An Introduction to Sum Product Networks

José Miguel Hernández-Lobato

Department of Engineering, Cambridge University

April 5, 2013

Probabilistic Modeling of Data

Requires to specify a high-dimensional distribution $p(x_1, \ldots, x_k)$ on the **data** and possibly some **latent variables**. The specific form of p will depend on some **parameters** w.

The basic operations will be to adjust $p(x_1, \ldots, x_k)$ to the data (learning), and to compute its marginals and modes (inference).

Working with fully flexible joint distributions is intractable!

We must work with structured or compact distributions. For example, distributions in which the random variables interact directly with only very few others in simple ways.

One solution is to use probabilistic graphical models.

Probabilistic Graphical Models

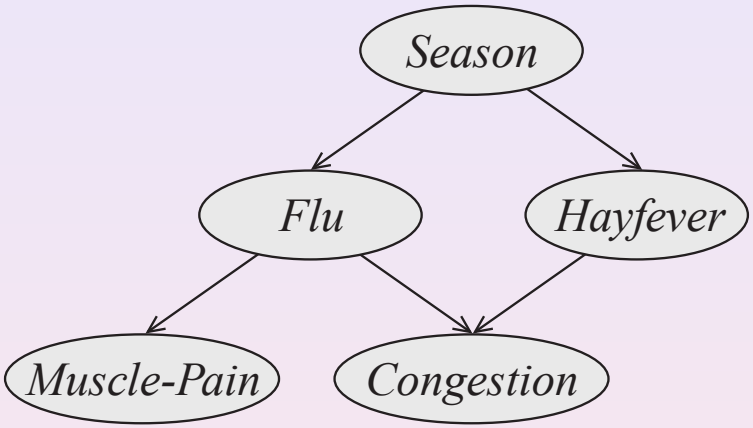
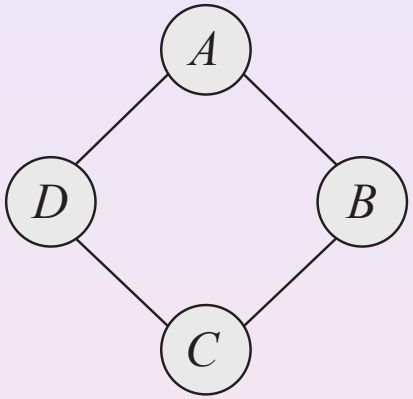
	Bayesian Network	Markov Network
Graphs		
Independencies	$(F \perp H \mid C)$, $(C \perp S \mid F, H)$ $(M \perp H, C \mid F)$, $(M \perp C \mid F)$, ...	$(A \perp C \mid B, D)$, $(B \perp D \mid A, C)$
Factorization	$p(S, F, H, M, C) = p(S)p(F \mid S)$ $p(H \mid S)p(C \mid F, H)p(M \mid F)$	$p(A, B, C, D) = \frac{1}{Z}\phi_1(A, B)$ $\phi_2(B, C)\phi_3(C, D)\phi_4(A, D)$

Figure source [Koller et al. 2009].

Limitations of Graphical Models

GM are limited in some aspects:

- Many compact distributions cannot be represented as a GM, e.g., uniform distribution over binary vectors with even number of 1's.
- The cost of exact inference in GM is exponential in the worst case. This means that we will often have to use approximate techniques.
- Because learning requires inference, learning GM will be difficult.
- Some distributions require GM with many layers of hidden variables to be compactly encoded. However, intractable inference makes learning these models extremely challenging.

An alternative are sum product networks [Poon and Domingos, 2011]:

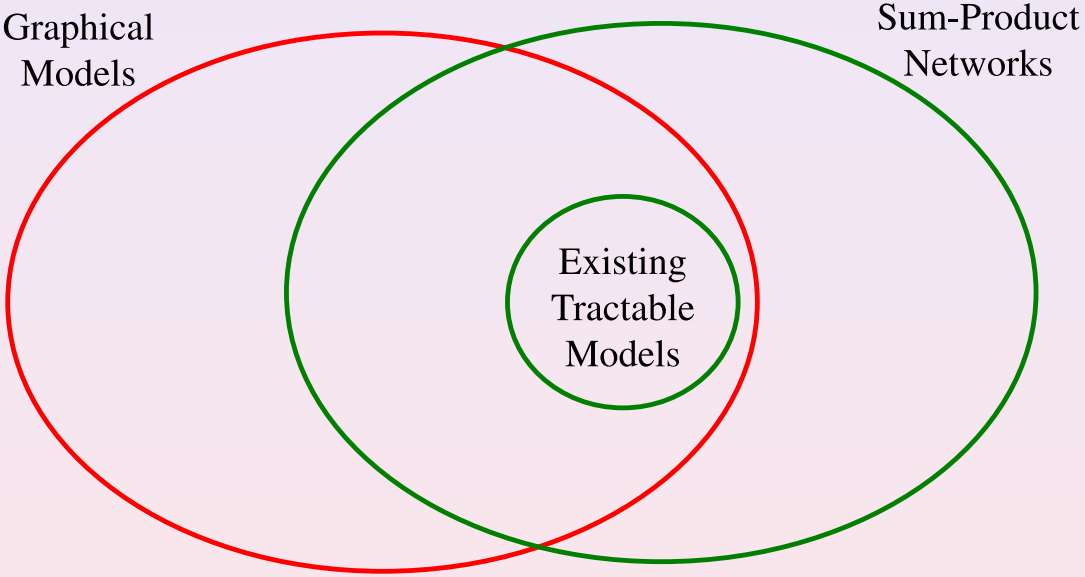
- New deep model with many layers of hidden variables.
- Exact inference is tractable (linear in the size of the model).

Sum Product Networks and GMs

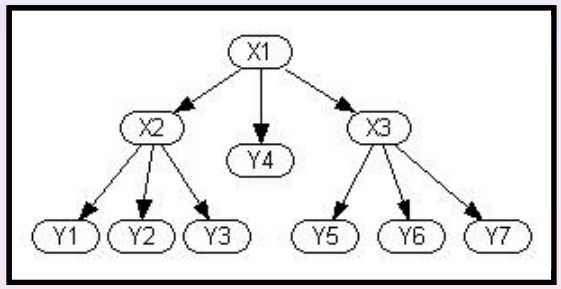
SNPs are more general than other tractable GMs:

1 - Hierarchical mixture models [Zhang et al 2004].

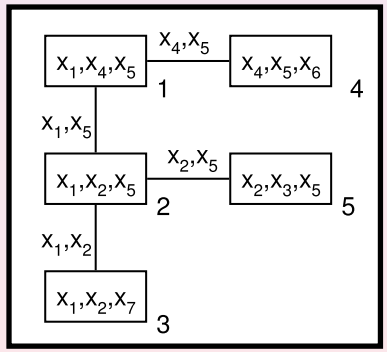
2 - Thin junction trees [Chechetka et al. 2008].



Hierarchical Mixture Model



Junction Tree



Figures: Pedro Domingos, [Zhang et al. 2004], [Chechetka et al. 2008].

Network Polynomial I

SPNs are based on the notion of network polynomial [Darwiche, 2003].

Indicator functions on a binary random variable X_i and its negation $\bar{X}_i$:

1 - $[X_i] = 1$ when $X_i = 1$ and 0 otherwise.
2 - $[\bar{X}_i] = 1$ when $X_i = 0$ and 0 otherwise.
3 - We abbreviate $[X_i]$ and $[\bar{X}_i]$ by x_i and $\bar{x}_i$, respectively.
3 - Instantiations of X_i and $\bar{X}_i$ are represented by $\bar{x}_i$ and x_i.

Let $\Phi(\mathbf{x})$ an **unnormalized** distribution of a vector of binary variables $\mathbf{X}$.

The network polynomial of $\Phi(\mathbf{x})$, $\mathbf{x} = (x_1, \ldots, x_n)$ is $\sum_{\mathbf{x}} \Phi(\mathbf{x}) \prod(\mathbf{x})$, where $\prod(\mathbf{x})$ is the product of indicators which are one in state $\mathbf{x}$.

Example I: the NP of the Bernoulli distribution $\text{Bern}(x_1|p)$ is

$$f(x_1, \bar{x}_1) = p x_1 + (1 - p)\bar{x}_1 \,.$$

Network Polynomial II

Example II: the NP of the Bayesian network $X_1 \rightarrow X_2$ is

$$f(x_1, \bar{x}_1, x_2, \bar{x}_2) = p(x_2|x_1)p(x_1)x_1x_2 + p(x_2|x_1)p(\bar{x}_1)\bar{x}_1x_2 + p(\bar{x}_2|x_1)p(x_1)x_1\bar{x}_2 + p(\bar{x}_2|\bar{x}_1)p(\bar{x}_1)\bar{x}_1\bar{x}_2 .$$

The NP for $\Phi(\mathbf{x})$, that is, $\sum_{\mathbf{x}} \Phi(\mathbf{x}) \prod(\mathbf{x})$

- is a **multivariate** polynomial where each variable has degree 1.
- is a **multilinear** function over the indicators of $X_1, \ldots, X_n$.
- has an **exponential** number of terms, one for each possible state of $\mathbf{x}$.

Let $\mathbf{E}$ be a set of states. We compute $\sum_{\mathbf{x} \in \mathbf{E}} \Phi(\mathbf{x})$ by evaluating f when only the indicators consistent with $\mathbf{E}$ are active.

For example, $\sum_{\mathbf{x}: x_1 = 1} \Phi(\mathbf{x}) = f(x_1 = 1, \bar{x}_1 = 0, x_2 = 1, \bar{x}_2 = 1)$.

We can marginalize Φ over a set of states $\mathbf{E}$ in a single evaluation of f!!!!

Network Polynomial III

The NP is an alternative way to encode a probability table.

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

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

Network Polynomial IV

The NP allows to easily marginalize Φ by activating specific indicators.

When $\mathbf{E} = \{\mathbf{x} : x_1 = 1\}$, $\sum_{\mathbf{x} \in \mathbf{E}} \Phi(\mathbf{x})$ is computed by evaluating the NP on $x_1 = 1, \bar{x}_1 = 0, x_2 = 1, \bar{x}_2 = 1$.

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

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

Network Polynomial V

Two reasons for activating the indicator x_i, that is, setting $x_i = 1$:

1 - Evidence. We know $x_i = 1$. In this case, we also set $\bar{x}_i = 0$.

