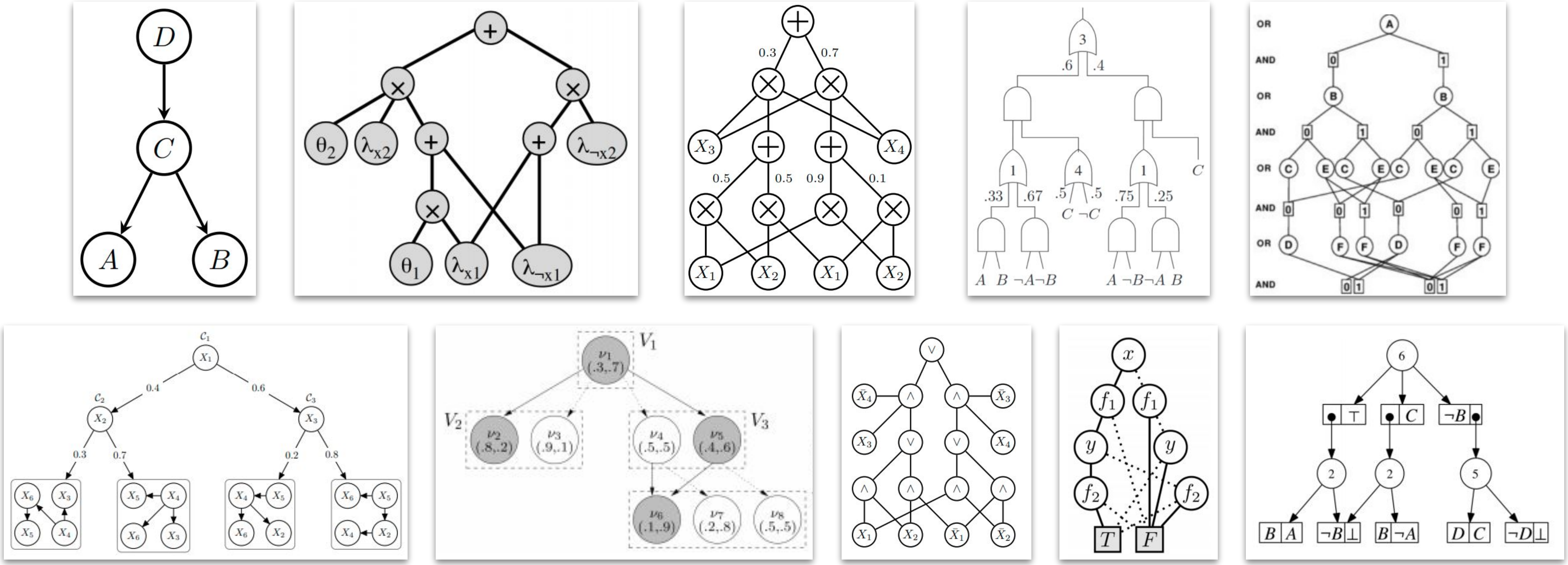
DPPs FVSBNs TACs IAFs NAFs RAEs

Mixtures BNs NICE FGs GANs

RealNVP MNs

Intractable and *tractable* models

Tractable Probabilistic Models



"Every talk needs a joke and a literature overview slide, not necessarily distinct"

- after Ron Graham

Fully factorized

NaiveBayes *AndOrGraphs* *PDGs*

Trees *PSDDs* *CNets* *LTMs* *SPNs* *NADEs*

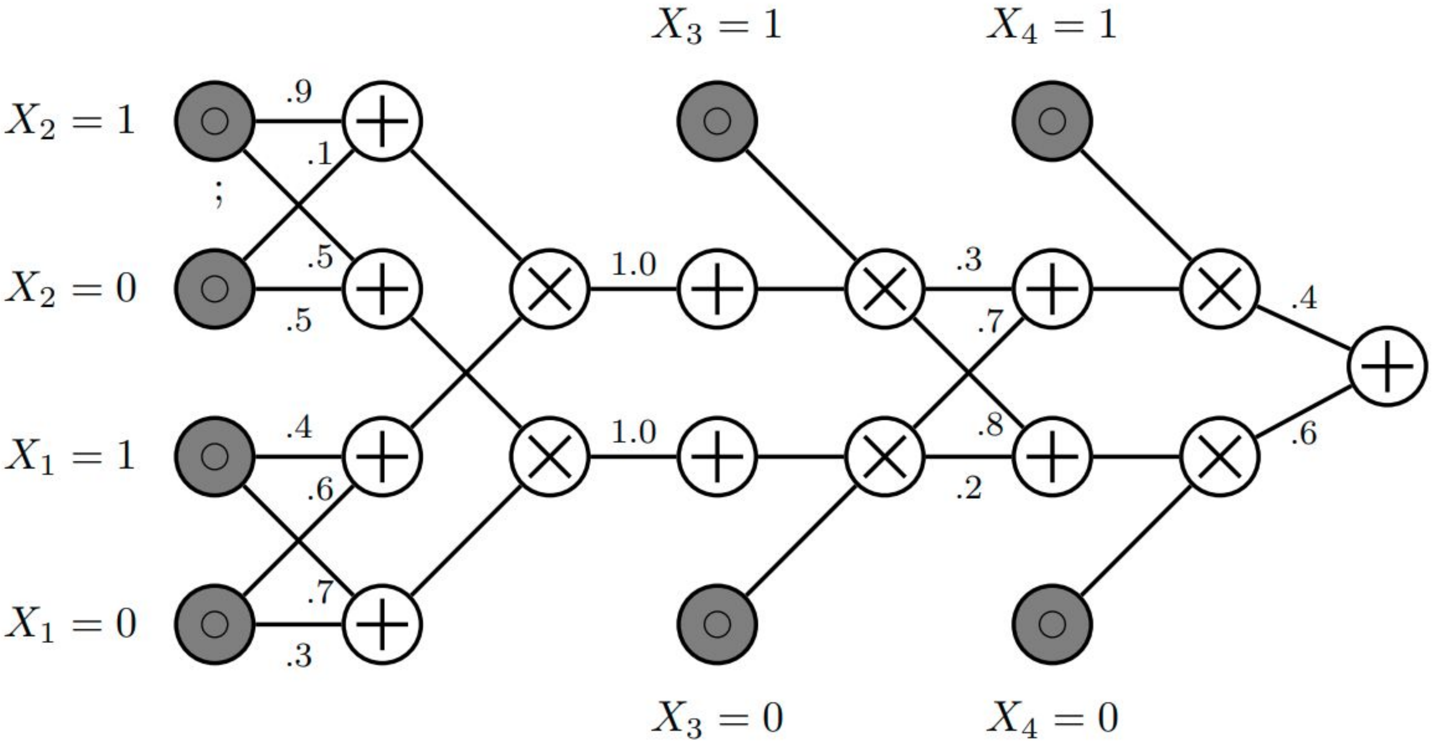
Thin Junction Trees *ACs* *MADEs* *MAFs* *VAEs*

