Sum-Product Networks: A New Deep Architecture

Hoifung Poon
Microsoft Research

Joint work with Pedro Domingos
Graphical Models: Challenges

Bayesian Network

Markov Network

Restricted Boltzmann Machine (RBM)

Advantage: Compactly represent probability

Problem: Inference is intractable

Problem: Learning is difficult
Deep Learning

- Stack many layers
  - E.g.: DBN [Hinton & Salakhutdinov, 2006]
  - CDBN [Lee et al., 2009]
  - DBM [Salakhutdinov & Hinton, 2010]

- Potentially much more powerful than shallow architectures [Bengio, 2009]

- But …
  - Inference is even harder
  - Learning requires extensive effort
Learning: Requires approximate inference
Inference: Still approximate
Graphical Models

E.g., hierarchical mixture model, thin junction tree, etc.

Problem: Too restricted

Existing Tractable Models
This Talk: Sum-Product Networks

Compactly represent partition function using a deep network

Graphical Models

Existing Tractable Models

Sum-Product Networks
Graphical Models

Exact inference linear time in network size

Sum-Product Networks
Can compactly represent many more distributions

Graphical Models

Existing Tractable Models

Sum-Product Networks
Learn optimal way to reuse computation, etc.

Graphical Models

Existing Tractable Models

Sum-Product Networks
Outline

- Sum-product networks (SPNs)
- Learning SPN
- Experimental results
- Conclusion
Why Is Inference Hard?

\[ P(\mathbf{X}_1, \cdots, \mathbf{X}_N) = \frac{1}{Z} \prod_j \Phi_j(\mathbf{X}_1, \cdots, \mathbf{X}_N) \]

- Bottleneck: Summing out variables
- E.g.: Partition function
  
  Sum of exponentially many products

\[ Z = \sum_{\mathbf{X}} \prod_j \Phi_j(\mathbf{X}) \]
Alternative Representation

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$P(X)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0.3</td>
</tr>
</tbody>
</table>

$P(X) = 0.4 \cdot I[X_1=1] \cdot I[X_2=1]$

$+ 0.2 \cdot I[X_1=1] \cdot I[X_2=0]$

$+ 0.1 \cdot I[X_1=0] \cdot I[X_2=1]$

$+ 0.3 \cdot I[X_1=0] \cdot I[X_2=0]$
Alternative Representation

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

Network Polynomial [Darwiche, 2003]
Shorthand for Indicators

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$P(X)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0.3</td>
</tr>
</tbody>
</table>

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

Network Polynomial [Darwiche, 2003]
Sum Out Variables

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$P(X)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0.3</td>
</tr>
</tbody>
</table>

\[
P(e) = 0.4 \cdot X_1 \cdot X_2 \\
+ 0.2 \cdot X_1 \cdot \overline{X}_2 \\
+ 0.1 \cdot \overline{X}_1 \cdot X_2 \\
+ 0.3 \cdot \overline{X}_1 \cdot \overline{X}_2
\]

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

Easy: Set both indicators to 1
Graphical Representation

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$P(X)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0.3</td>
</tr>
</tbody>
</table>
But ... Exponentially Large

Example: Parity

Uniform distribution over states with even number of 1's
But … Exponentially Large

Can we make this more compact?
Use a Deep Network

Example: Parity

Uniform distribution over states with even number of 1’s
Use a Deep Network

Example: Parity
Uniform distribution over states of even number of 1’s

Induce many hidden layers

Reuse partial computation
Arithmetic Circuits (ACs)

- Data structure for efficient inference
  - Darwiche [2003]
  - Compilation target of Bayesian networks

**Key idea:** Use ACs instead to define a new class of deep probabilistic models

- Develop new deep learning algorithms for this class of models
Sum-Product Networks (SPNs)

- Rooted DAG
- Nodes: Sum, product, input indicator
- Weights on edges from sum to children
Distribution Defined by SPN

\[ P(X) \propto S(X) \]
Can We Sum Out Variables?