2 - Marginalization. We sum x_i out. In this case, we also set $\bar{x}_i = 1$.

x_i determines if terms of f compatible with $x_i = 1$ are added to the sum.
$\bar{x}_i$ determines if terms of f compatible with $x_i = 0$ are added to the sum.

$Z = \sum_{\mathbf{x}} \Phi(\mathbf{x})$ is obtained by activating all the indicators in f.

For a set of states $\mathbf{E}$, the cost of computing $p(\mathbf{E}) = Z^{-1} \sum_{\mathbf{x} \in \mathbf{E}} \Phi(\mathbf{x})$ is linear in the size of the NP f.

But the size of f is exponential in the number of variables $x_1, \ldots, x_n$.

Solution:

use an arithmetic circuit of polynomial size which computes f exactly.

A sum product network is precisely such a circuit!

Sum Product Network (Arithmetic Circuit)

Rooted DAG whose leaves are $x_1, \ldots, x_n$ and $\bar{x}_1, \ldots, \bar{x}_n$ with internal sum and product nodes, where each edge (i, j) emanating from sum node i has a weight $w_{i,j} \geq 0$.

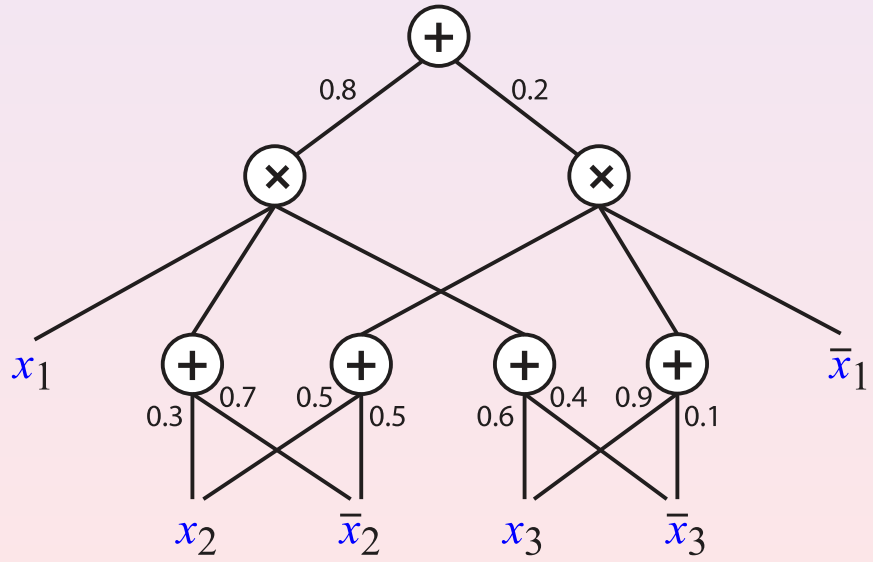
The value of a product node is the product of the value of its children.

The value of a sum node i is $\sum_{j \in \mathrm{Ch}(i)} w_{ij} v_j$, where $\mathrm{Ch}(j)$ are the children of node i and v_j is the value of node j.

The value of a SPN is the value of the root after a bottom up evaluation.

Layers of sum and product nodes alternate.

Figure source [Gens et al. 2012].

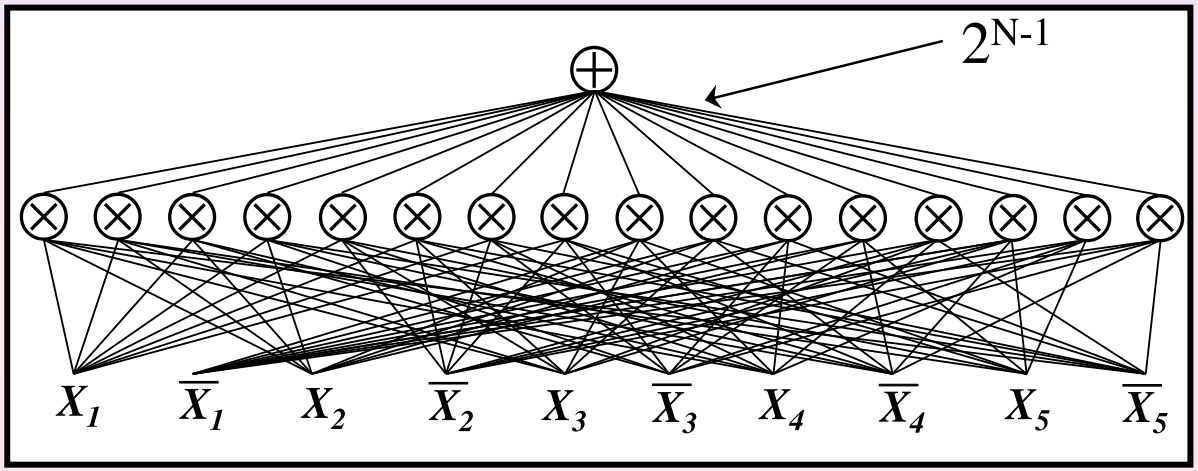


Deep SPNs vs. Shallow SPNs

Any distribution can be encoded using a shallow exponentially large SPN.

However, some distributions can be encoded using a compact deep SPN.

For example, uniform distribution over states with even number of 1's.



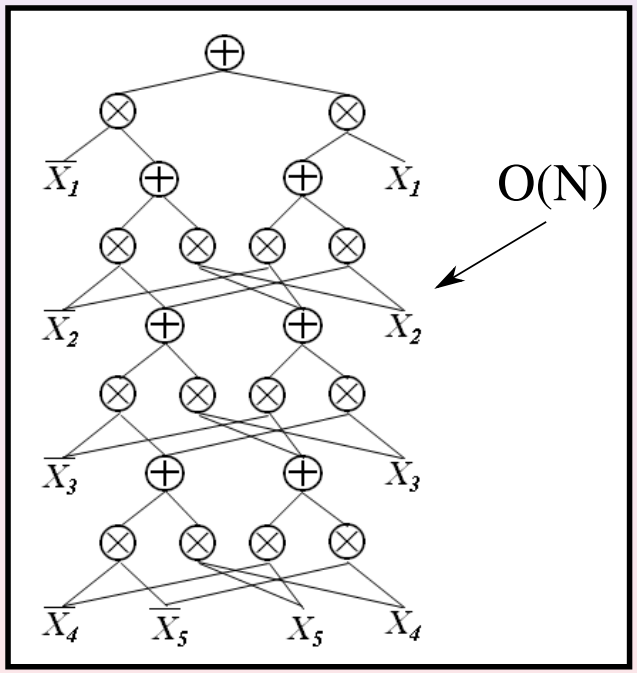


Figure source [Poon et al. 2011].

Valid SPNs I

Problem: some SPNs may not encode a valid network polynomial.

Let $S(\mathbf{E})$ denote the value of the SPN when all the indicators are activated according to the set of states $\mathbf{E}$.

Let $S(\mathbf{x})$ denote the value of the SPN when all the indicators are activated according to the individual state $\mathbf{x}$.

Then S is valid if $S(\mathbf{E}) = \sum_{\mathbf{x} \in \mathbf{E}} S(\mathbf{x})$ for all $\mathbf{E}$.

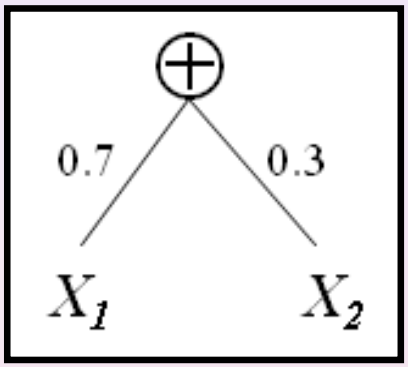
Definition 1: A SPN is complete if all children of a sum node cover the same set of variables.

Definition 2: A SPN is consistent if no variable appears negated in a child of a product node and non-negated in another.

Theorem: A SPN is valid if it is complete and consistent.

Valid SPNs II

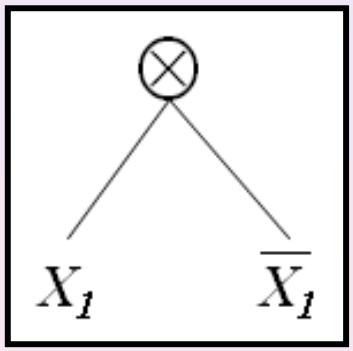
Complete: Under sum, children cover the same set of variables



Incomplete

$$S(\mathbf{E}) \leq \sum_{\mathbf{x} \in \mathbf{E}} S(\mathbf{x})$$

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



Inconsistent

$$S(\mathbf{E}) \geq \sum_{\mathbf{x} \in \mathbf{E}} S(\mathbf{x})$$

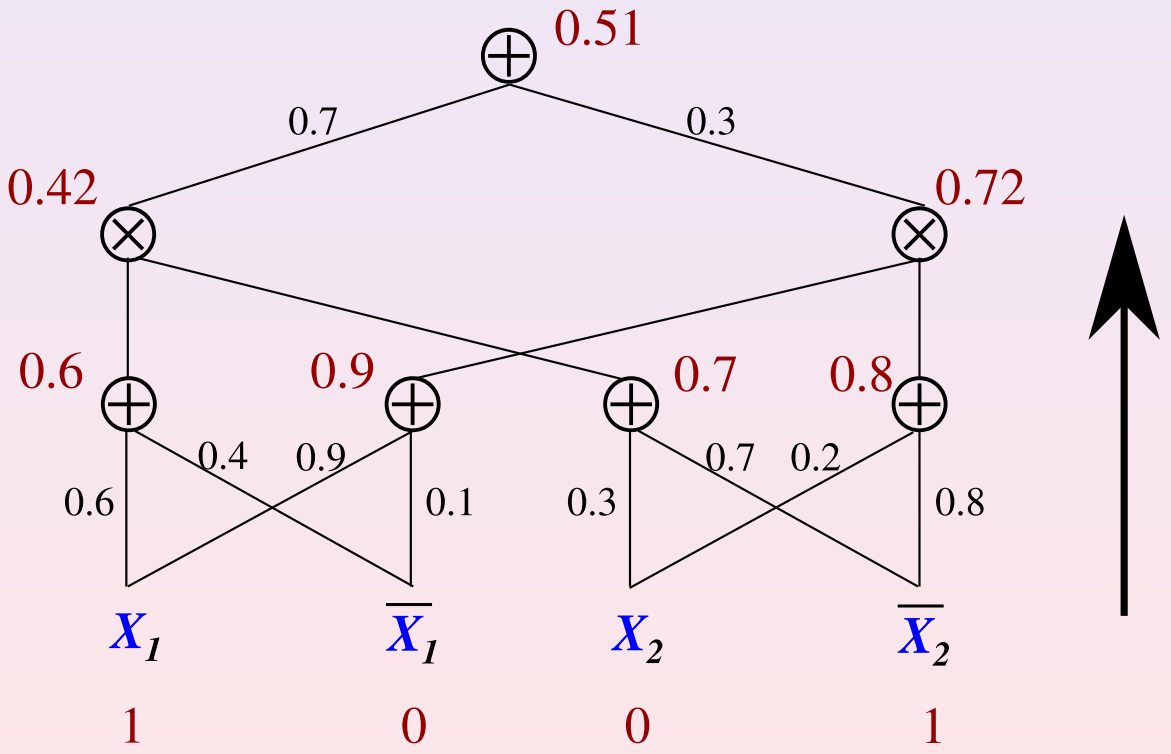
Slide source Poon.

