From Probabilistic Circuits to Probabilistic Programs and Back

Guy Van den Broeck

ICAART - Feb 6, 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 \( \text{Smokes}(x) \land \text{Friends}(x,y) \implies \text{Smokes}(y) \)
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
\[
\begin{align*}
\mu_k &\sim \text{Normal}(\alpha, \beta) \\
\sigma_k &\sim \text{Gamma}(\nu, \rho) \\
\theta_k &\sim \text{Dirichlet}(\kappa) \\
x_i &\sim \begin{cases} 
\text{Categorical}(\text{init}) & \text{if } i = 0 \\
\text{Categorical}(\theta_{x_{i-1}}) & \text{if } i > 0
\end{cases} \\
y_i &\sim \text{Normal}(\mu_{x_i}, \sigma_{x_i})
\end{align*}
\]

[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
The Alphabet Soup of probabilistic models
Intractable and tractable models
"Every talk needs a joke and a literature overview slide, not necessarily distinct"
- after Ron Graham
a **unifying framework** for tractable models
Input nodes $c$ are tractable (simple) distributions, e.g., univariate gaussian or indicator $p_c(X=1) = [X=1]$
Product nodes are factorizations $\prod_{c \in \text{in}(n)} p_c(x)$.
Sum nodes are mixture models $\sum_{c \in \text{in}(n)} \theta_{n,c} p_c(x)$
*Smoothness* + *decomposability* = \text{tractable MAR}

If \( p(x) = \sum_i w_i p_i(x) \), *(smoothness)*:

\[
\int p(x) \, dx = \int \sum_i w_i p_i(x) \, dx =
\]

\[
= \sum_i w_i \int p_i(x) \, dx
\]

\( \implies \) *integrals are “pushed down” to children*

[Darwiche & Marquis JAIR 2001, Poon & Domingos UAI11]
Smoothness + decomposability = tractable MAR

If \( p(x, y, z) = p(x)p(y)p(z) \), (decomposability):

\[
\int \int \int p(x, y, z) dx dy dz = \\
= \int \int \int p(x)p(y)p(z) dx dy dz = \\
= \int p(x) dx \int p(y) dy \int p(z) dz
\]

\[\Rightarrow \text{integrals decompose into easier ones}\]
\textbf{Smoothness} + \textbf{decomposability} = \textbf{tractable MAR}

Forward pass evaluation for MAR

\[ \implies \text{linear in circuit size!} \]

E.g. to compute \( p(x_2, x_4) \):
- leafs over \( X_1 \) and \( X_3 \) output \( Z_i = \int p(x_i) \, dx_i \)
  \[ \implies \text{for normalized leaf distributions:} \quad 1.0 \]
- leafs over \( X_2 \) and \( X_4 \) output \textbf{EVI}
- feedforward evaluation (bottom-up)
<table>
<thead>
<tr>
<th>Property</th>
<th>MAR</th>
<th>CON</th>
<th>MOM</th>
<th>MAP</th>
<th>MMAP</th>
<th>ENT</th>
<th>DIV</th>
<th>EXP</th>
<th>Expected Predictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>smoothness</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>decomposability</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>consistency</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>determinism</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>marginal determinism</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>structured decomposability</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>paired str. decomposability</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
tractability is a spectrum
<table>
<thead>
<tr>
<th>Model</th>
<th>smooth</th>
<th>dec.</th>
<th>det.</th>
<th>str.dec.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic Circuits (ACs)</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>Sum-Product Networks (SPNs)</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Cutset Networks (CNets)</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>Probabilistic Decision Graphs</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>(Affine) ADDs</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>AndOrGraphs</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>PSDDs</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>
Expressive models without compromises
# How expressive are probabilistic circuits?

**density estimation benchmarks**

<table>
<thead>
<tr>
<th>dataset</th>
<th>best circuit</th>
<th>BN</th>
<th>MADE</th>
<th>VAE</th>
<th>dataset</th>
<th>best circuit</th>
<th>BN</th>
<th>MADE</th>
<th>VAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>nltcs</td>
<td>-5.99</td>
<td>-6.02</td>
<td>-6.04</td>
<td>-5.99</td>
<td>dna</td>
<td>-79.88</td>
<td>-80.65</td>
<td>-82.77</td>
<td>-94.56</td>
</tr>
<tr>
<td>msnbc</td>
<td>-6.04</td>
<td>-6.04</td>
<td>-6.06</td>
<td>-6.09</td>
<td>kosarek</td>
<td>-10.52</td>
<td>-10.83</td>
<td>-</td>
<td>-10.64</td>
</tr>
<tr>
<td>plants</td>
<td>-11.84</td>
<td>12.65</td>
<td>12.32</td>
<td>12.34</td>
<td>book</td>
<td>-33.82</td>
<td>-36.41</td>
<td>-33.95</td>
<td>-33.19</td>
</tr>
<tr>
<td>audio</td>
<td>-39.39</td>
<td>-40.50</td>
<td>-38.95</td>
<td>-38.67</td>
<td>movie</td>
<td>-50.34</td>
<td>-54.37</td>
<td>-48.7</td>
<td>-47.43</td>
</tr>
<tr>
<td>jester</td>
<td>-51.29</td>
<td>-51.07</td>
<td>-52.23</td>
<td>-51.54</td>
<td>webkb</td>
<td>-149.20</td>
<td>-157.43</td>
<td>-149.59</td>
<td>-146.9</td>
</tr>
<tr>
<td>netflix</td>
<td>-55.71</td>
<td>-57.02</td>
<td>-55.16</td>
<td>-54.73</td>
<td>cr52</td>
<td>-81.87</td>
<td>-87.56</td>
<td>-82.80</td>
<td>-81.33</td>
</tr>
<tr>
<td>accidents</td>
<td>-26.89</td>
<td>-26.32</td>
<td>-26.42</td>
<td>-29.11</td>
<td>c20ng</td>
<td>-151.02</td>
<td>-158.95</td>
<td>-153.18</td>
<td>-146.9</td>
</tr>
<tr>
<td>retail</td>
<td>-10.72</td>
<td>-10.87</td>
<td>-10.81</td>
<td>-10.83</td>
<td>bbc</td>
<td>-229.21</td>
<td>-257.86</td>
<td>-242.40</td>
<td>-240.94</td>
</tr>
</tbody>
</table>
Want to learn more?