DPPs *FVSBNs* *TACs* *IAFs* *NAFs* *RAEs*

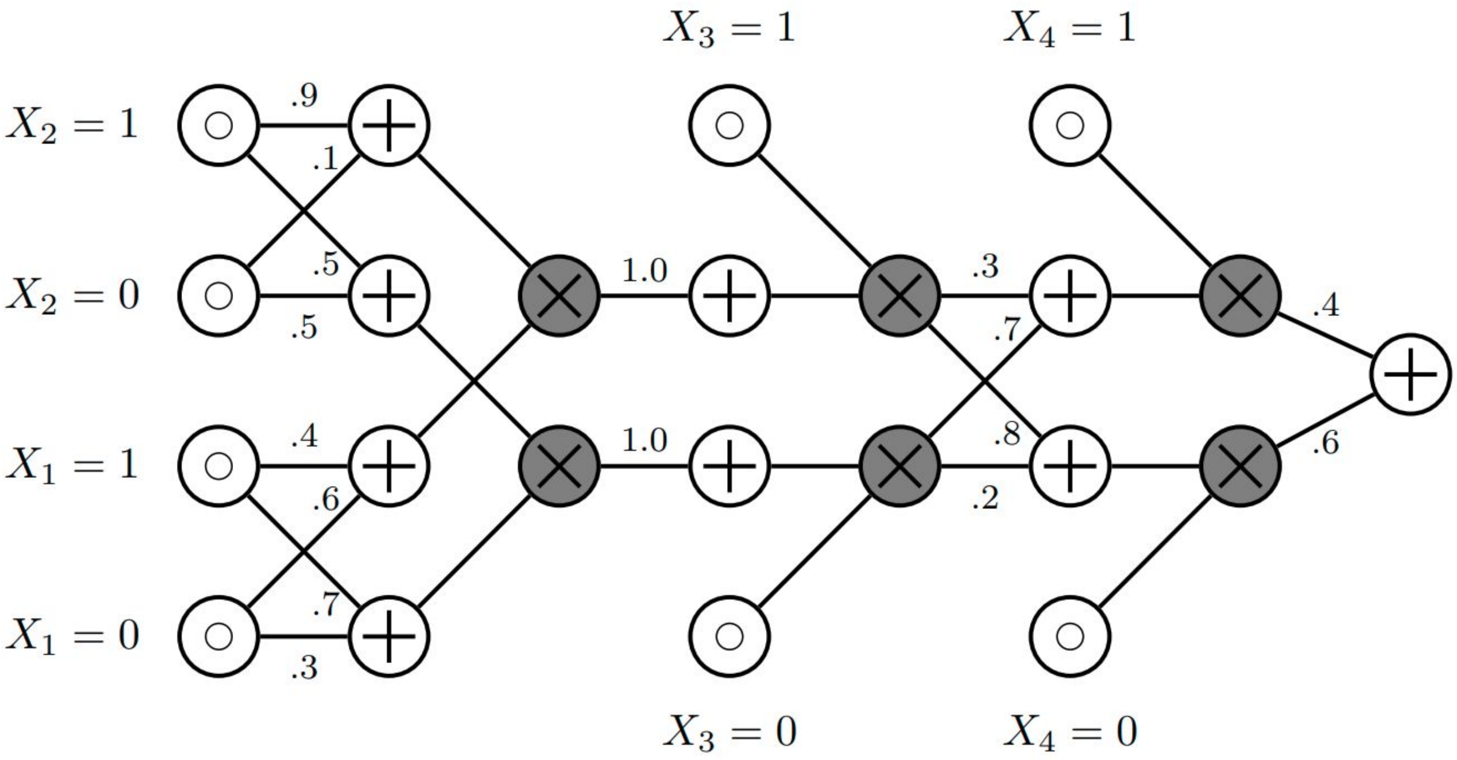
Mixtures *BNs* *NICE* *FGs* *GANs*

RealNVP *MNs*

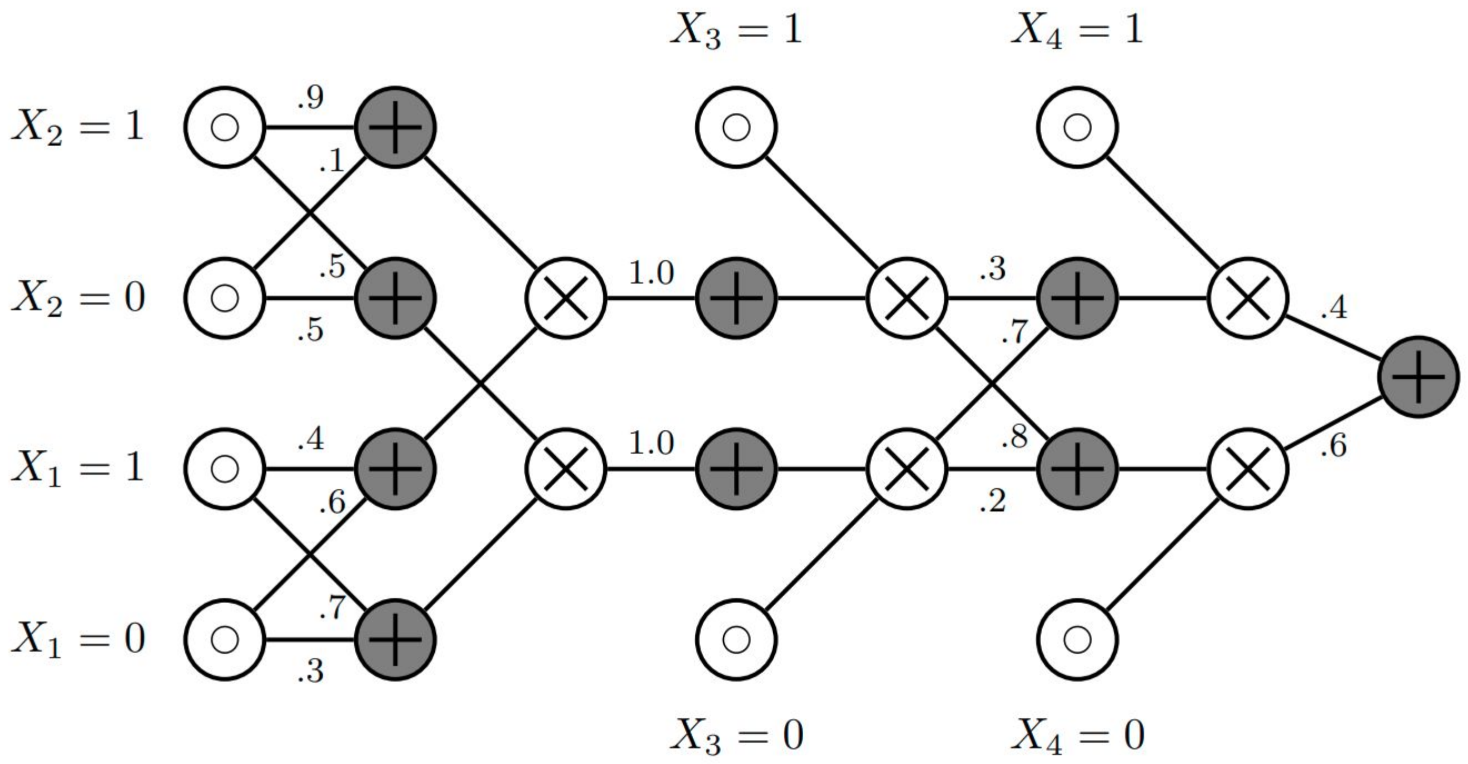
a *unifying framework* for tractable models



Input nodes are tractable (simple) distributions,
e.g., univariate gaussian or indicator p(X=1) = [X=1]

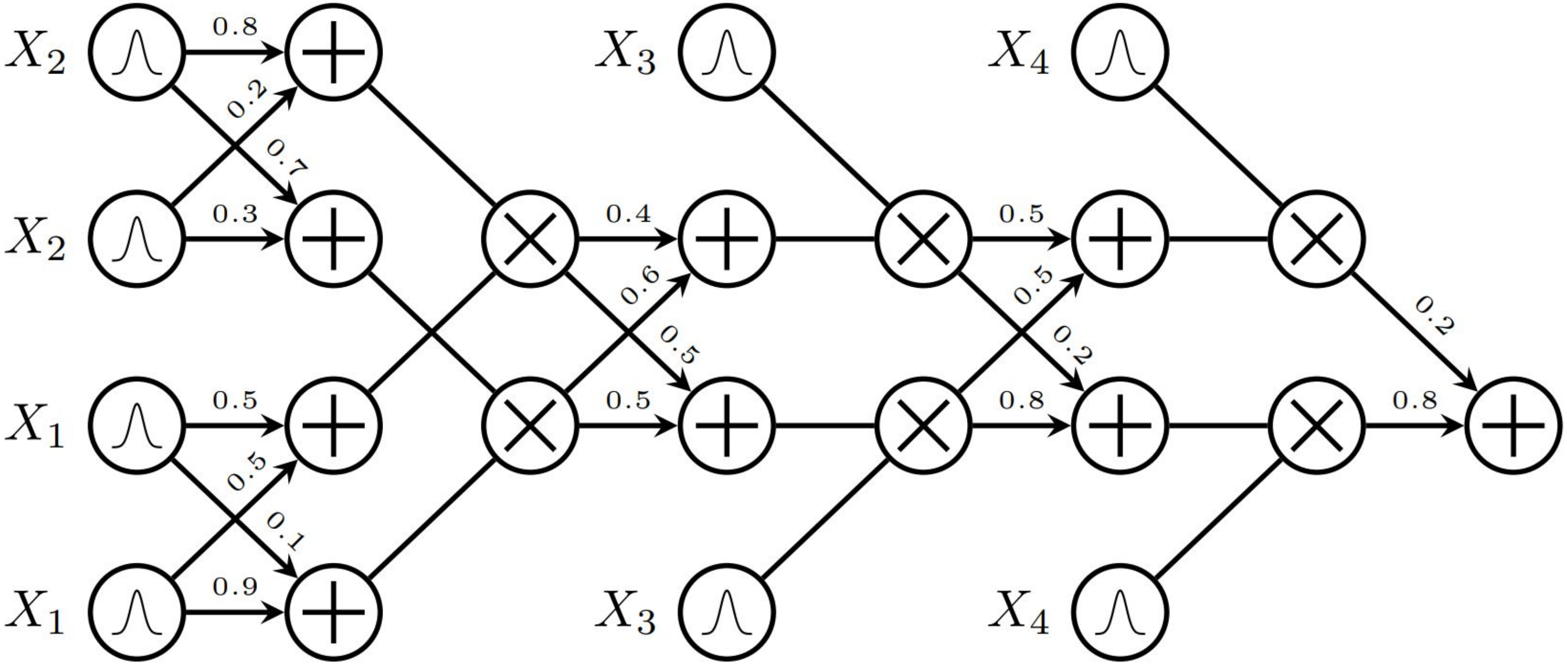


Product nodes are factorizations $\prod_{c \in \mathsf{in}(n)} \mathrm{p}_c(\mathbf{x})$

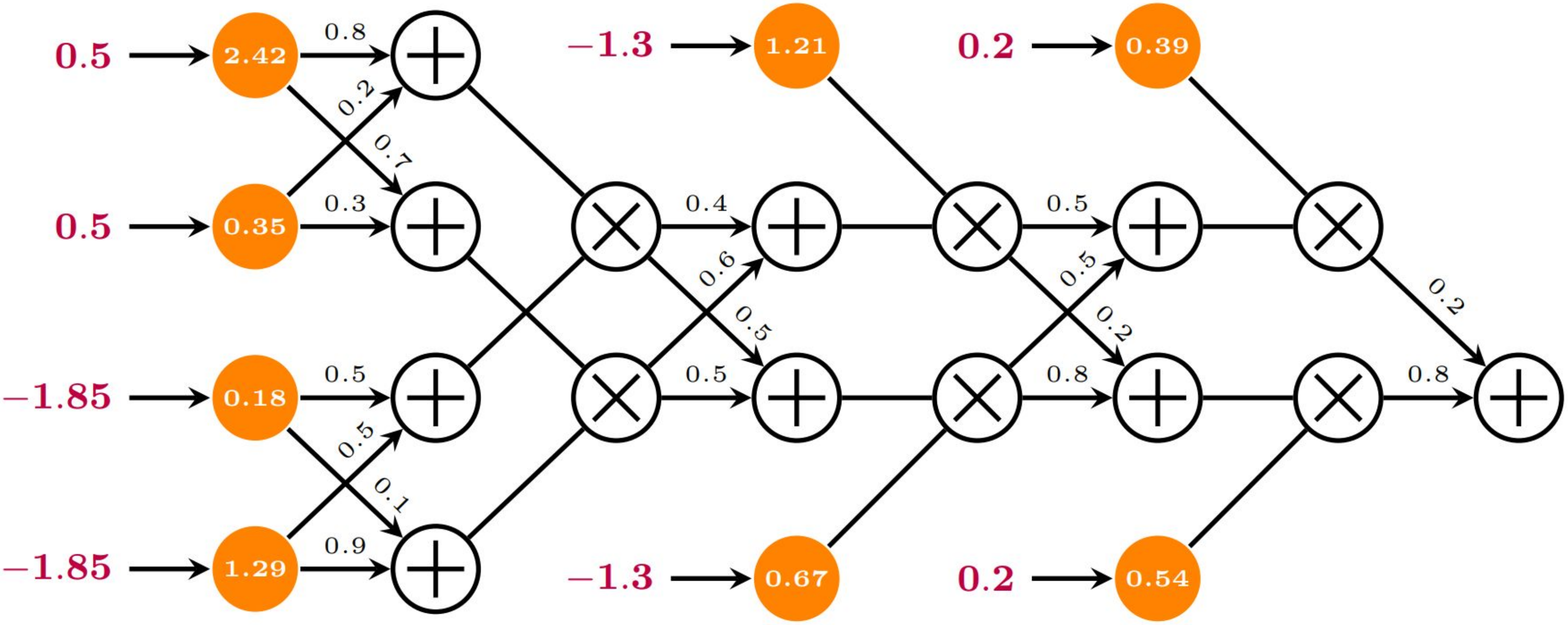


Sum nodes are mixture models $\sum_{c \in \mathsf{in}(n)} \theta_{n,c}\, \mathrm{p}_c(\mathbf{x})$