Computing State Probabilities in SPNs

The probability of $\mathbf{x} = (x_1 = 1, x_2 = 0)$ is $P(\mathbf{x}) = S(\mathbf{x})/Z$.

$S(\mathbf{x})$ is obtained in a bottom up pass with $x_1 = 1$, $\bar{x}_1 = 0$, $x_2 = 0$, $\bar{x}_2 = 1$.

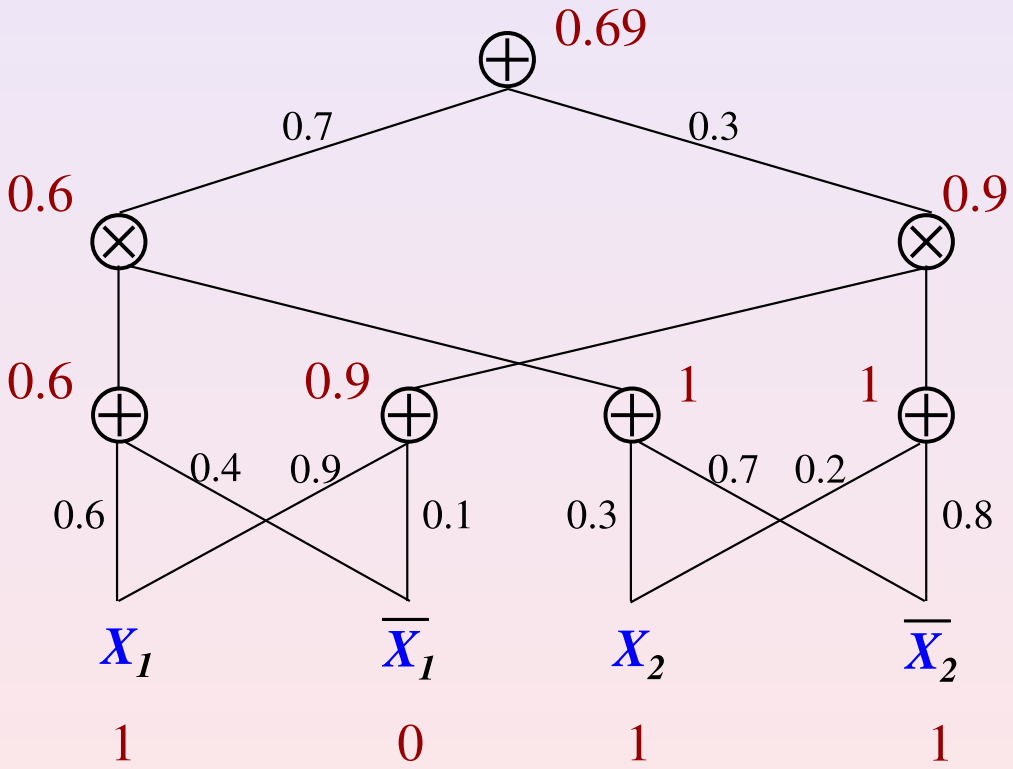
Z is obtained in a bottom up pass with $x_1 = 1$, $\bar{x}_1 = 1$, $x_2 = 1$, $\bar{x}_2 = 1$.



Computing Marginal Probabilities in SPNs

The probability of $\mathbf{E} = \{\mathbf{x} : x_1 = 1\}$ is $P(\mathbf{E}) = S(\mathbf{E})/Z$.

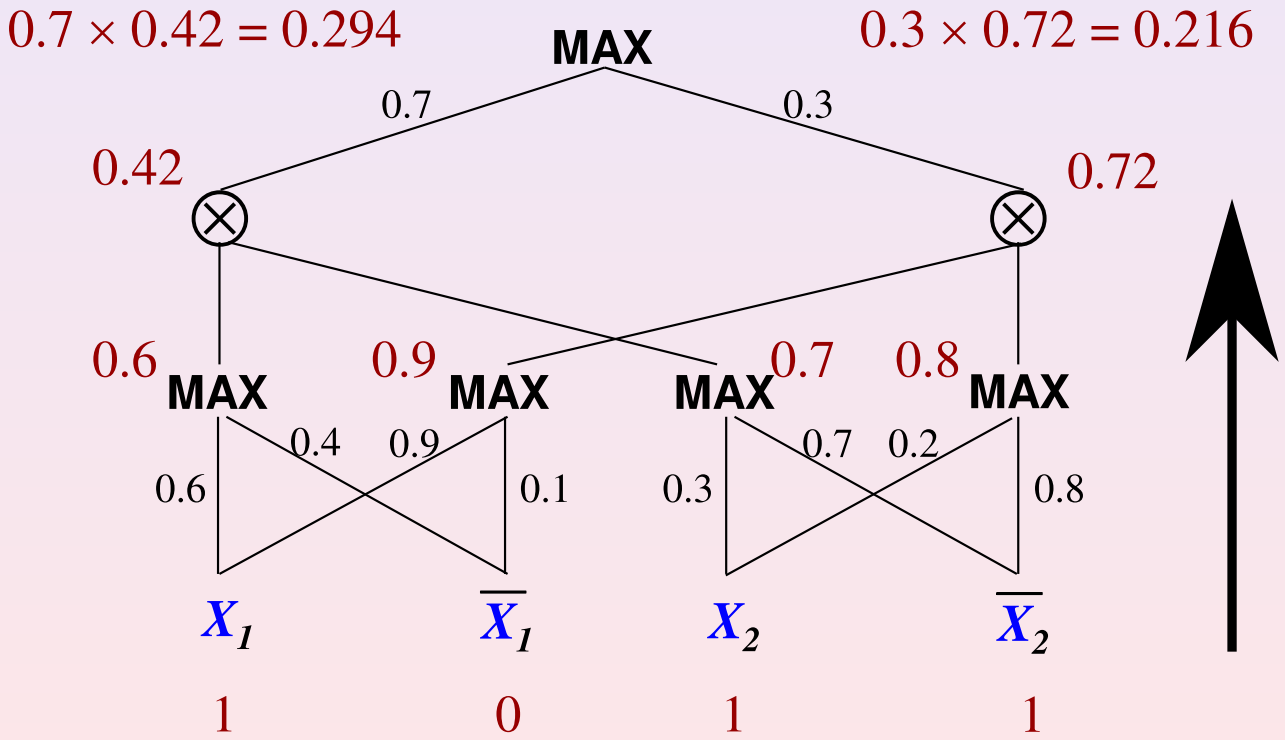
$S(\mathbf{E})$ is obtained in a bottom up pass with $x_1 = 1$, $\bar{x}_1 = 0$, $x_2 = 1$, $\bar{x}_2 = 1$.



Computing Most Probable Explanation (MPE) I

We know that $x_1 = 1$. What value is most likely for x_2?

Replace sum nodes by max nodes. Evaluate SPN in a bottom up pass with $x_1 = 1$, $\bar{x}_1 = 0$, $x_2 = 1$, $\bar{x}_2 = 1$.

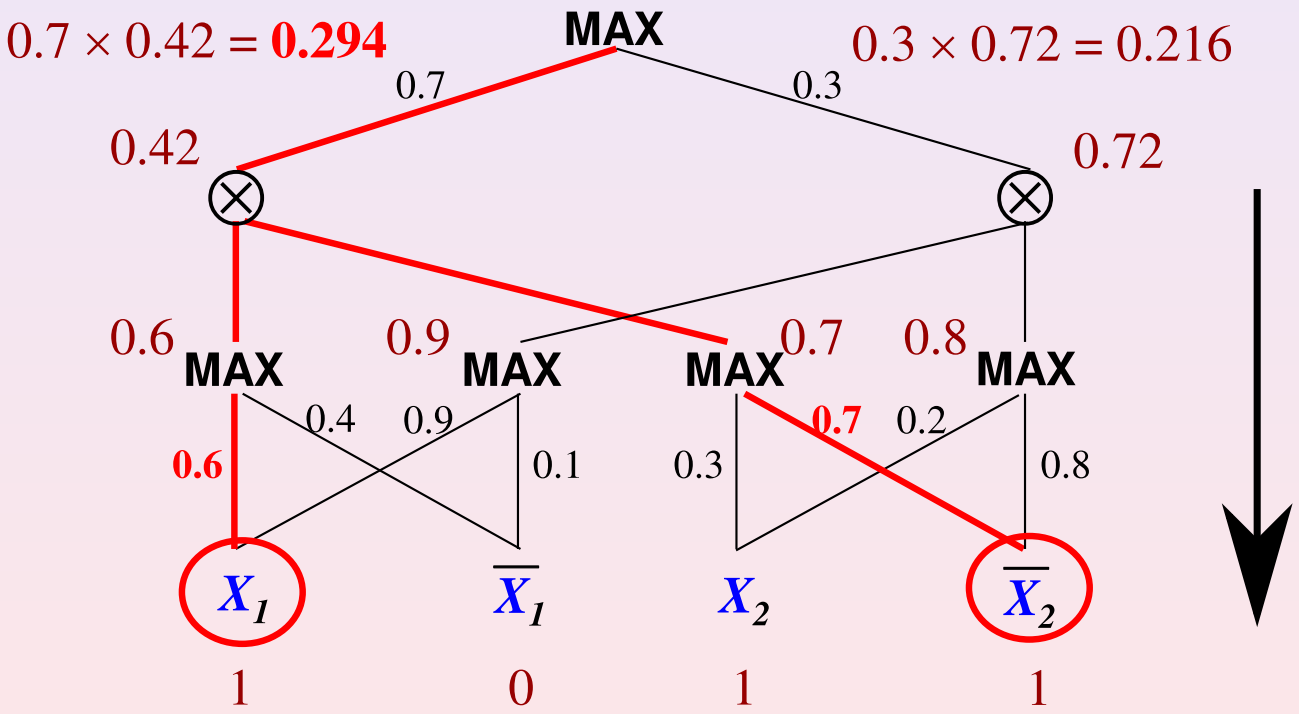


Computing Most Probable Explanation (MPE) II

In a top down pass, pick the child of a max node with highest value .