Tutorial (3h)

Probabilistic Circuits

Overview Paper (80p)

Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Models

YooJung Choi
Antonio Vergari
Guy Van den Broeck

Contents
1 Introduction
2 Probabilistic Inference: Models, Queries, and Tractability
   2.1 Probabilistic Models
   2.2 Probabilistic Queries
   2.3 Tractable Probabilistic Inference
   2.4 Properties of Tractable Probabilistic Models

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

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?
Determinantal Point Processes (DPPs)

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

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

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

Computing marginal probabilities is *tractable.*

[Zhang et al. UAI20]
We cannot tractably represent DPPs with classes of PCs … yet

An *almost* universal tractable language

Stay Tuned!

[Zhang et al. UAI20; Martens & Medabalimi Arxiv15]
The AI Dilemma

Pure Logic  Pure Learning
The AI Dilemma

Pure Logic

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

Pure Learning
The AI Dilemma

Pure Logic

- 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 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

Train

Classifier

Predict

Test with missing features
Expected Predictions

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

\[ E_{x^m \sim p(x^m | x^o)} \left[ f(x^m, x^o) \right] \]

\( x^o = \) observed features \\
\( x^m = \) missing features

Experiment:

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

[Graphs showing accuracy and RMSE for different datasets (MNIST, FMNIST, Abalone, Delta, Insurance) as a function of percentage of missing data.]

[References: Khosravi et al. IJCAI19, NeurIPS20, Artemiss20]
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.
Probabilistic Sufficient Explanations

**Goal:** 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 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
What are probabilistic programs?

```plaintext
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);
```

- `let x = flip 0.5 in` means “flip a coin, and output true with probability $\frac{1}{2}$”
- Standard (functional) programming constructs: `let`, `if`, `...`
- `let w = if z then
  my_func(x,y)
else
  ...
in
observe(z);` 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, CProlog, 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** probabilistic programming language

http://dicelang.cs.ucla.edu/

https://github.com/SHoltzen/dice

---

### Dice

The dice probabilistic programming language

---

**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.

```plaintext
fun sendChar(key: int(2), observation: int(2)) {
    let gen = discrete(.15, .65, .25, .155, .155) in // sample a Foolang character
    let enc = key + gen in // encrypt the character
    observe observation == enc
}
```

---

[Holtzen et al. OOPSLA20]
Why should I care?

Better abstraction than probabilistic graphical models:

• Beyond variable-level dependencies (contextual)
• modularity through functions reuse (cf. relational graphical models)
• intuitive language for local structure; arithmetic
• data structures
• first-class observations
First-Class Observations

```kotlin
fun EncryptChar(key:int, obs:char):Bool {
    let randomChar = ChooseChar() in
    let ciphertext = (randomChar + key) % 26 in
    let _ = observe ciphertext = obs in
    true}
let k = UniformInt(0, 25) in
let _ = EncryptChar(k, 'H') in ...
let _ = EncryptChar(k, 'D') in k
```

Frequency Analyzer for a Caesar cipher in Dice
Probabilistic Program Inference

Key ingredient: factorization .... aka the product nodes

```
1 let x = flip₁ 0.1 in
2 let y = if x then flip₂ 0.2 else flip₃ 0.3 in
4 let z = if y then flip₄ 0.4 else flip₅ 0.5 in z
```

\[
\begin{align*}
0.1 \cdot 0.2 \cdot 0.4 & + 0.1 \cdot 0.8 \cdot 0.5 & + 0.9 \cdot 0.3 \cdot 0.4 & + 0.9 \cdot 0.7 \cdot 0.5 \\
\hline
x=T & y=T & z=T & x=F & y=F & z=T & x=F & y=T & z=T & x=F & y=F & z=T
\end{align*}
\]
Symbolic Compilation in Dice