Feedforward $p(X_1 = \mathbf{-1.85}, X_2 = \mathbf{0.5}, X_3 = \mathbf{-1.3}, X_4 = \mathbf{0.2})$

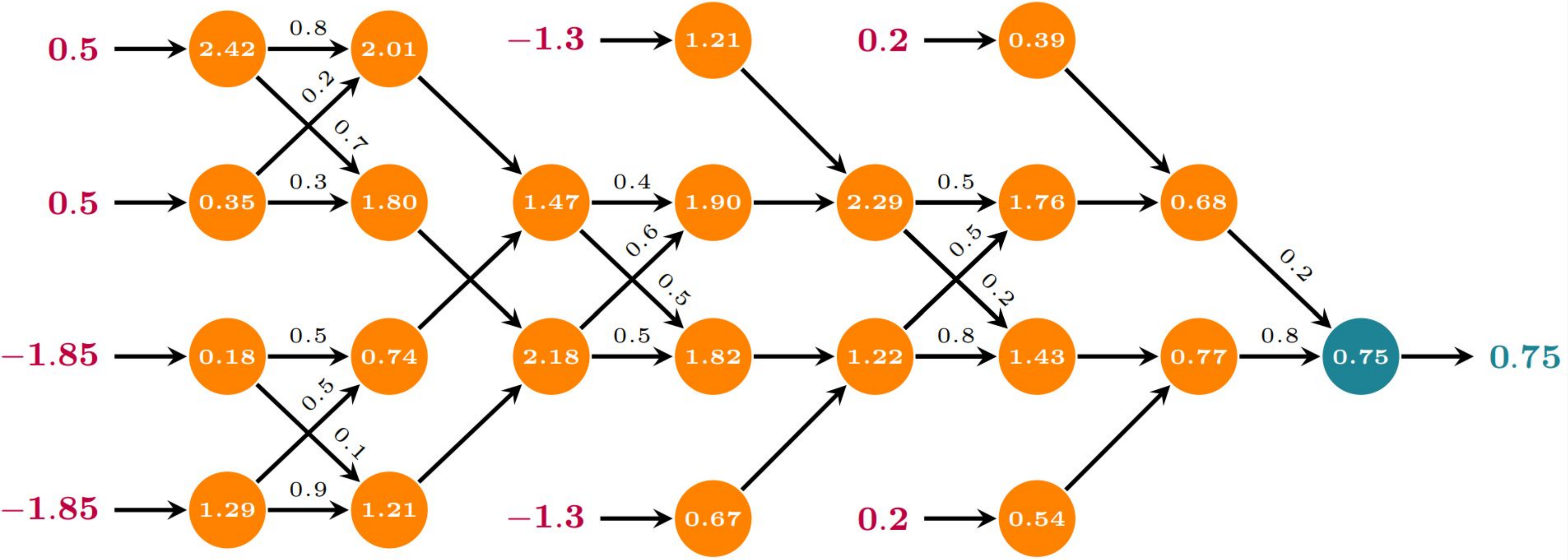


Feedforward $p(X_1 = \mathbf{-1.85}, X_2 = \mathbf{0.5}, X_3 = \mathbf{-1.3}, X_4 = \mathbf{0.2})$



Feedforward

$$p(X_1 = \mathbf{-1.85}, X_2 = \mathbf{0.5}, X_3 = \mathbf{-1.3}, X_4 = \mathbf{0.2})$$



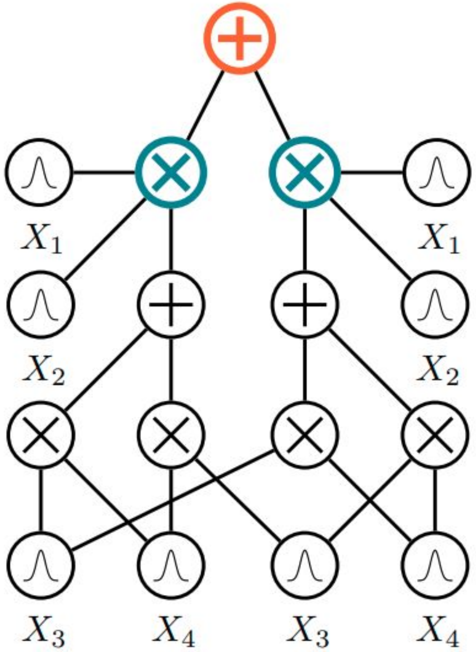
Smoothness + decomposability = tractable MAR

If $p(\mathbf{x}) = \sum_i w_i p_i(\mathbf{x})$, (***smoothness***):

$$\int p(\mathbf{x})d\mathbf{x} = \int \sum_i w_i p_i(\mathbf{x})d\mathbf{x} =$$

$$= \sum_i w_i \int p_i(\mathbf{x})d\mathbf{x}$$

$\Rightarrow$ *integrals are "pushed down" to children*



[Darwiche & Marquis JAIR 2001, Poon & Domingos UAI11]

Smoothness + decomposability = tractable MAR

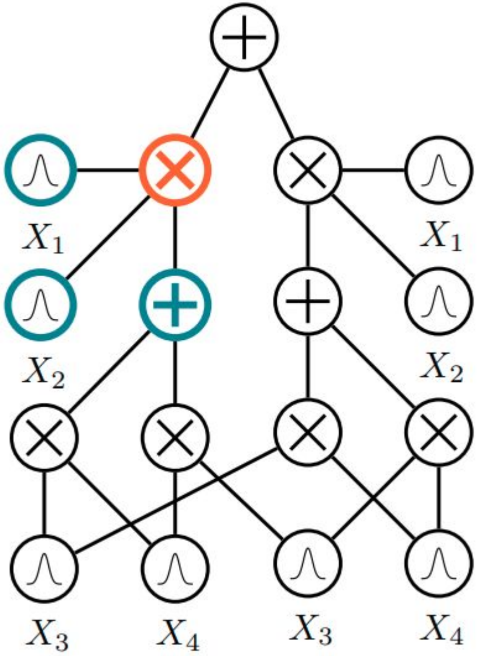
If $p(\mathbf{x}, \mathbf{y}, \mathbf{z}) = p(\mathbf{x})p(\mathbf{y})p(\mathbf{z})$, (***decomposability***):

$$\int\int\int p(\mathbf{x}, \mathbf{y}, \mathbf{z})d\mathbf{x}d\mathbf{y}d\mathbf{z} =$$

$$= \int\int\int p(\mathbf{x})p(\mathbf{y})p(\mathbf{z})d\mathbf{x}d\mathbf{y}d\mathbf{z} =$$

$$= \int p(\mathbf{x})d\mathbf{x} \int p(\mathbf{y})d\mathbf{y} \int p(\mathbf{z})d\mathbf{z}$$

$\Rightarrow$ *integrals decompose into easier ones*



Smoothness + decomposability = tractable MAR

Forward pass evaluation for MAR

$\Rightarrow$ *linear in circuit size!*

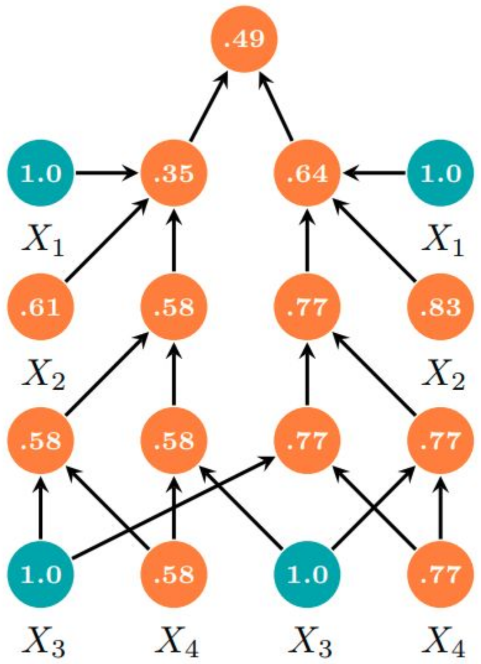
E.g. to compute $p(x_2, x_4)$:

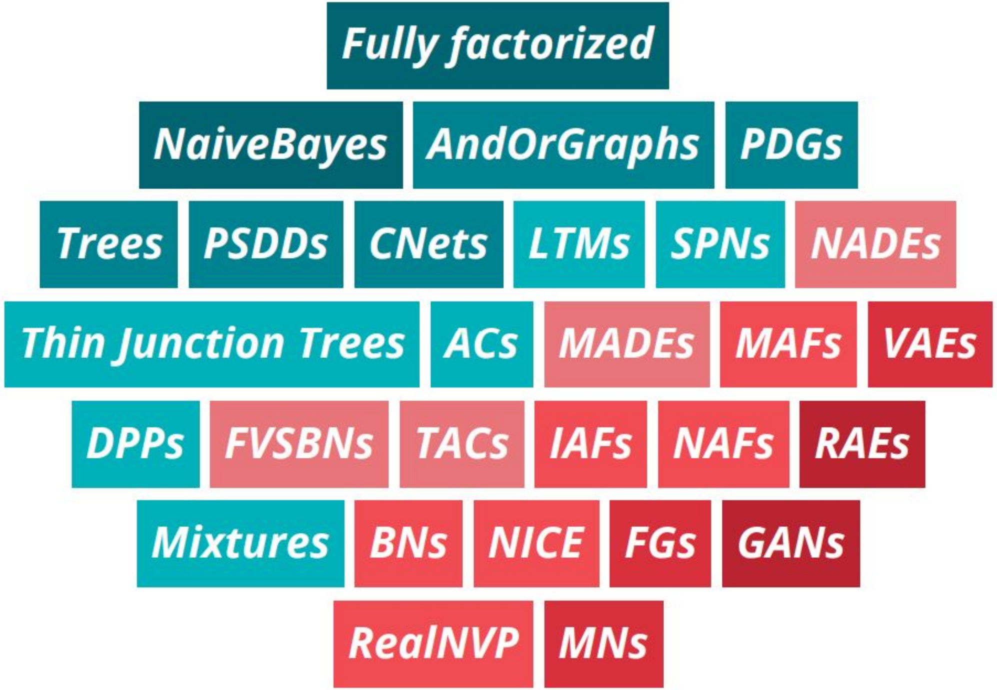
- leafs over X_1 and X_3 output $\boldsymbol{Z}_i = \int p(x_i)dx_i$