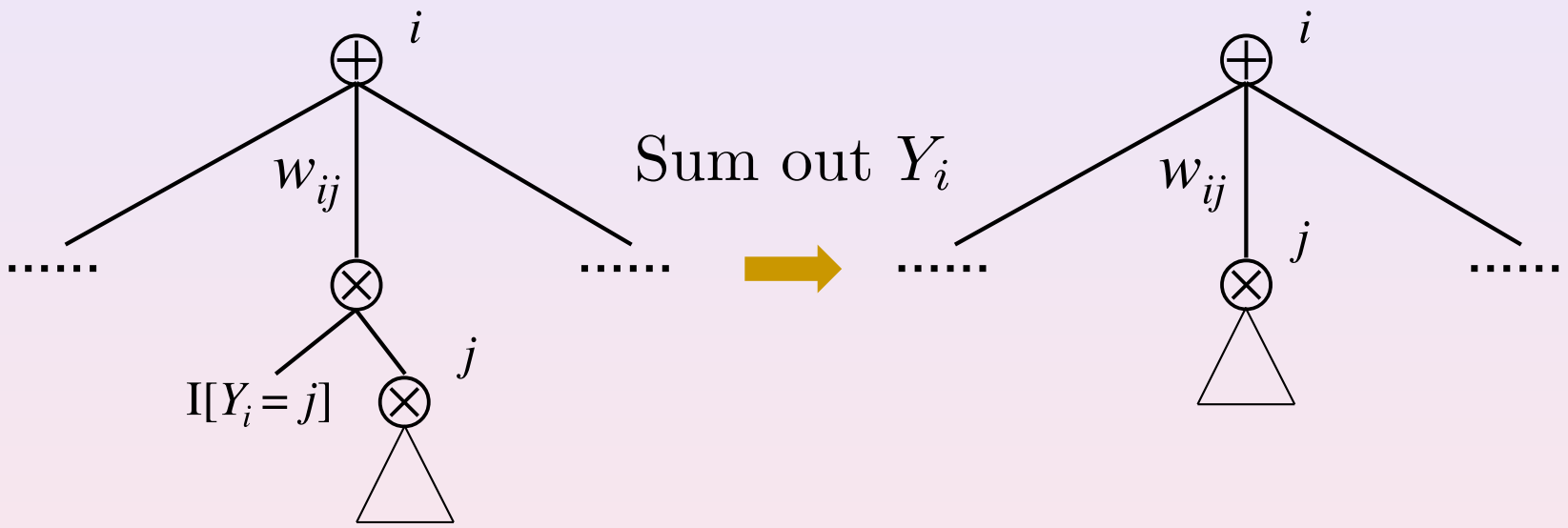
Pick always all the children of a product node.

The most likely value for x_2 is 0.



Semantics of SPNs

When for any sum node i we have that $\sum_{j \in \mathrm{Ch}(i)} w_{ij} = 1$, the SPN is normalized and i can be viewed as summing out a hidden variable $Y_i \in \mathrm{Ch}(i)$ which indicates which child is selected to compute v_i.



Therefore, each product node represents a different mixture component.

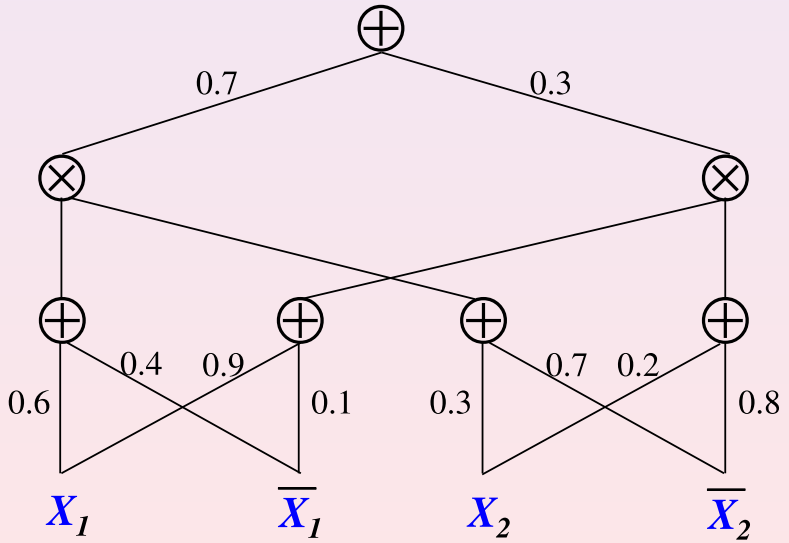
An SPN specifies a mixture model with exponentially many components, where subcomponents are composed and reused in larger ones.

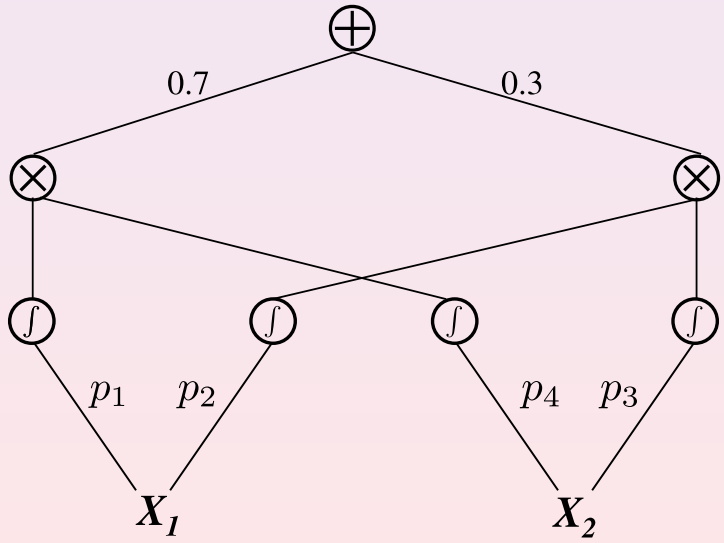
SPNs with Continuous Variables

SPNs can be generalized to continuous variables by viewing these as multinomial variables with an infinite number of values.

At a sum node i, the multinomial weighted sum of indicators $\sum_{j=1}^{m} p_i^j x_i^j$ becomes the integral $\int p(x)dx$. The value of node i is then $p(x)$ or 1.

When all the p are Gaussian the SPN compactly defines a very large mixture of Gaussians.





Marginals and Differentiation in SPNs

Let i be an arbitrary node in an SPN, $\mathrm{Pa}(i)$ be its parents and $S(\mathbf{x})$ and $S_i(\mathbf{x})$ be the value of the SPN and of node i on input $\mathbf{x}$.

Node i is product node:

$$\frac{\partial S(\mathbf{x})}{\partial S_i(\mathbf{x})} = \sum_{k \in \mathrm{Pa}} \left[\frac{\partial S(\mathbf{x})}{\partial S_k(\mathbf{x})} w_{ki} \right] .$$

Node i is sum node:

$$\frac{\partial S(\mathbf{x})}{\partial S_i(\mathbf{x})} = \sum_{k \in \mathrm{Pa}} \left[\frac{\partial S(\mathbf{x})}{\partial S_k(\mathbf{x})} \prod_{l \in \mathrm{Ch}_{-i}(k)} S_l(\mathbf{x}) \right] ,$$

where $\mathrm{Ch}_{-i}(k)$ are the children of node k excluding node i.
We also have that

$$\frac{\partial S(\mathbf{x})}{\partial w_{ij}} = \frac{\partial S(\mathbf{x})}{\partial S_i(\mathbf{x})} S_j(\mathbf{x}) .$$

We evaluate all the S_i's in an upward pass. We evaluate $\partial S(\mathbf{x})/\partial S_i(\mathbf{x})$ and $\partial S(\mathbf{x})/\partial w_{ij}$ in a downward pass.

The marginal of the latent variable Y_i is $p(Y_i = j|\mathbf{E}) \propto w_{ij} \partial S(\mathbf{E})/\partial S_i(\mathbf{E})$.

Learning SPNs

1 - Initialize the SPN using a dense valid arithmetic circuit.

2 - Learn the SPN weights using gradient descent or EM.

3 - Add some penalty to the weights so that they tend to be zero.

4 - Prune edges with zero weights at convergence.

Input: Set D of instances over variables X.
Output: An SPN with learned structure and parameters.
$S \leftarrow \text{GenerateDenseSPN}(X)$
$\text{InitializeWeights}(S)$
repeat
  for all $d \in D$ **do**
    $\text{UpdateWeights}(S, \text{Inference}(S, d))$
  end for
until convergence
$S \leftarrow \text{PruneZeroWeights}(S)$
return S

Main Difficulty: Gradient Diffusion

When learning deep SPNs, gradient descent and EM give poor results!

As more layers are added, the gradient signal rapidly vanishes.

EM is also affected. Its updates get smaller and smaller as we go deeper.

Solution: Hard EM + Online learning:

- Replace marginal inference with MPE inference.
- Maintain a count for each sum child.
- The M step increments the count of the wining child.
- The weights are obtained by renormalizing the counts.

The gradient diffusion problem is avoided because all updates, from the root to the inputs, are of unit size.

Hard EM makes possible to learn SPNs with more than 30 layers!