\[ P(e) \propto \sum_{X \sim e} S(X) \quad ? \quad S(e) \]

\[
e: X_1 = 1
\]

\[
\begin{array}{cccc}
X_1 & \overline{X}_1 & X_2 & \overline{X}_2 \\
1 & 0 & 1 & 1 \\
0.6 & 0.4 & 0.9 & 0.1 \\
0.7 & 0.3 & 0.7 & 0.2 \\
0.8 & & & \\
\end{array}
\]
Valid SPN

- \( SPN \) is valid if \( S(e) = \sum_{X \sim e} S(X) \) for all \( e \)

- Valid \( \rightarrow \) Can compute marginals efficiently

- Partition function \( Z \) can be computed by setting all indicators to 1
Valid SPN: General Conditions

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

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

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

\[ S(e) \leq \sum_{X \sim e} S(X) \]

\[ S(e) \geq \sum_{X \sim e} S(X) \]
Semantics of Sums and Products

- Product $\sim$ Feature $\rightarrow$ Form feature hierarchy
- Sum $\sim$ Mixture (with hidden var. summed out)

\[
\begin{align*}
&= \sum_{i} w_{ij} \prod_{j} I[Y_i = j] \\
&= \sum_{i} w_{ij}
\end{align*}
\]
Inference

Probability: \( P(X) = \frac{S(X)}{Z} \)

\( X: X_1 = 1, X_2 = 0 \)

<table>
<thead>
<tr>
<th>( X_1 )</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \overline{X}_1 )</td>
<td>0</td>
</tr>
<tr>
<td>( X_2 )</td>
<td>0</td>
</tr>
<tr>
<td>( \overline{X}_2 )</td>
<td>1</td>
</tr>
</tbody>
</table>
Inference

If weights sum to 1 at each sum node
Then \( Z = 1, \ P(X) = S(X) \)

\[ X: X_1 = 1, \ X_2 = 0 \]

<table>
<thead>
<tr>
<th></th>
<th>( X_1 )</th>
<th>( \overline{X}_1 )</th>
<th>( X_2 )</th>
<th>( \overline{X}_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X_1 )</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \overline{X}_1 )</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( X_2 )</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>( \overline{X}_2 )</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Inference

Marginal: \( P(e) = S(e) / Z \)

e: \( X_1 = 1 \)

<table>
<thead>
<tr>
<th></th>
<th>( X_1 )</th>
<th>( \bar{X}_1 )</th>
<th>( X_2 )</th>
<th>( \bar{X}_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P(e) )</td>
<td>0.6</td>
<td>0.4</td>
<td>0.9</td>
<td>0.1</td>
</tr>
<tr>
<td>( X_1 )</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( \bar{X}_1 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( X_2 )</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( \bar{X}_2 )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\( 0.69 = 0.51 + 0.18 \)
MPE: Replace sums with maxes

e: $X_1 = 1$

0.7 \times 0.42 = 0.294

$0.7 \times 0.42 = 0.294$

$0.3 \times 0.72 = 0.216$

Darwiche [2003]
Inference

MAX: Pick child with highest value

e: \( X_1 = 1 \)

\[
\begin{array}{c|c}
X_1 & 1 \\
\overline{X}_1 & 0 \\
X_2 & 1 \\
\overline{X}_2 & 1 \\
\end{array}
\]

\[
0.7 \times 0.42 = 0.294
\]

\[
0.3 \times 0.72 = 0.216
\]

Darwiche [2003]
Handling Continuous Variables

- Sum $\rightarrow$ Integral over input

- Simplest case: Indicator $\rightarrow$ Gaussian

SPN compactly defines a very large mixture of Gaussians
SPNs Everywhere

- Graphical models

  - Existing tractable mdls. & inference mthds.
  - Determinism, context-specific indep., etc.
  - Can potentially learn the optimal way
SPNs Everywhere

- Graphical models
- Methods for efficient inference

E.g., arithmetic circuits, AND/OR graphs, case-factor diagrams

SPNs are a class of probabilistic models
SPNs have validity conditions
SPNs can be learned from data
SPNs Everywhere