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

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

\[
\begin{align*}
0.1 & \cdot 0.2 \cdot 0.4 + 0.1 \cdot 0.8 \cdot 0.5 + 0.9 \cdot 0.3 \cdot 0.4 + 0.9 \cdot 0.7 \cdot 0.5 \\
x=T & \quad y=T \quad z=T \quad x=T \quad y=F \quad z=T \quad x=F \quad y=F \quad z=T
\end{align*}
\]

\[
\text{let } x = \text{flip}_1 0.1 \text{ in } \\
\text{let } y = \text{if } x \text{ then } \text{flip}_2 0.2 \text{ else } \text{flip}_3 0.3 \text{ in } \\
\text{let } z = \text{if } y \text{ then } \text{flip}_4 0.4 \text{ else } \text{flip}_5 0.5 \text{ in } z
\]
Symbolic Compilation to Probabilistic Circuits

- Probabilistic Program
- Symbolic Compilation
- Weighted Boolean Formula
- Weighted Model Count
- Probabilistic Circuit

Circuit compilation

Logic Circuit (BDD)

State of the art for discrete probabilistic program inference!
Factorized Inference in Dice

```
fun diamond(s1:Boolean):Boolean {
  let route = flip1 0.5 in
  let s2 = if route then s1 else F in
  let s3 = if route then F else s1 in
  let drop = flip2 0.0001 in
  s2 ∨ (s3 ∧ ¬drop))
  diamond(diamond(diamond(T)))
}
```

Network Verification
PPL benchmarks from PL community

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Psi (ms)</th>
<th>DP (ms)</th>
<th>Dice (ms)</th>
<th># Paths</th>
<th>BDD Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grass</td>
<td>167</td>
<td>57</td>
<td>1.0</td>
<td>9.5x10^{1}</td>
<td>15</td>
</tr>
<tr>
<td>Burglar Alarm</td>
<td>98</td>
<td>10</td>
<td>1.1</td>
<td>2.5x10^{2}</td>
<td>11</td>
</tr>
<tr>
<td>Coin Bias</td>
<td>94</td>
<td>23</td>
<td>1.0</td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>Noisy Or</td>
<td>81</td>
<td>152</td>
<td>1.0</td>
<td>1.6x10^{4}</td>
<td>35</td>
</tr>
<tr>
<td>Evidence1</td>
<td>48</td>
<td>32</td>
<td>1.0</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>Evidence2</td>
<td>59</td>
<td>28</td>
<td>1.0</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>Murder Mystery</td>
<td>193</td>
<td>75</td>
<td>1.0</td>
<td>1.6x10^{1}</td>
<td>6</td>
</tr>
</tbody>
</table>
## Scalable Inference

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Psi (ms)</th>
<th>DP (ms)</th>
<th>Dice (ms)</th>
<th># Parameters</th>
<th># Paths</th>
<th>BDD Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancer [48]</td>
<td>772</td>
<td>46</td>
<td>1.0</td>
<td>10</td>
<td>1.1×10³</td>
<td>28</td>
</tr>
<tr>
<td>Survey [73]</td>
<td>2477</td>
<td>152</td>
<td>2.0</td>
<td>21</td>
<td>1.3×10⁴</td>
<td>73</td>
</tr>
<tr>
<td>Alarm [5]</td>
<td>$\times$</td>
<td>$\times$</td>
<td>9.0</td>
<td>509</td>
<td>1.0×10₃⁶</td>
<td>1.3×10³</td>
</tr>
<tr>
<td>Insurance [7]</td>
<td>$\times$</td>
<td>$\times$</td>
<td>75.0</td>
<td>984</td>
<td>1.2×10⁴⁰</td>
<td>1.0×10⁵</td>
</tr>
<tr>
<td>Hepar2 [63]</td>
<td>$\times$</td>
<td>$\times$</td>
<td>54.0</td>
<td>1453</td>
<td>2.9×10⁶⁹</td>
<td>1.3×10³</td>
</tr>
<tr>
<td>Hailfinder [1]</td>
<td>$\times$</td>
<td>$\times$</td>
<td>526.0</td>
<td>2656</td>
<td>2.0×10⁷⁶</td>
<td>6.5×10⁴</td>
</tr>
<tr>
<td>Pigs</td>
<td>$\times$</td>
<td>$\times$</td>
<td>32.0</td>
<td>5618</td>
<td>7.3×10⁴⁹²</td>
<td>35</td>
</tr>
<tr>
<td>Water [43]</td>
<td>$\times$</td>
<td>$\times$</td>
<td>2926.0</td>
<td>1.0×10⁴</td>
<td>3.2×10⁵⁴</td>
<td>5.1×10⁴</td>
</tr>
<tr>
<td>Munin [3]</td>
<td>$\times$</td>
<td>$\times$</td>
<td>1945.0</td>
<td>8.1×10⁵</td>
<td>2.1×10⁶²²</td>
<td>1.1×10⁴</td>
</tr>
</tbody>
</table>
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

My students/postdoc who did the real work are graduating.

There are some awesome people on the academic job market!