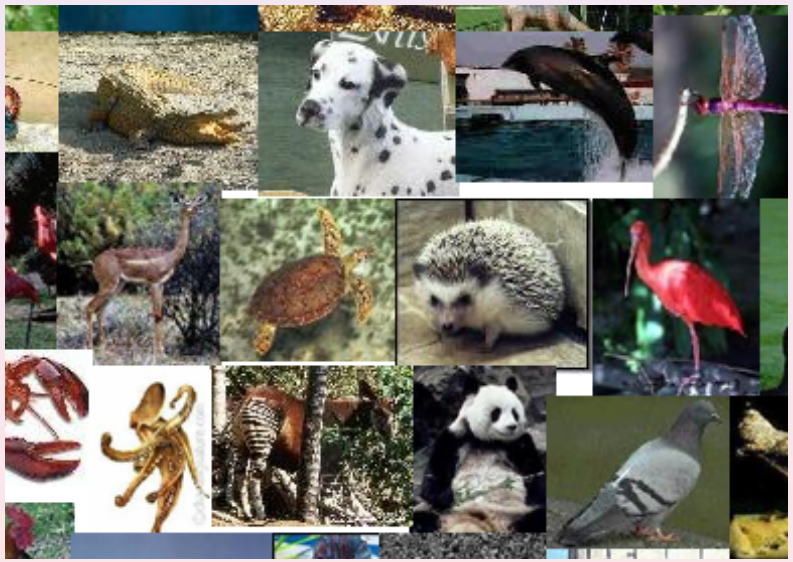
Experimental Evaluation: Image Completion

Half of an unseen image is occluded at test time.

The objective is to reconstruct the missing half given the observed half.

Very difficult task where detecting deep structure is key.

Dataset Caltech-101: 9146 images, 101 classes, 30 - 800 images per class.
Dataset Olivetti faces: 400 64x64 faces, 10 images per individual.



Benchmarks: DBNs, DBMs, PCA and 1-NN.

Initial SPN Architecture I

All rectangular regions selected, with the smallest ones being pixels.

Rectangular regions are decomposed into all possible two subregions.

However, in large regions only consider coarse region decompositions.

Decompositions at a coarse resolution of m-by-m for large regions.

Finer decompositions only inside each m-by-m block, with $m = 4$.

Very deep SPNs.

In general, $2(d-1)$ layers between the root and input for $d \times d$ images.

Initial SPN Architecture II

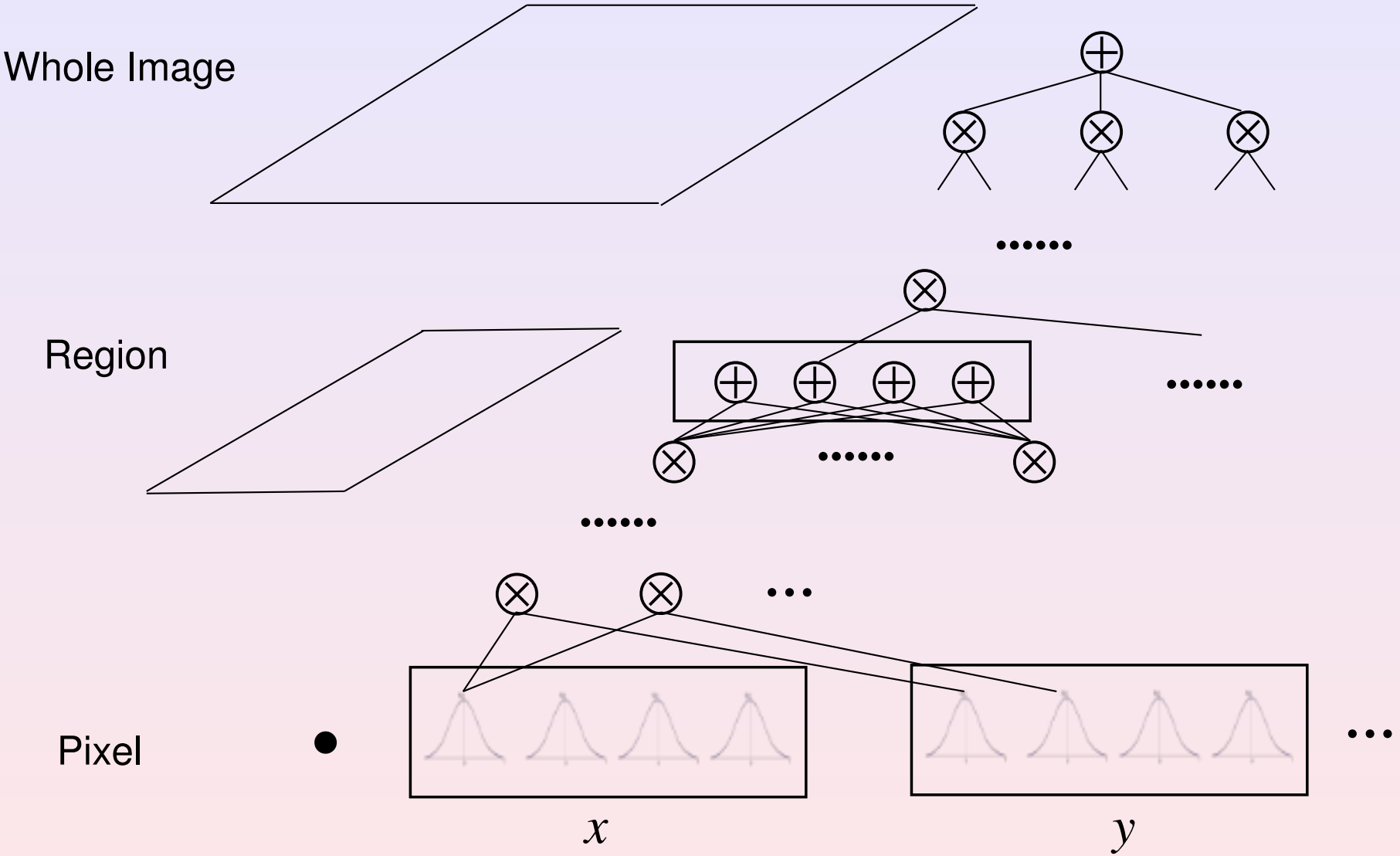


Figure source Poon.

Specific Details of the Learning Process Used

Mini-batches in online hard EM.

Best results: sums on the upward pass and maxes on the downward pass.

This means that the MPE value of each hidden variable is computed:

1 - Conditioning on the MPE values of the hidden variables above it.

2 - Summing out the hidden variables below it.

All weights initialized to zero.

Add-one smoothing when mapping the counts to the weights.

Gray-scale intensities normalized to have zero mean and unit variance.

Each pixel is modeled with a mixture of 4 unit-variance Gaussians whose means are placed at equal empirical quantiles.

Non-zero weights penalized with an L_0 prior with parameter 1.

Result on Caltech-101

Average test MSE for each method.

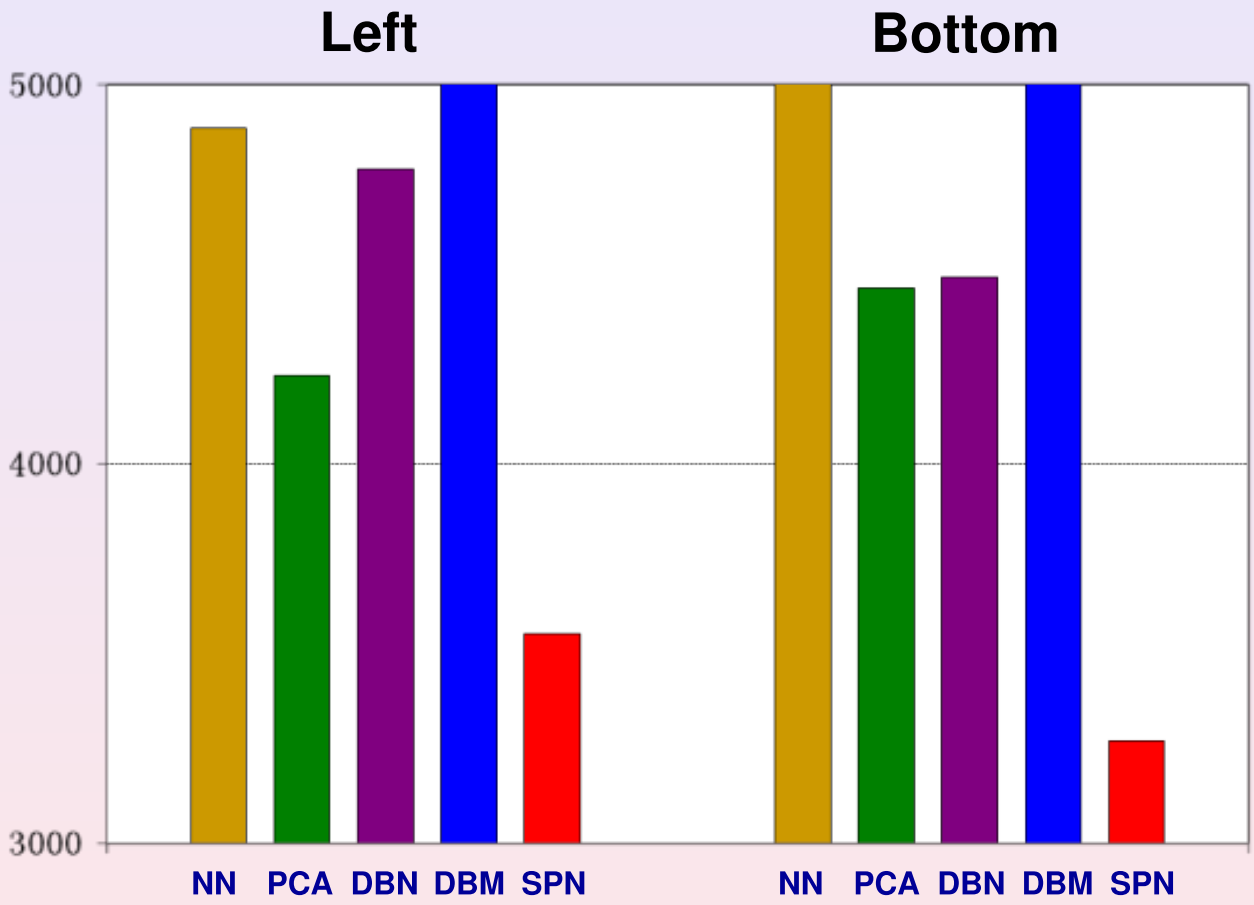


Figure source Poon.