- Graphical models
- Models for efficient inference
- General, probabilistic convolutional network

Sum: Average-pooling
Max: Max-pooling
SPNs Everywhere

- Graphical models
- Models for efficient inference
- General, probabilistic convolutional network
- Grammars in vision and language

E.g., object detection grammar, probabilistic context-free grammar

Sum: Non-terminal
Product: Production rule
Outline

- Sum-product networks (SPNs)
- Learning SPN
- Experimental results
- Conclusion
General Approach

- Start with a dense SPN
- Find the structure by learning weights
  Zero weights signify absence of connections
- Can learn with gradient descent or EM
The Challenge

- **Gradient diffusion**: Gradient quickly dilutes
- Similar problem with EM
- **Hard EM overcomes this problem**
Our Learning Algorithm

- Online learning + Hard EM
- Sum node maintains counts for each child
- For each example
  - Find MPE instantiation with current weights
  - Increment count for each chosen child
  - Renormalize to set new weights
- Repeat until convergence
Outline

- Sum-product networks (SPNs)
- Learning SPN
- **Experimental results**
- Conclusion
Task: Image Completion

- Methodology:
  - Learn a model from training images
  - Complete unseen test images
  - Measure mean square errors

- Very challenging

- Good for evaluating deep models
Datasets

- **Main evaluation: Caltech-101** [Fei-Fei et al., 2004]
  - 101 categories, e.g., faces, cars, elephants
  - Each category: 30 – 800 images

- Also, Olivetti [Samaria & Harter, 1994] (400 faces)
  - Each category: Last third for test

  **Test images: Unseen objects**
SPN Architecture

Whole Image

Region

Pixel
Decomposition
Decomposition
Systems

- SPN
- DBM [Salakhutdinov & Hinton, 2010]
- DBN [Hinton & Salakhutdinov, 2006]
- PCA [Turk & Pentland, 1991]
- Nearest neighbor [Hays & Efros, 2007]
Caltech: Mean-Square Errors

Left

Bottom

<table>
<thead>
<tr>
<th></th>
<th>NN</th>
<th>PCA</th>
<th>DBN</th>
<th>DBM</th>
<th>SPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bottom</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>NN</th>
<th>PCA</th>
<th>DBN</th>
<th>DBM</th>
<th>SPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bottom</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bar chart showing mean-square errors for different models on Caltech dataset.
SPN vs. DBM / DBN

- SPN is order of magnitude faster

<table>
<thead>
<tr>
<th></th>
<th>SPN</th>
<th>DBM / DBN</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Learning</strong></td>
<td>2-3 hours</td>
<td>Days</td>
</tr>
<tr>
<td><strong>Inference</strong></td>
<td>&lt; 1 second</td>
<td>Minutes or hours</td>
</tr>
</tbody>
</table>

- No elaborate preprocessing, tuning
- Reduced errors by 30-60%
- Learned up to 46 layers
Example Completions

Original

SPN

DBM

DBN

PCA

Nearest Neighbor
Example Completions

Original

SPN

DBM

DBN

PCA

Nearest Neighbor
Example Completions

Original

SPN

DBM

DBN

PCA

Nearest Neighbor
Example Completions

Original

SPN

DBM

DBN

PCA

Nearest Neighbor
Example Completions

Original

SPN

DBM

DBN

PCA

Nearest Neighbor
Example Completions

Original

SPN

DBM

DBN

PCA

Nearest Neighbor
Open Questions

- Other learning algorithms
- Discriminative learning
- Architecture
- Continuous SPNs
- Sequential domains
- Other applications
End-to-End Comparison

Given same computation budget, which approach has better performance?
True Model

Graphical Models

Approximate Inference

Optimal SPN

Existing Tractable Models

Sum-Product Networks

Existing Tractable Models

Existing Tractable Models
Conclusion

- Sum-product networks (SPNs)
  - DAG of sums and products
  - Compactly represent partition function
  - Learn many layers of hidden variables

- Exact inference: Linear time in network size

- Deep learning: Online hard EM

- Substantially outperform state of the art on image completion