  $\Rightarrow$ *for normalized leaf distributions:* 1.0

- leafs over X_2 and X_4 output **EVI**

- feedforward evaluation (bottom-up)





tractability is a spectrum

$\mathcal{P}$ \ $\mathcal{Q}$:	MAR marginal queries	CON conditional queries	MOM moments (mean,...)	MAP maximum a posteriori	MMAP marginal MAP	ENT entropy	DIV divergences (KLD,...)	EXP expected predictions
smoothness SMO	✓	✓	✓	✗	✗	✓	✓	✓
decomposability DEC	✓	✓	✓	✗	✗	✗	✗	✗
consistency CON	✗	✗	✗	✓	✓	✗	✗	✗
determinism DET	✗	✗	✗	✓	✗	✗	✗	✗
marginal determinism MAR-DET	✗	✗	✗	✗	✓	✓	✓	✗
structured decomposability STR-DEC	✗	✗	✗	✗	✗	✓	✗	✗
paired str. decomposability P-STR-DEC	✗	✗	✗	✗	✗	✗	✓	✓

Fully factorized

NaiveBayes AndOrGraphs PDGs

Trees PSDDs CNets LTMs SPNs NADEs

Thin Junction Trees ACs MADEs MAFs VAEs

DPPs FVSBNs TACs IAFs NAFs RAEs

Mixtures BNs NICE FGs GANs

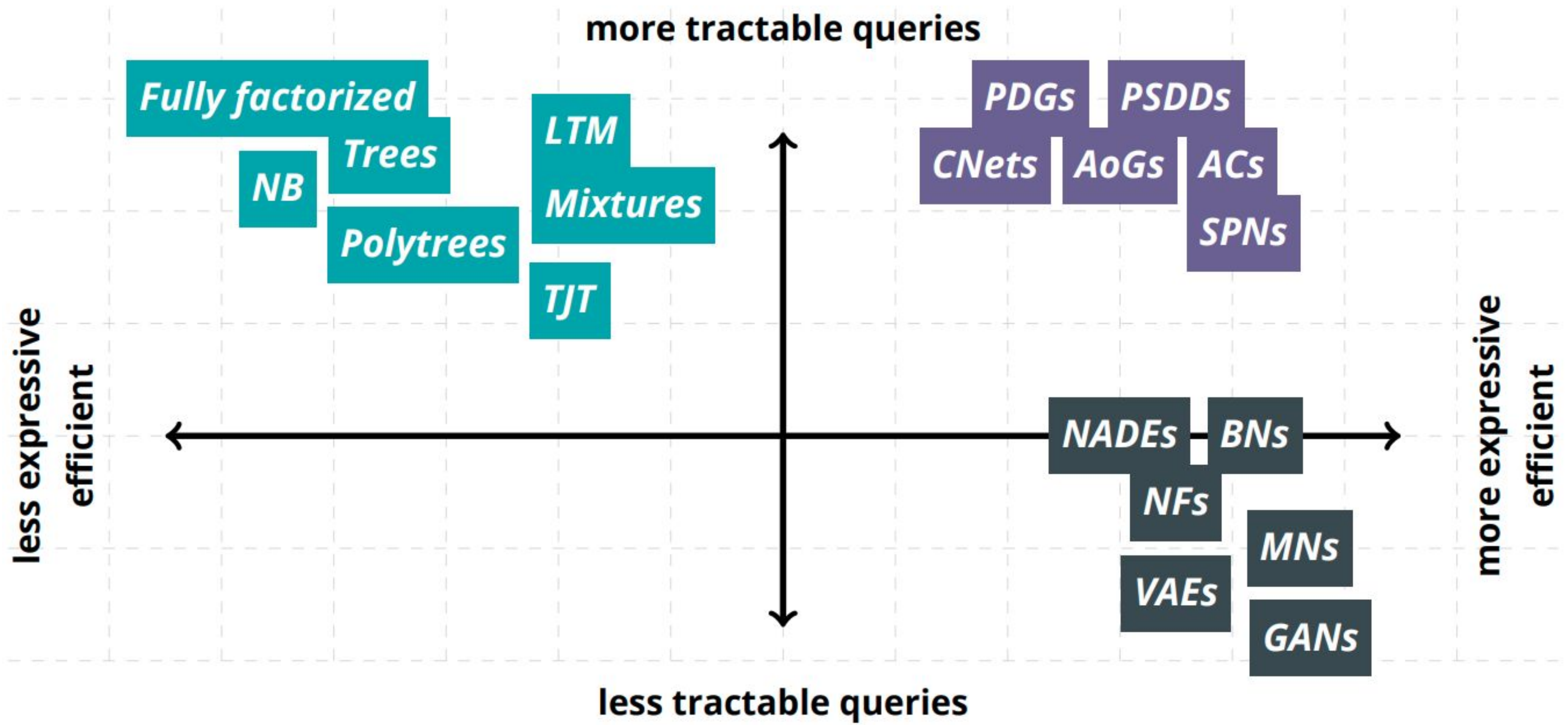
RealNVP MNs

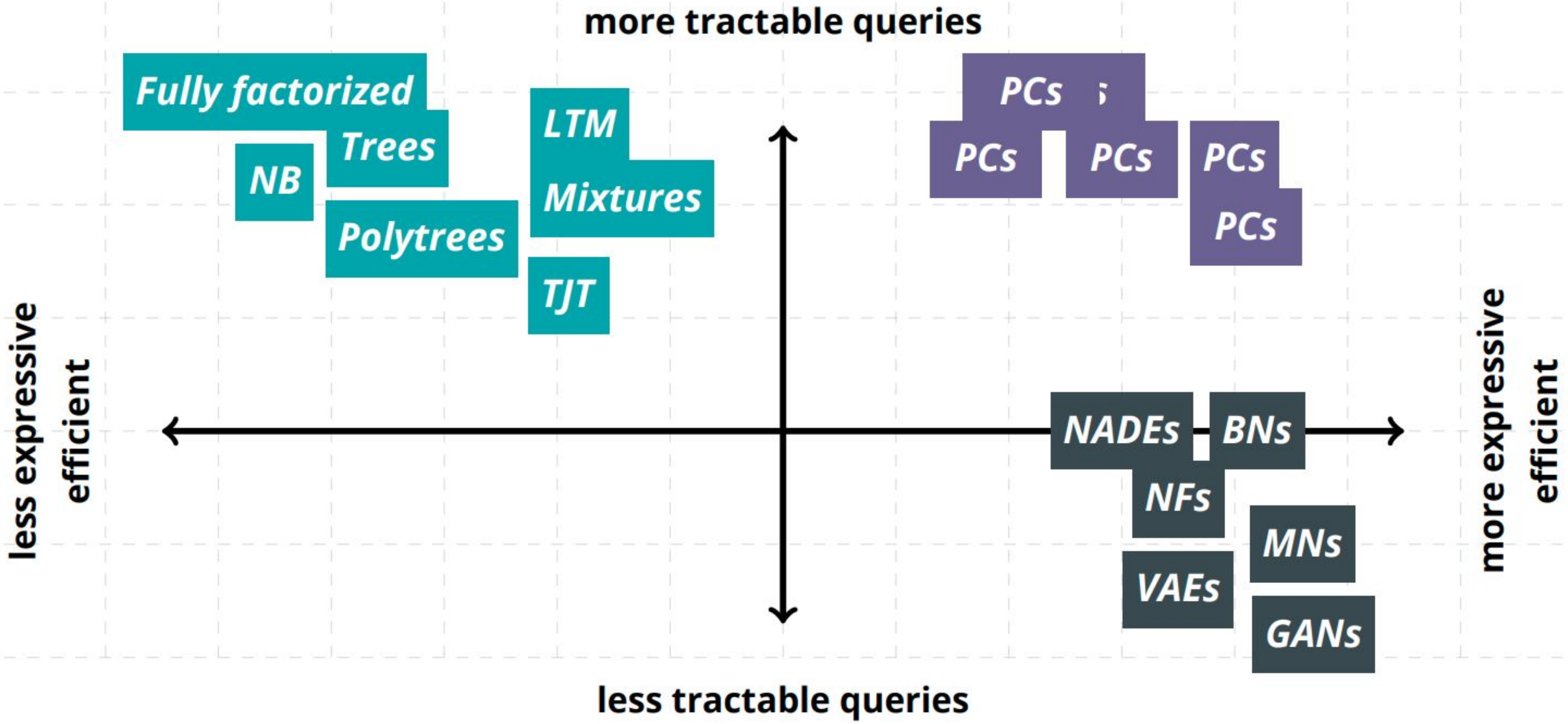
Expressive models without *compromises*

How expressive are probabilistic circuits?

density estimation benchmarks

dataset	best circuit	BN	MADE	VAE	dataset	best circuit	BN	MADE	VAE
nltcs	**-5.99**	-6.02	-6.04	**-5.99**	***dna***	**-79.88**	-80.65	-82.77	-94.56
msnbc	**-6.04**	**-6.04**	-6.06	-6.09	***kosarek***	**-10.52**	-10.83	-	-10.64
kdd	-2.12	-2.19	**-2.07**	-2.12	***msweb***	-9.62	-9.70	**-9.59**	-9.73
plants	**-11.84**	-12.65	-12.32	-12.34	***book***	-33.82	-36.41	-33.95	**-33.19**
audio	-39.39	-40.50	-38.95	**-38.67**	***movie***	-50.34	-54.37	-48.7	**-47.43**
jester	-51.29	**-51.07**	-52.23	-51.54	***webkb***	-149.20	-157.43	-149.59	**-146.9**
netflix	-55.71	-57.02	-55.16	**-54.73**	***cr52***	-81.87	-87.56	-82.80	**-81.33**
accidents	-26.89	**-26.32**	-26.42	-29.11	***c20ng***	-151.02	-158.95	-153.18	**-146.9**
retail	**-10.72**	-10.87	-10.81	-10.83	***bbc***	**-229.21**	-257.86	-242.40	-240.94
pumbs*	-22.15	**-21.72**	-22.3	-25.16	***ad***	-14.00	-18.35	**-13.65**	-18.81


more tractable queries
Fully factorized
Trees
NB
Polytrees
LTM
Mixtures
TJT
PDGs
PSDDs
CNets
AoGs
ACs
SPNs
less expressive
efficient
more expressive
efficient
NADEs
BNs
NFs
MNs
VAEs
GANs
less tractable queries