Result on Olivetti Faces

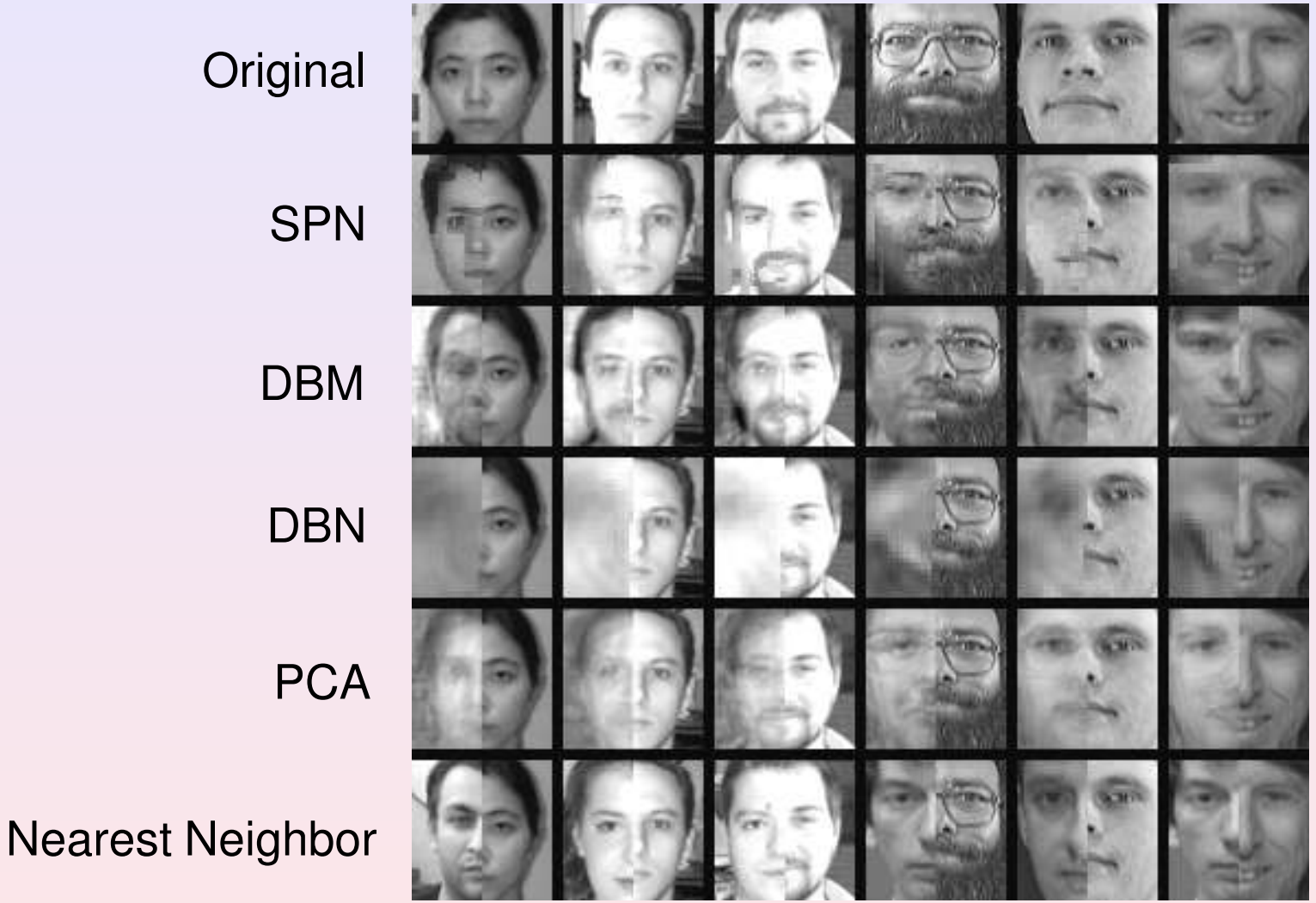


Figure source [Poon et al, 2011].

Advantages of SPNs vs. DBNs and DBMs

Exact inference: DBNs and DBMs require to use approximations (learning and prediction).

Not affected by gradient diffusion: Online hard EM allows to use very deep SPNs. Gradient diffusion limits DBNs and DBMs to a few layers.

Control of the learning process: SPN learning stops when the avg. log-likelihood does not improve. For DBNs and DBMs, the training iterations must be determined empirically.

Improved speed: SPNs are an order of magnitude faster in both learning and inference.

	SPNs	DBM / DBN
Learning	2-3 hours	Days
Inference	$<$ 1 second	Minutes or hours

Improved learning: SPNs seem to learn much more effectively.

Discriminative Learning of SPNs

We define $S[\mathbf{y}, \mathbf{h}|\mathbf{x}]$ as a SPN with disjoint sets of variables $\mathbf{H}$, $\mathbf{Y}$, and $\mathbf{X}$ (hidden, query, and given).

The setting of all $\mathbf{h}$ indicator functions to 1 is denoted as $S[\mathbf{y}, \mathbf{1}|\mathbf{x}]$.

Given an instance, we do not sum over $\mathbf{X}$, which is treated as a constant.

For the SPN to be valid

A variable x can be the child of any product node.

x can only be the child of a sum node that has scope outside of $\mathbf{Y}$ or $\mathbf{H}$.

Algorithm 1: LearnSPN

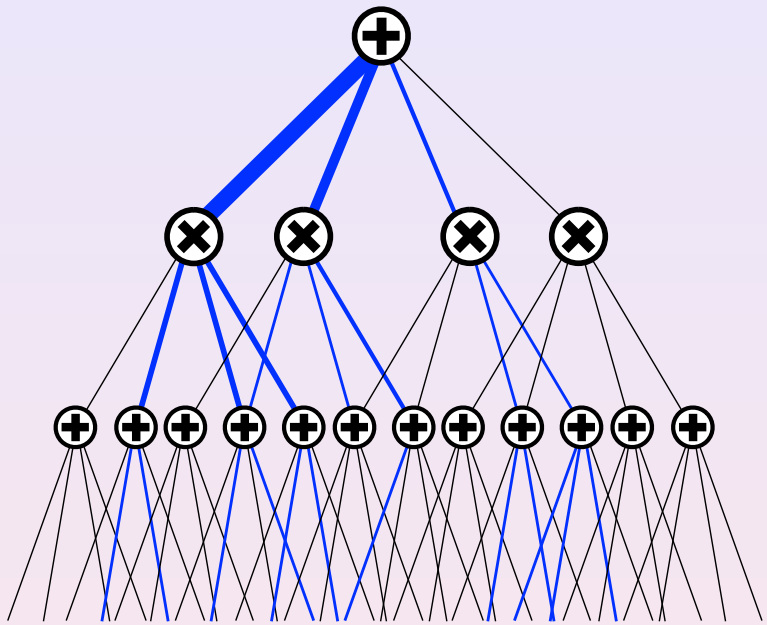
Input: Set D of instances over variables $\mathbf{X}$,$\mathbf{Y}$, a valid SPN S.
Output: An SPN with learned weights
repeat
  forall the $d \in D$ **do**
    UpdateWeights(S, Inference(S,$\mathbf{x_d}$,$\mathbf{y_d}$))
until *convergence or early stopping condition*;

Discriminative Training with Marginal Inference

$$\frac{\partial}{\partial w} \log p(\mathbf{y}|\mathbf{x}) = \frac{\partial}{\partial w} \log \sum_{\mathbf{h}} \Phi(\mathbf{Y} = \mathbf{y}, \mathbf{H} = \mathbf{h}|\mathbf{x}) - \frac{\partial}{\partial w} \log \sum_{\mathbf{y}', \mathbf{h}} \Phi(\mathbf{Y} = \mathbf{y}', \mathbf{H} = \mathbf{h}|\mathbf{x})$$

$$= \frac{1}{S[\mathbf{y}, \mathbf{1}|\mathbf{x}]} \frac{\partial S[\mathbf{y}, \mathbf{1}|\mathbf{x}]}{\partial w} - \frac{1}{S[\mathbf{1}, \mathbf{1}|\mathbf{x}]} \frac{\partial S[\mathbf{1}, \mathbf{1}|\mathbf{x}]}{\partial w}.$$

Gradient Diffusion



The partial derivatives of the SPN with respect to all weights can be computed with a bottom-up top-down evaluation of the SPN.

The gradient descent update is $\Delta w = \eta \frac{\partial}{\partial w} \log p(\mathbf{y}|\mathbf{x})$ with learning rate η.

Affected by gradient diffusion!

Figure source Gens.

Discriminative Training with MPE Inference I

Transform sum nodes into max nodes to go from $S[\mathbf{y}, \mathbf{h}|\mathbf{x}]$ to $M[\mathbf{y}, \mathbf{h}|\mathbf{x}]$.

$$\frac{\partial}{\partial w} \log \tilde{p}(\mathbf{y}|\mathbf{x}) = \frac{\partial}{\partial w} \log \max_{\mathbf{h}} \Phi(\mathbf{Y} = \mathbf{y}, \mathbf{H} = \mathbf{h}|\mathbf{x}) - \frac{\partial}{\partial w} \log \max_{\mathbf{y}', \mathbf{h}} \Phi(\mathbf{Y} = \mathbf{y}', \mathbf{H} = \mathbf{h}|\mathbf{x}) .$$

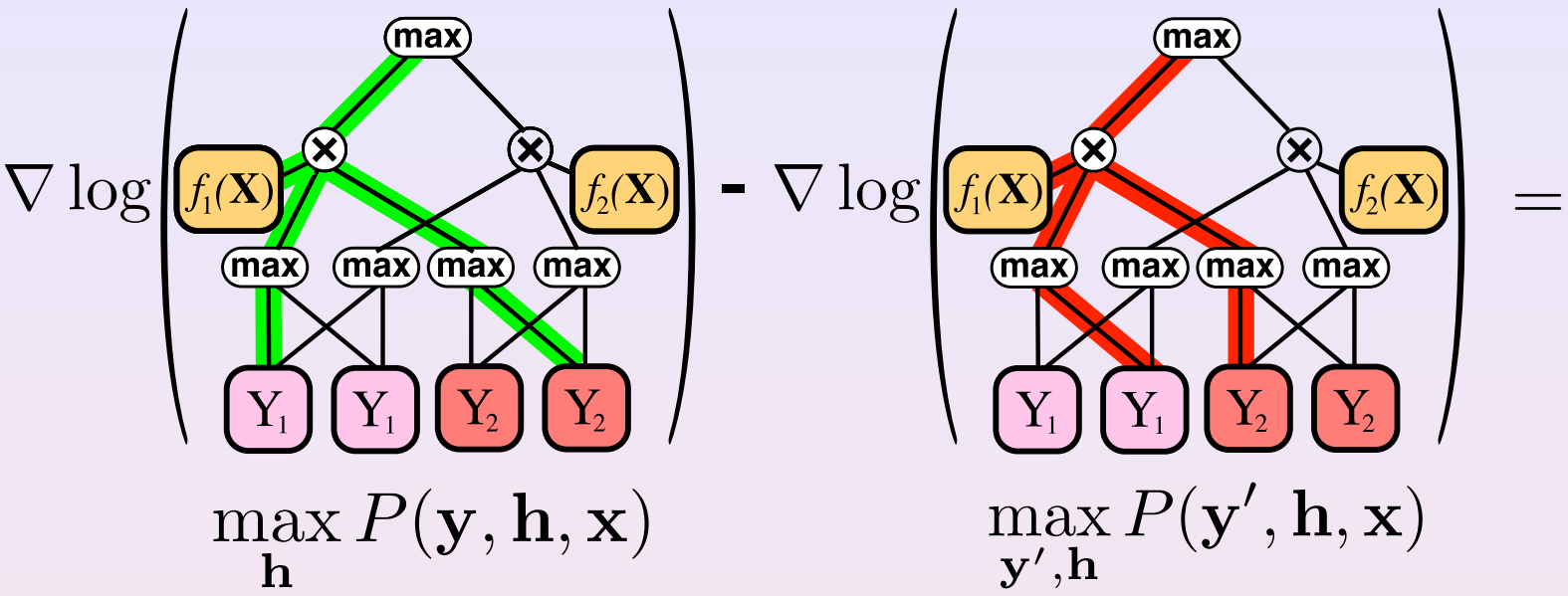
The two maximizations are computed by $M[\mathbf{y}, \mathbf{1}|\mathbf{x}]$ and $M[\mathbf{1}, \mathbf{1}|\mathbf{x}]$.

MPE inference yields a branching path through the SPN. Let $\mathbf{W}$ be the multiset of weights traversed by this path. Then $M = \prod_{w_i \in \mathbf{W}} w_i^{c_i}$, where c_i is the multiplicity of w_i in $\mathbf{W}$ and

$$\frac{\partial \log M}{\partial w_i} = \frac{1}{M} \frac{\partial M}{\partial w_i} = \frac{c_i w_i^{c_i - 1} \prod_{w_j \in \mathbf{W} \setminus \{w_i\}} w_j^{c_j}}{\prod_{w_i \in \mathbf{W}} w_i^{c_i}} = \frac{c_i}{w_i} .$$

The gradient of the conditional log likelihood with MPE inference is therefore $\Delta c_i / w_i$, where $\Delta c_i = c_i' - c_i''$ is the difference between the number of times w_i is traversed by the two MPE inference paths in $M[\mathbf{y}, \mathbf{1}|\mathbf{x}]$ and $M[\mathbf{1}, \mathbf{1}|\mathbf{x}]$, respectively. The Hard gradient update is $\Delta w_i = \eta \Delta c_i / w_i$.

Discriminative Training with MPE Inference II



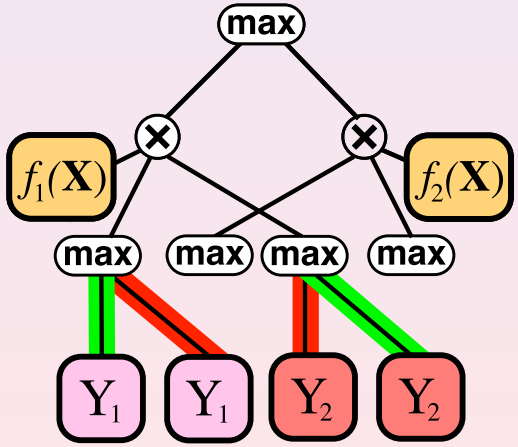
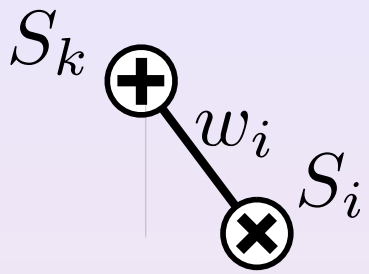


Figure source Gens.

Summary of Weight Updates



Update	Soft Inference	Hard Inference
Gen. GD	$\Delta w = \eta \frac{\partial S[\boldsymbol{x},\boldsymbol{y}]}{\partial w}$	$\Delta w_i = \eta \frac{c_i}{w_i}$
Gen. EM	$P(\mathrm{H_k = i}\|\mathbf{x},\mathbf{y}) \propto w_{ki} \frac{\partial S[\boldsymbol{x},\boldsymbol{y}]}{\partial S_k}$	$P(\mathrm{H_k = i}\|\mathbf{x},\mathbf{y}) = \begin{cases} 1 & : w_{ki} \in W \\ 0 & : \text{otherwise} \end{cases}$
Disc. GD	$\Delta w = \eta \left(\frac{1}{S[\boldsymbol{y},\mathbf{1}\|\boldsymbol{x}]} \frac{\partial S[\boldsymbol{y},\mathbf{1}\|\boldsymbol{x}]}{\partial w} - \frac{1}{S[\mathbf{1},\mathbf{1}\|\boldsymbol{x}]} \frac{\partial S[\mathbf{1},\mathbf{1}\|\boldsymbol{x}]}{\partial w} \right)$	$\Delta w_i = \eta \frac{\Delta c_i}{w_i}$

When working with the log of the weights the hard gradient update for discriminative training is $\Delta w_i' = \eta \Delta c_i$.

Experimental Evaluation

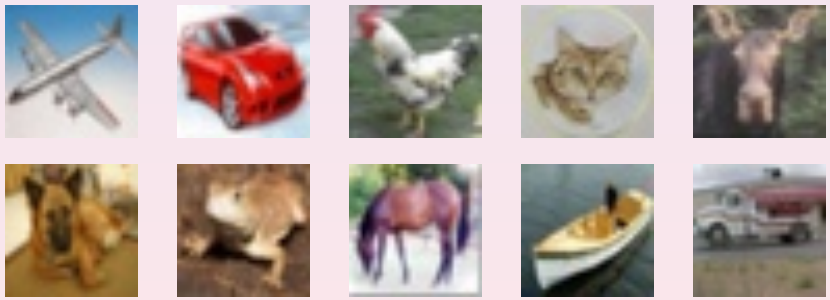
Two 10-class image classification tasks.

CIFAR-10: 32×32 pixels, $5 \cdot 10^4$ train, 10^4 test.

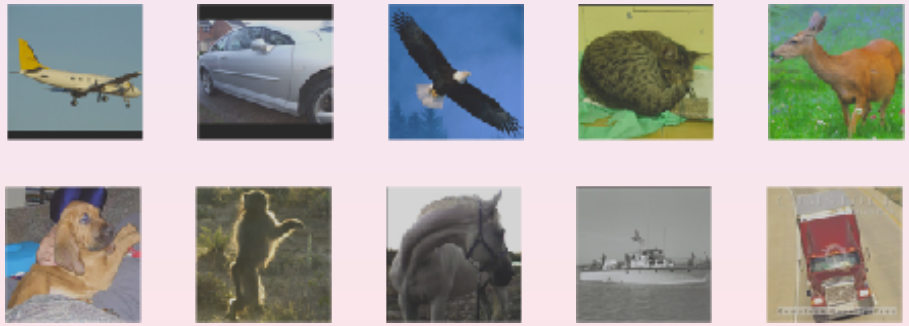
STL-10: 96×96 pixels, $5 \cdot 10^3$ train, $8 \cdot 10^3$ test (unlabeled data ignored).

Standard datasets for deep networks.

CIFAR-10



STL-10



Feature Extraction

Extract 6×6 image patches.

Preprocessing by ZCA whitening the patches, running k-means and then normalizing the dictionary to have zero mean and unit variance.

The dictionary is used to extract K features at each 6x6 patch.

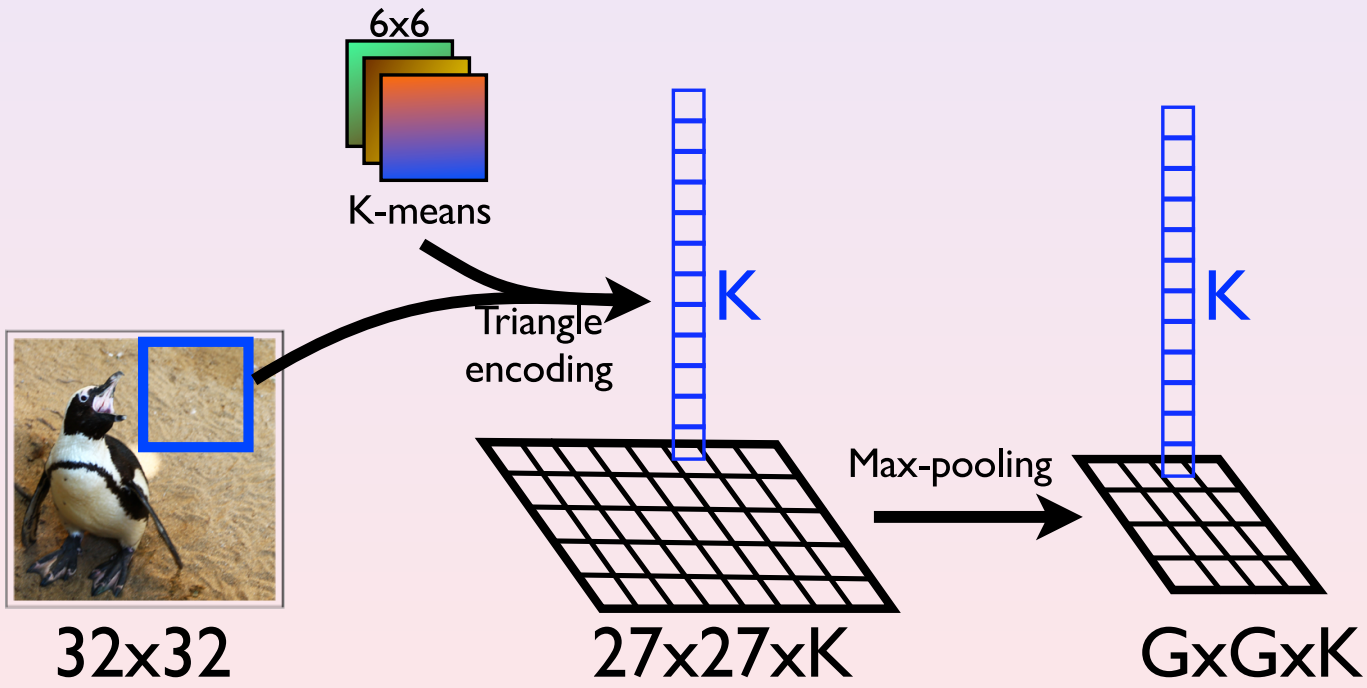


Figure source Gens.