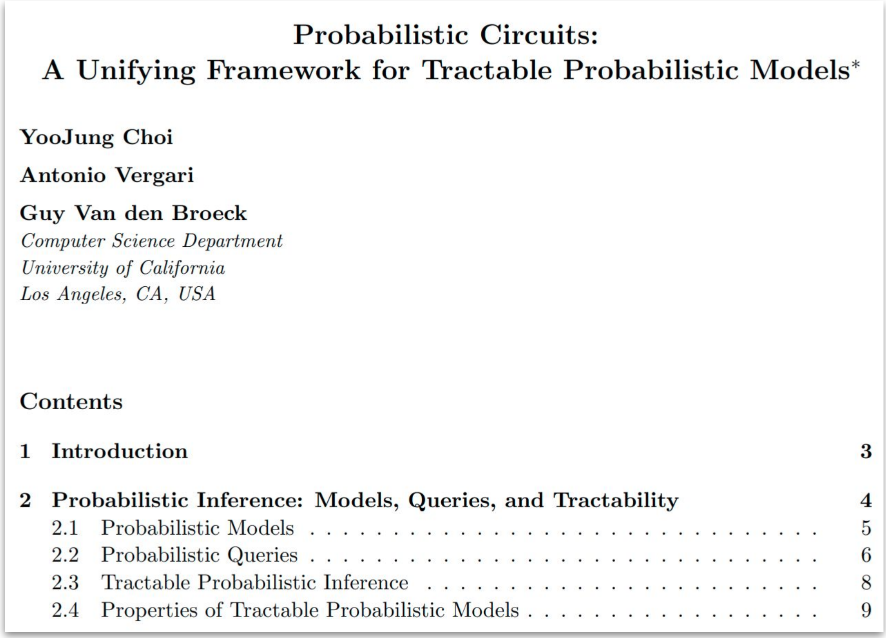
more tractable queries
Fully factorized
Trees
NB
Polytrees
LTM
Mixtures
TJT
PCs
PCs
PCs
PCs
PCs
less expressive
efficient
more expressive
efficient
NADEs
BNs
NFs
MNs
VAEs
GANs
less tractable queries

Want to learn more?

Tutorial (3h)

https://youtu.be/2RAG5-L9R70

Overview Paper (80p)



Probabilistic Circuits:
A Unifying Framework for Tractable Probabilistic Models*

YooJung Choi
Antonio Vergari
Guy Van den Broeck
Computer Science Department
University of California
Los Angeles, CA, USA

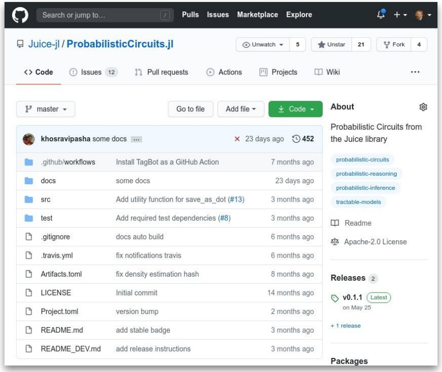
Contents

http://starai.cs.ucla.edu/papers/ProbCirc20.pdf

Training PCs in Julia with Juice.jl

Training maximum likelihood parameters of probabilistic circuits

```
julia> using ProbabilisticCircuits;
julia> data, structure = load(...);
julia> num_examples(data)
17,412
julia> num_edges(structure)
270,448
julia> @btime estimate_parameters(structure , data);
63 ms
```



Custom SIMD and CUDA kernels to parallelize over layers and training examples.

[https://github.com/Juice-jl/]

Probabilistic circuits seem awfully general.

Are all tractable probabilistic models probabilistic circuits?

Enter: Determinantal Point Processes (DPPs)

DPPs are models where probabilities are specified by (sub)determinants

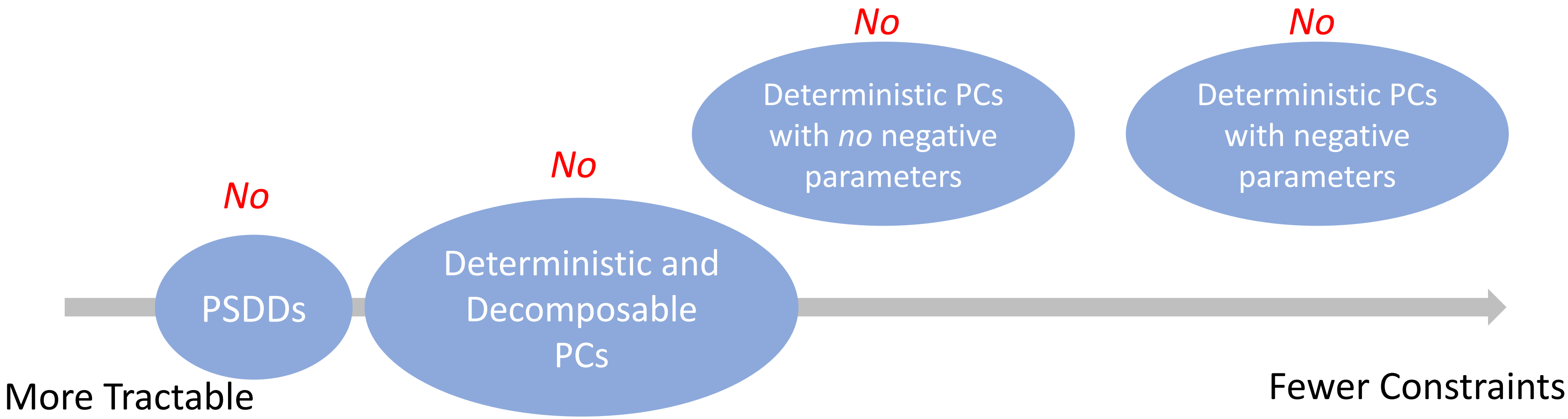
$$L = \begin{bmatrix} 1 & 0.9 & 0.8 & 0 \\ 0.9 & 0.97 & 0.96 & 0 \\ 0.8 & 0.96 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

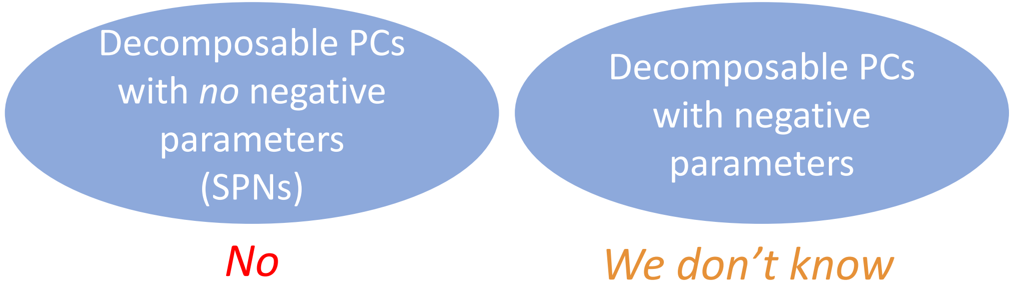
$$\Pr_L(X_1 = 1, X_2 = 0, X_3 = 1, X_4 = 0) = \frac{1}{\det(L+I)} \det(L_{\{1,2\}})$$

Computing marginal probabilities is *tractable.*

[Zhang et al. UAI20]

Foundational Question: Can PCs represent DPPs efficiently?





An *almost* universal tractable language…

Solution: **Probabilistic Generating Circuits**

[Zhang et al. Arxiv 21, Zhang et al. UAI20; Martens & Medabalimi Arxiv15]

The AI Dilemma

Pure Logic **Pure Learning**

The AI Dilemma

Pure Logic

Pure Learning

- Slow thinking: deliberative, cognitive, model-based, extrapolation
- Amazing achievements until this day
- "*Pure logic is brittle*"
  noise, uncertainty, incomplete knowledge, …

The AI Dilemma

Pure Logic **Pure Learning**

- Fast thinking: instinctive, perceptive, model-free, interpolation
- Amazing achievements recently
- “*Pure learning is brittle*”

  bias, algorithmic fairness, interpretability, explainability, adversarial attacks, unknown unknowns, calibration, verification, missing features, missing labels, data efficiency, shift in distribution, general robustness and safety

  fails to incorporate a sensible model of the world

Pure Logic Probabilistic World Models Pure Learning

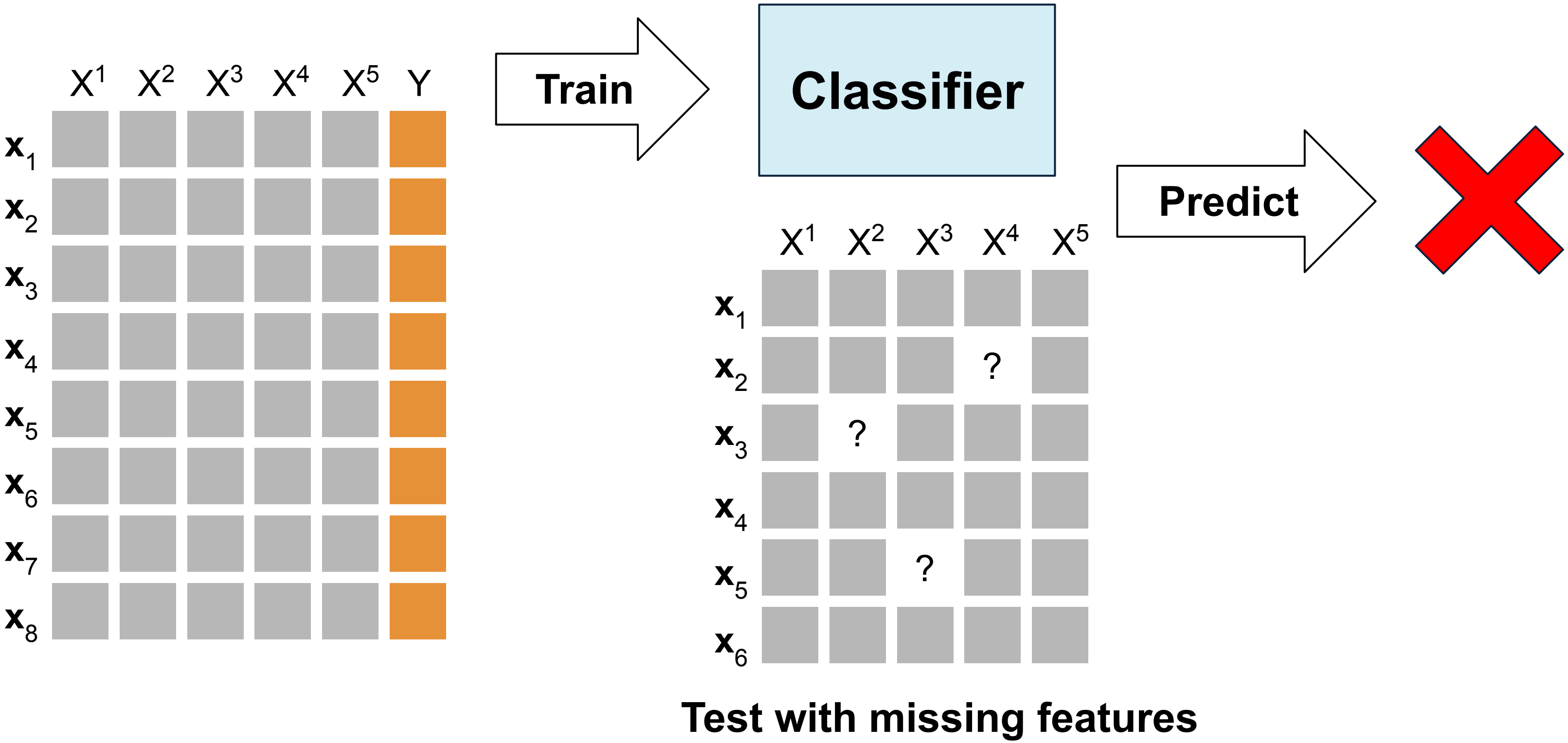
A New Synthesis of Learning and Reasoning

- "*Pure learning is brittle*"

bias, **algorithmic fairness**, interpretability, **explainability**, adversarial attacks, unknown unknowns, calibration, verification, **missing features**, missing labels, data efficiency, shift in distribution, general robustness and safety

fails to incorporate a sensible model of the world

Prediction with Missing Features



Expected Predictions

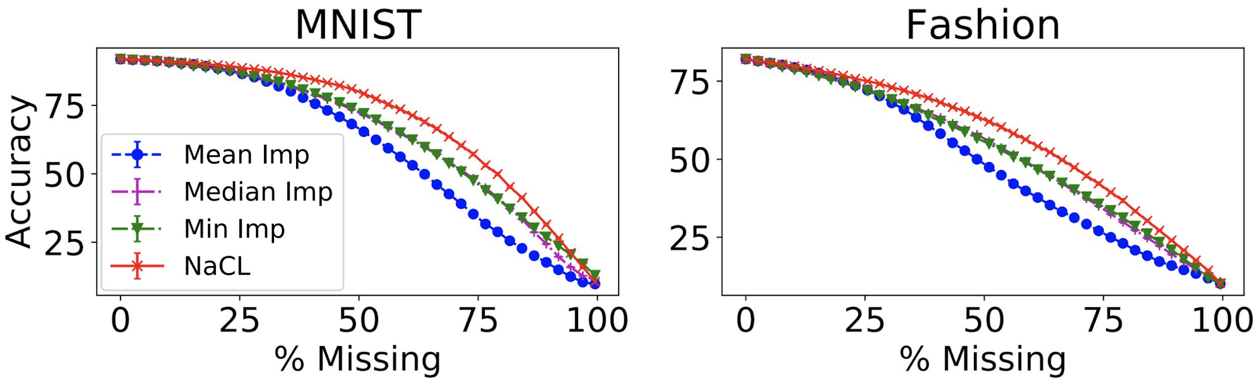
Consider **all possible complete inputs** and **reason** about the *expected* behavior of the classifier

$$\mathbb{E}_{\mathbf{x}^m \sim p(\mathbf{x}^m | \mathbf{x}^o)} \left[f(\mathbf{x}^m \mathbf{x}^o) \right]$$

$\boldsymbol{x}^{\mathrm{o}}$ = observed features
$\boldsymbol{x}^{\mathrm{m}}$ = missing features

Experiment:

- *f(x)* = logistic regres.
- *p(x)* = naive Bayes



[Khosravi et al. IJCAI19, NeurIPS20, Artemiss20]

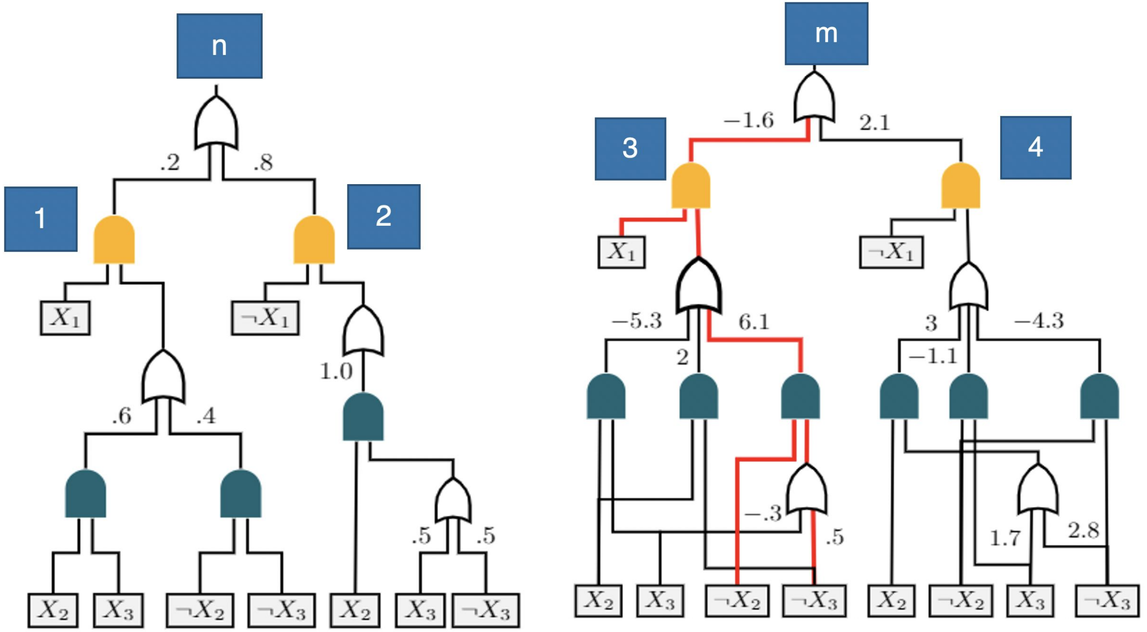
What about complex feature distributions?

- feature distribution is a probabilistic circuits
- classifier is a compatible regression circuit

Recursion that "breaks down" the computation.

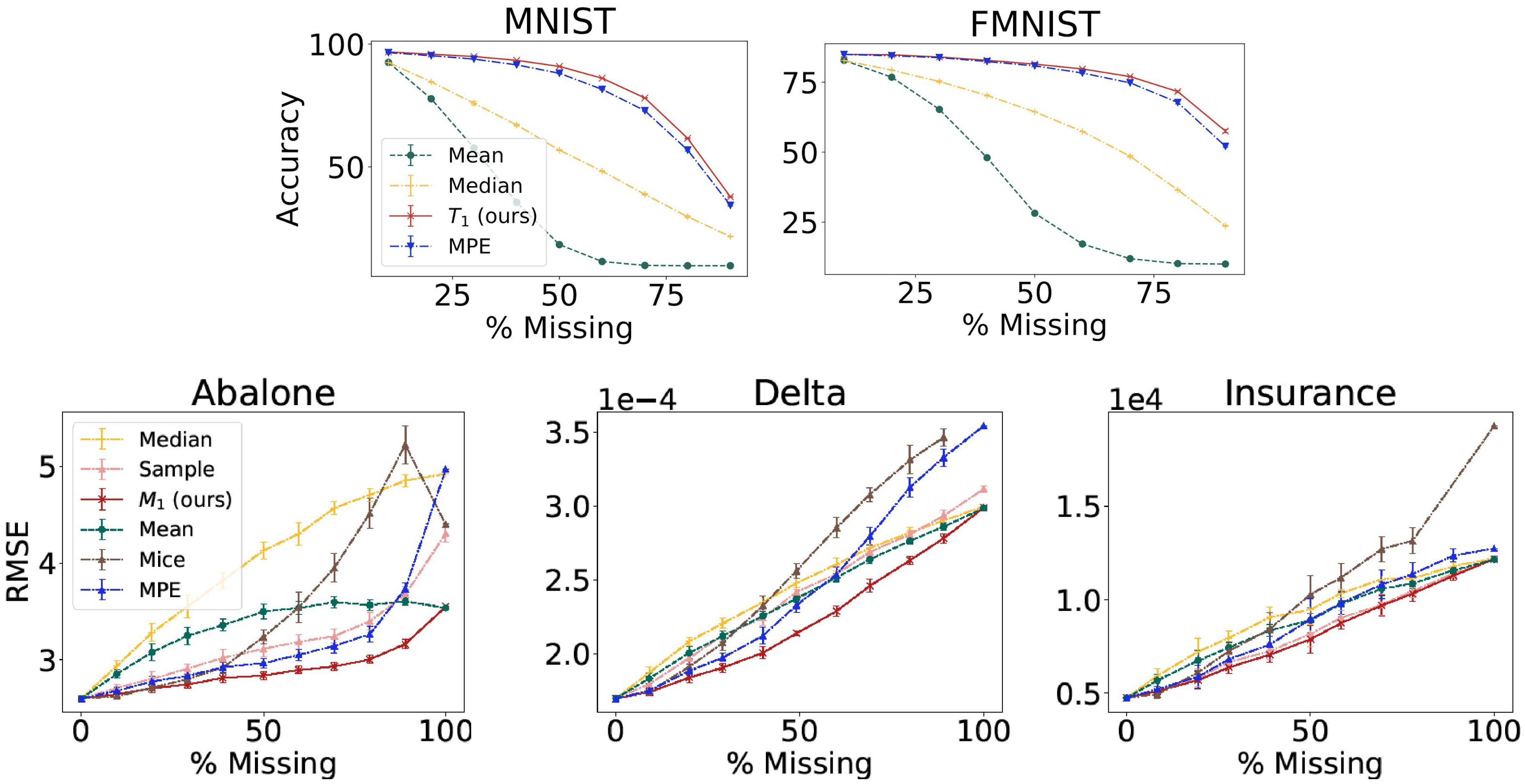
Expectation of function m w.r.t. dist. n ?