SNP Architecture

An architecture which cannot be generatively trained as it violates consistency over $\mathbf{X}$.

C classes, P parts per class, and T mixture components per part.

Part: pattern of patch features that can occur anywhere in the image.

Networks learned with SGD regularized by early stopping.

Marginal inference for the root and MPE inference for the rest of the network worked best.

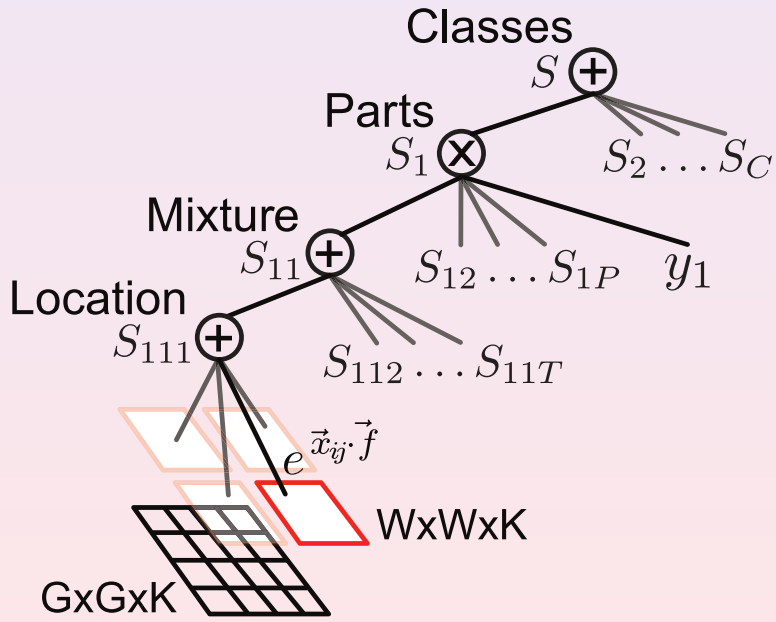


Figure source [Gens et al. 2012].

Results on CIFAR-10

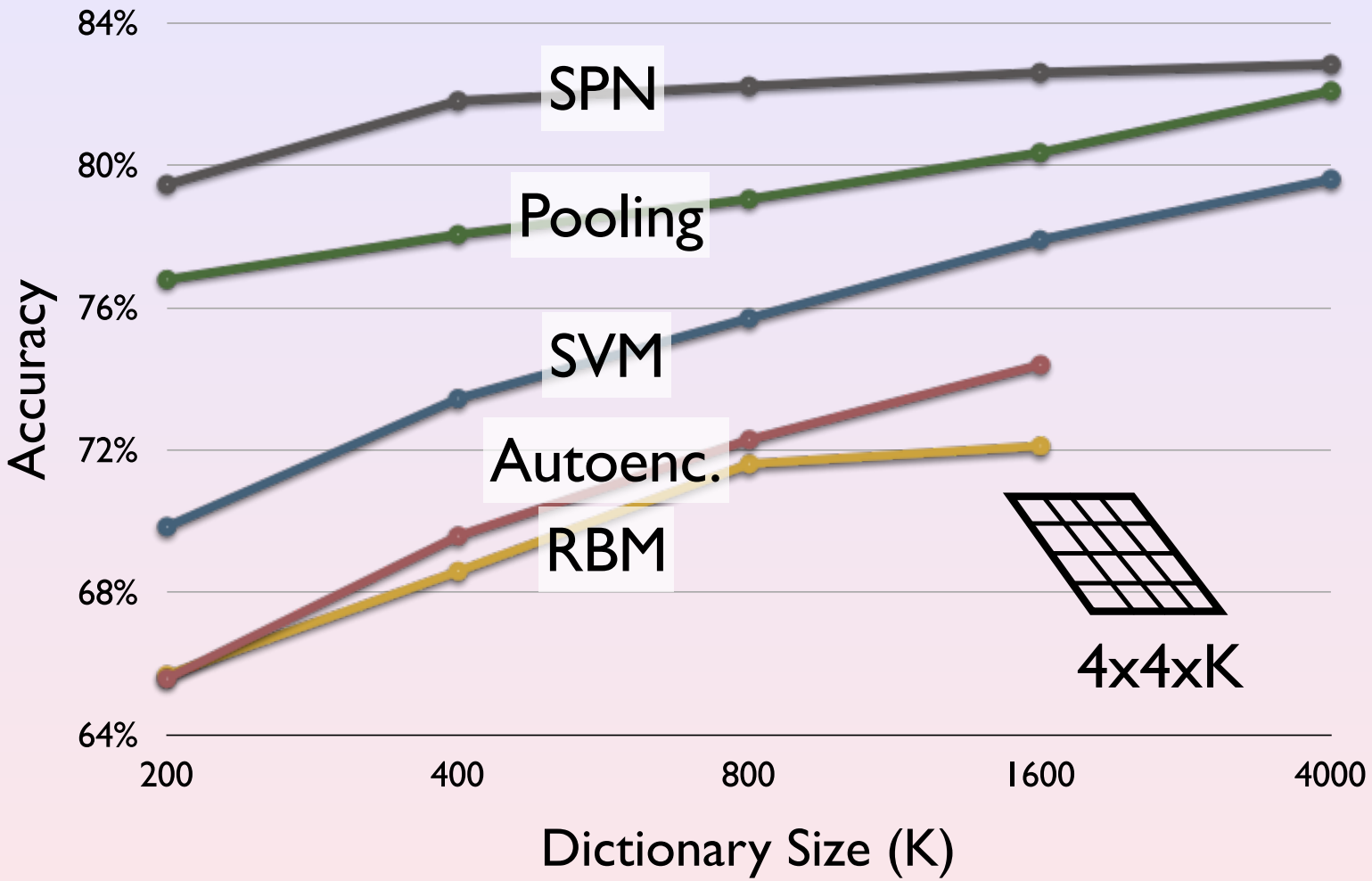


Figure source Gens.

Results on STL-10

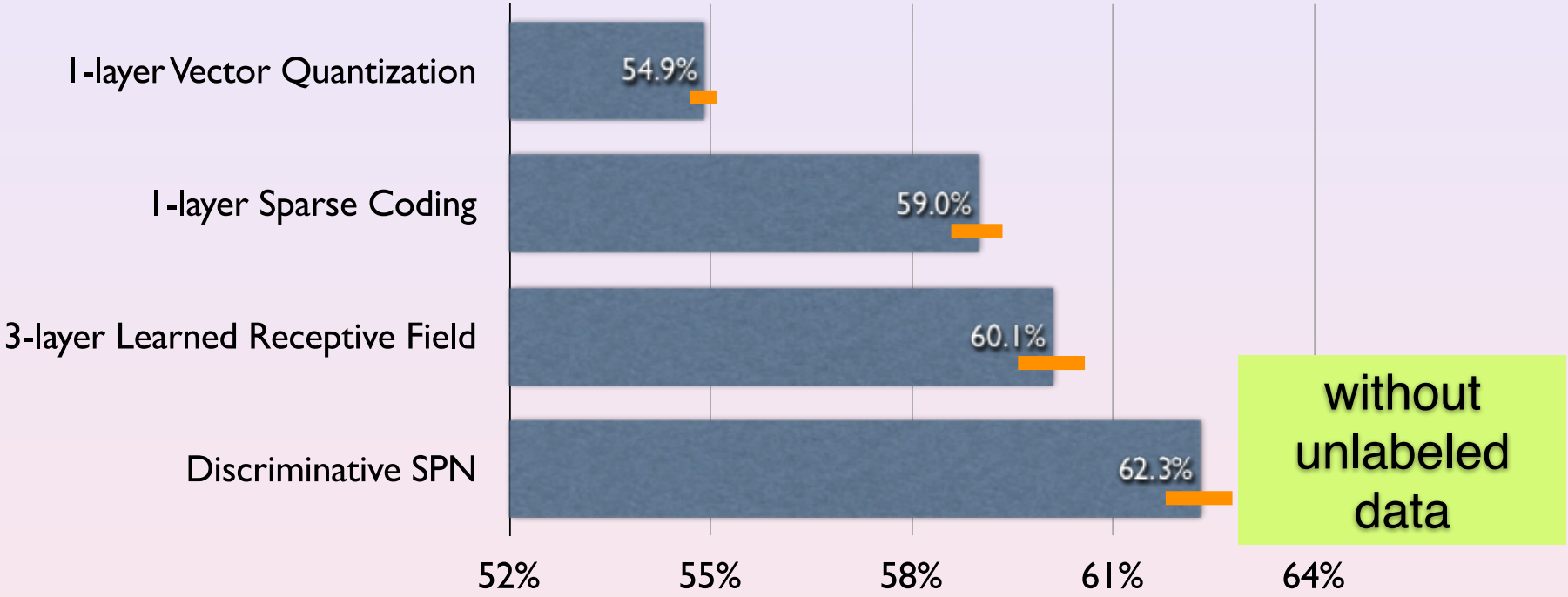


Figure source Gens.

Summary

Generative SPNs:

- Sum product networks...
  - are rooted DAG with sum and product nodes.
  - compactly represent a probability distribution.
  - contain many layers of hidden variables.
- Exact inference with cost linear in the size of the network.
- Deep learning implemented by online hard EM.
- SPNs can outperform state of the art on image completion.

Discriminative SPNs:

- Discriminative SPNs combine the advantages of
  - Tractable inference.
  - Deep architectures.
  - Discriminative learning.
- Hard gradient combats diffusion in deep models.
- SPNs can outperform the best existing methods on image classication.

References

- Darwiche, A. A differential approach to inference in Bayesian networks J. ACM, ACM, 2003, 50, 280-305
- Poon, H. and Domingos, P. Sum-Product Networks: A New Deep Architecture, UAI, 2011.
- Gens, R. and Domingos, P. Discriminative Learning of Sum-Product Networks NIPS, 2012, 3248-3256
- Delalleau, O. and Bengio, Y. Shallow vs. Deep Sum-Product Networks NIPS, 2011, 666-674
- Koller, D. and Friedman, N. Probabilistic Graphical Models: Principles and Techniques Mit Press, 2009
- Zhang, N. L. Hierarchical Latent Class Models for Cluster Analysis Journal of Machine Learning Research, 2004, 5, 697-723
- Chechetka, A. and Guestrin, C. Efficient Principled Learning of Thin Junction Trees NIPS, 2008