Solve subproblems: (1,3), (1,4), (2,3), (2,4)



[Khosravi et al. IJCAI19, NeurIPS20, Artemiss20]

Probabilistic Circuits for Missing Data



[Khosravi et al. IJCAI19, NeurIPS20, Artemiss20]

ADV inference in Julia with Juice.jl

```
using ProbabilisticCircuits
pc = load_prob_circuit(zoo_psdd_file("insurance.psdd"));
rc = load_logistic_circuit(zoo_lc_file("insurance.circuit"), 1);
```

Is the predictive model biased by gender?

```
groups = make_observations([["male"], ["female"]])
exps, _ = Expectation(pc, rc, groups);
println("Female  : \$ $(exps[2])");
println("Male    : \$ $(exps[1])");
println("Diff    : \$ $(exps[2] - exps[1])");
Female  : $ 14170.125469335406
Male    : $ 13196.548926381849
Diff    : $ 973.5765429535568
```

Model-Based Algorithmic Fairness: FairPC

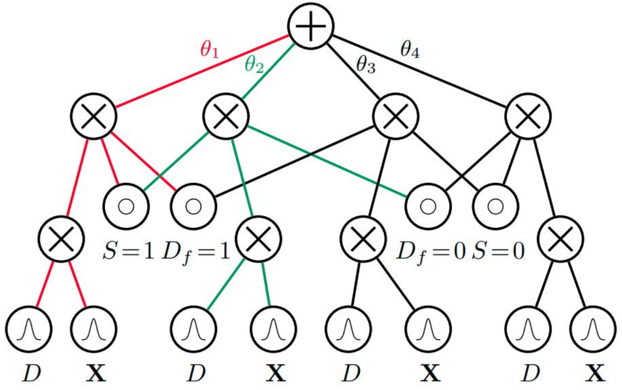
Learn classifier given

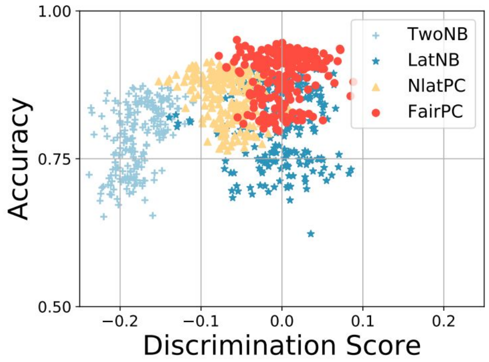
- features S and X
- training labels/decisions D

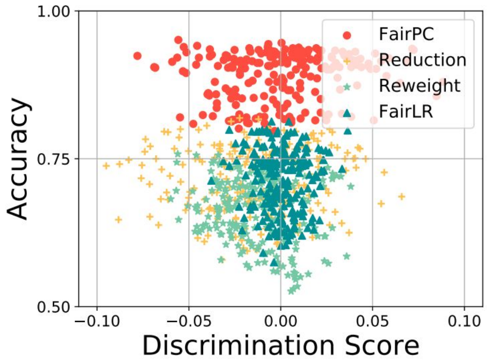
Group fairness by demographic parity:

Fair decision D_f should be independent of the sensitive attribute S

Discover the **latent fair decision** D_f by learning a PC.







[Choi et al. AAAI21]

Probabilistic Sufficient Explanations

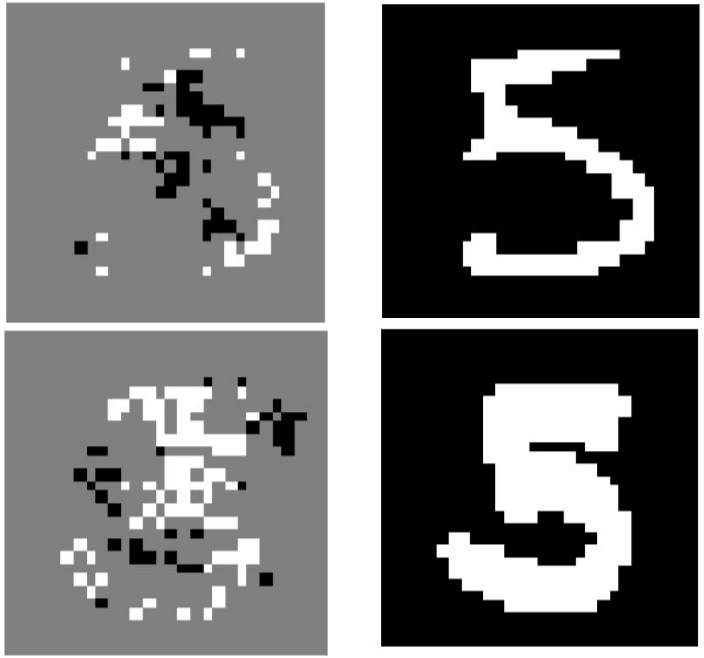
<u>Goal</u>: explain an instance of classification (a specific prediction)

Explanation is a subset of features, s.t.

1. The explanation is "probabilistically sufficient"

   Under the feature distribution, given the explanation, the classifier is likely to make the observed prediction.

2. It is minimal and "simple"



[Khosravi et al. IJCAI19, Wang et al. XXAI20]

Pure Logic **Probabilistic World Models** **Pure Learning**

A New Synthesis of Learning and Reasoning

"*Pure learning is brittle*"

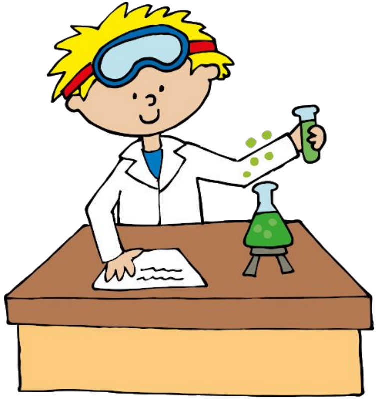
bias, **algorithmic fairness**, interpretability, **explainability**, adversarial attacks, unknown unknowns, calibration, verification, **missing features**, missing labels, data efficiency, shift in distribution, general robustness and safety

We need to incorporate a sensible probabilistic model of the world

Probabilistic Programs

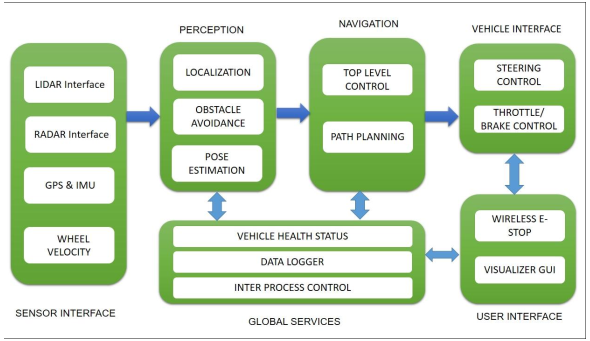
Motivation:
Making modern AI systems is **too hard**

System Builders



Model Builders

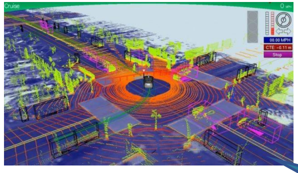
AI Systems Builder



Need to integrate uncertainty over the whole system

20% chance of obstacle!

94% chance of obstacle!



. . .

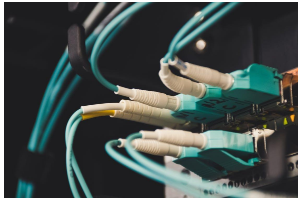
99% certain about current location

Inside the Self-Driving Tesla Fatal Accident

By ANJALI SINGHVI and KARL RUSSELL UPDATED July 12, 2016

The accident may have happened in part because the crash-avoidance system is designed to engage only when radar and computer vision systems agree that there is an obstacle, according to an industry executive with direct

AI Model Builder



"When you have the flu you have a cough 70% of the time"

"Routers fail on average every 5 years"

"What is the probability that a patient with a fever has the flu?"

"What is the probability that my packet will reach the target server?" [SGTVV SIGCOMM'20]

Motivation

1. Making modern AI systems is **too hard**
   – So few experts in probabilistic inference
2. How do we make it easier to build probabilistic systems?
   – Build a *common language* for specifying probabilistic models
   – Design generic *inference algorithms*

Probabilistic Programs

```
let x = flip 0.5 in
let y = flip 0.7 in
let z = x || y in
let w = if z then
      my_func(x,y)
else
      ...
in
observe(z);
```

means "flip a coin, and output true with probability ½"

Standard (functional) programming constructs: let, if, ...

means "reject this execution if z is not true"

Why Probabilistic Programming?

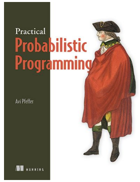
- PPLs are proliferating

Pyro

Edward

HackPPL

Stan



Figaro

Venture, Church, IBAL, WebPPL, Infer.NET, Tensorflow Probability, ProbLog, PRISM, LPADs, CPLogic, CLP(BN), ICL, PHA, Primula, Storm, Gen, PRISM, PSI, Bean Machine, etc. … *and many many more*

- Programming languages are humanity’s biggest knowledge representation achievement!
- Programs should be AI models

Dice language for discrete probabilistic programs

http://dicelang.cs.ucla.edu/ [Holtzen et al. OOPSLA20]

Dice

The dice probabilistic programming language

About GitHub

`dice` is a probabilistic programming language focused on fast exact inference for discrete probabilistic programs. For more information on `dice`, see the about page.

Below is an online `dice` code demo. To run the example code, press the "Run" button.

Run

```
fun sendChar(key: int(2), observation: int(2)) {
  let gen = discrete(0.5, 0.25, 0.125, 0.125) in    // sample a FooLang character
  let enc = key + gen in                            // encrypt the character
  observe observation == enc
}

// sample a uniform random key: A=0, B=1, C=2, D=3

let key = discrete(0.25, 0.25, 0.25, 0.25) in

// observe the ciphertext CCCC
let tmp = sendChar(key, int(2, 2)) in
let tmp = sendChar(key, int(2, 2)) in
let tmp = sendChar(key, int(2, 2)) in
let tmp = sendChar(key, int(2, 2)) in

key
```

Why focus on discrete?

Crucial open problem:

Stan

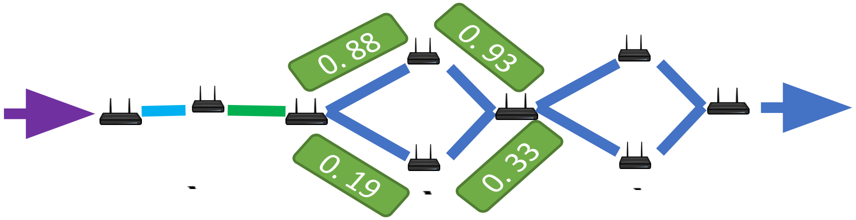
Does not support if-statements!

coroutines. Whenever a discrete variable is encountered in a program's execution, the program is suspended and resumed multiple times with all possible values in the support of that distribution. Listing 10, which implements a simple finite

WebPPL

[AADB+'19]

Network Verification in Dice




```
fun n1(init: bool) {
   let l1succeed = flip 0.99 in
   let l2succeed = flip 0.91 in
   init && l1succeed && l2succeed
}
```

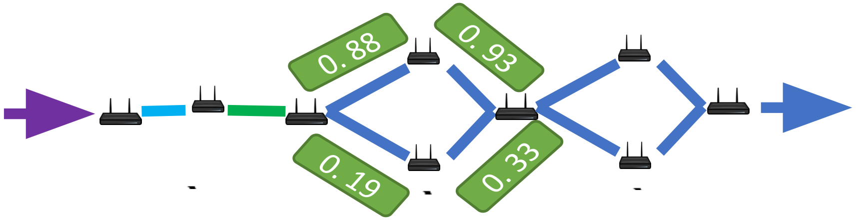
```
fun n2(init: bool) {
   let routeChoice = flip 0.5 in
   if routeChoice then
      init && flip 0.88 && flip 0.93
   else
      init && flip 0.19 && flip 0.33
}
```

ECMP equal-cost path protocol: choose randomly which router to forward to

Main routine, combines the networks

```
n2(n2(n1(true)))
```

Network Verification i




```
fun n1(
   let l1
   let l2
   init &
}
```

This doesn't show all the language features of dice:

- Integers
- Tuples
- Bounded recursion
- Bayesian conditioning
- …

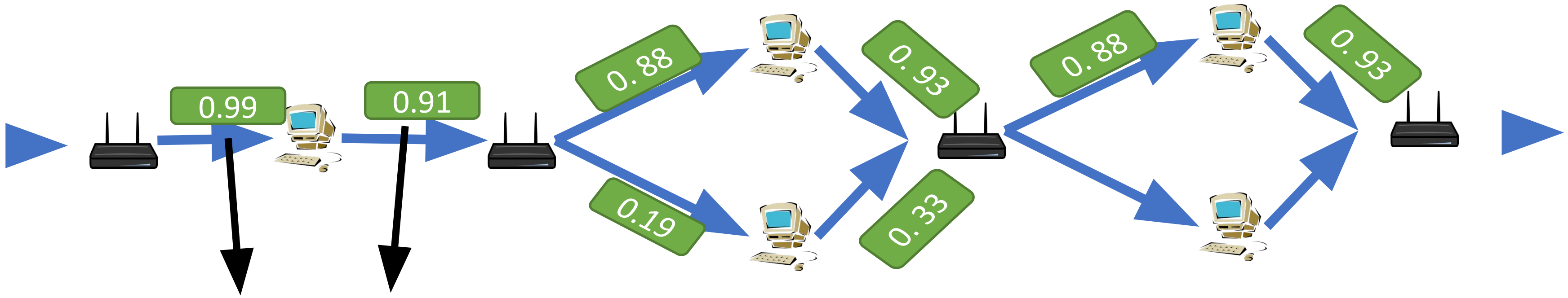
```
fun n2(init: bool) {
   let routeChoice = flip 0.5 in
   if routeChoice then
      init && flip 0.88 && flip 0.93
   else
      init && flip 0.19 && flip 0.33
}
```

ECMP equal-cost path protocol: choose randomly which router to forward to

Main routine, combines the networks

```
n2(n2(n1(true)))
```

Key to Fast Inference: **Factorization** (product nodes)



0.99 x 0.91 x 0.5 x 0.88 x 0.93 x 0.5 x 0.88 x 0.93

\+ 0.99 x 0.91 x 0.5 x 0.19 x 0.33 x 0.5 x 0.88 x 0.93

\+ ...

Easy to see on the graph structure …
how about on the program?

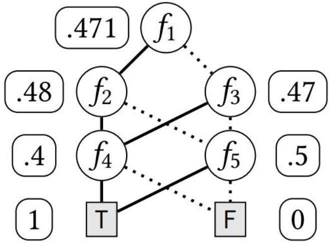
Symbolic Compilation in Dice

- Construct Boolean formula
- Satisfying assignments ≈ paths
- Variables are flips
- Associate weights with flips
- Compile factorized circuit

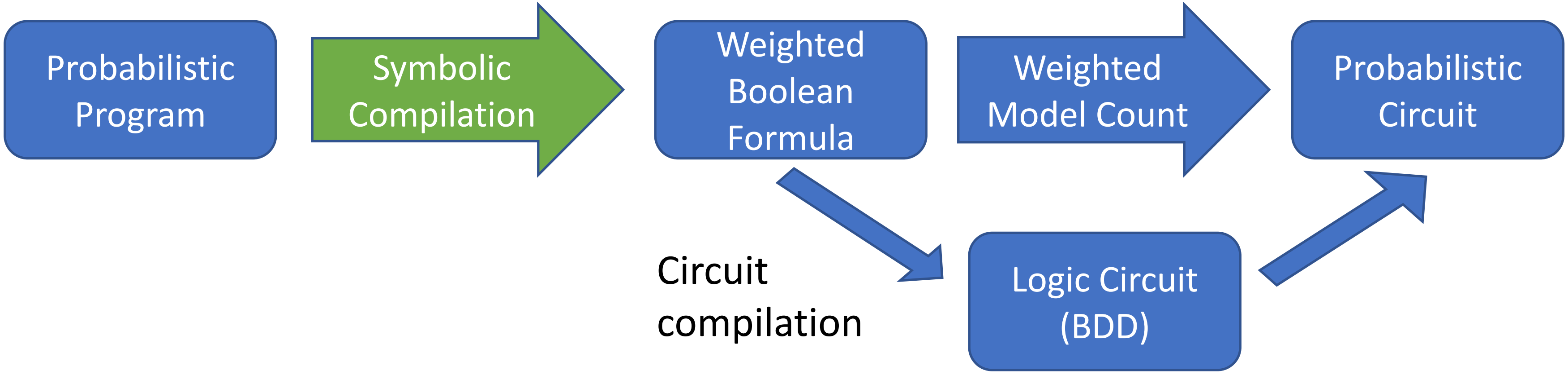
```
let x = flip_1 0.1 in
let y = if x then flip_2 0.2 else
    flip_3 0.3 in
let z = if y then flip_4 0.4 else
    flip_5 0.5 in z
```

$$\underbrace{0.1}_{\text{x=T}} \cdot \underbrace{0.2}_{\text{y=T}} \cdot \underbrace{0.4}_{\text{z=T}} + \underbrace{0.1}_{\text{x=T}} \cdot \underbrace{0.8}_{\text{y=F}} \cdot \underbrace{0.5}_{\text{z=T}} + \underbrace{0.9}_{\text{x=F}} \cdot \underbrace{0.3}_{\text{y=T}} \cdot \underbrace{0.4}_{\text{z=T}} + \underbrace{0.9}_{\text{x=F}} \cdot \underbrace{0.7}_{\text{y=F}} \cdot \underbrace{0.5}_{\text{z=T}}$$

$$f_1 f_2 f_4 \lor f_1 \bar{f}_2 f_5 \lor \bar{f}_1 f_3 f_4 \lor \bar{f}_1 \bar{f}_3 f_5$$



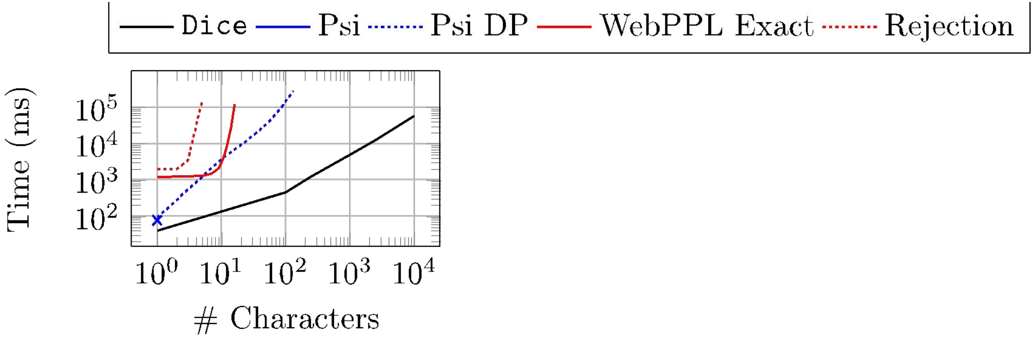
Symbolic Compilation in Dice to Probabilistic Circuits



State of the art for discrete probabilistic program inference!

Experimental Evaluation

- Example from text analysis: breaking a Caesar cipher



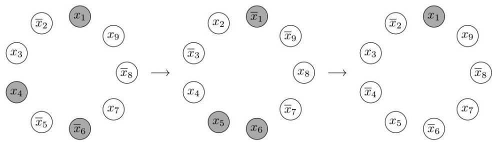
- Competitive with specialized Bayesian network solvers

Benchmark	Psi (ms)	DP (ms)	Dice (ms)	# Parameters	# Paths	BDD Size
Cancer	772	46	**13**	10	1.1×10^3	28
Survey	2477	152	**13**	21	1.3×10^4	73
Alarm	✗	✗	**25**	509	1.0×10^{36}	1.3×10^3
Insurance	✗	✗	**212**	984	1.2×10^{40}	1.0×10^5
Hepar2	✗	✗	**54**	48	2.9×10^{69}	1.3×10^3
Hailfinder	✗	✗	**618**	2656	2.0×10^{76}	6.5×10^4
Pigs	✗	✗	**72**	5618	7.3×10^{492}	35
Water	✗	✗	**2590**	1.0×10^4	3.2×10^{54}	5.1×10^4
Munin	✗	✗	**1866**	8.1×10^5	2.1×10^{1622}	1.1×10^4

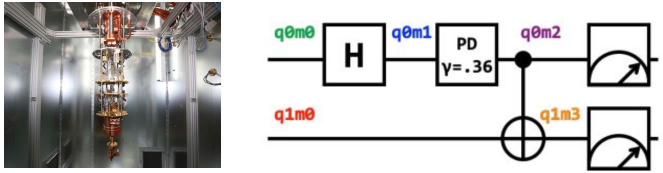
More program paths than atoms in the universe

If you build it they will come

- As soon as ***dice*** was put online people started using it in surprising ways we had not foreseen



Probabilistic Model Checking (verify randomized algorithms)



Quantum Simulation

- In both cases, ***dice*** outperforms existing specialized methods on important examples!

Conclusions

- Are we already in the age of computational abstractions?
- **Probabilistic circuits** for learning deep tractable probabilistic models
- **Probabilistic programs** as the new probabilistic knowledge representation language
- Two computational abstractions go hand in hand

Thanks

This was the work of many wonderful students/postdoc/collaborators!

References: http://starai.cs.ucla.edu/publications/