Sum-Product Networks: an alternative to Neural Networks?

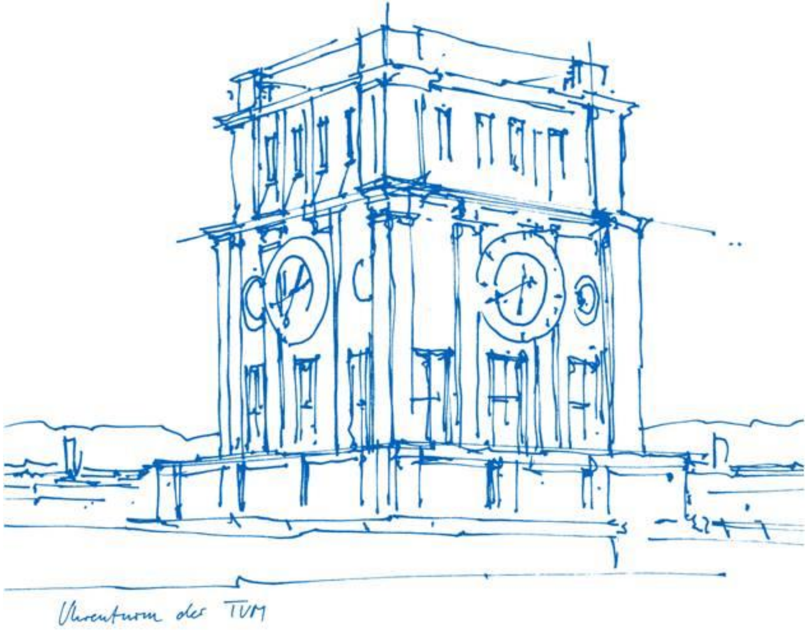
Author: Alex Pasquali

Supervisor: Yuesong Shen

Technische Universität München

Department of Informatics

Chair of Computer Vision and Artificial Intelligence



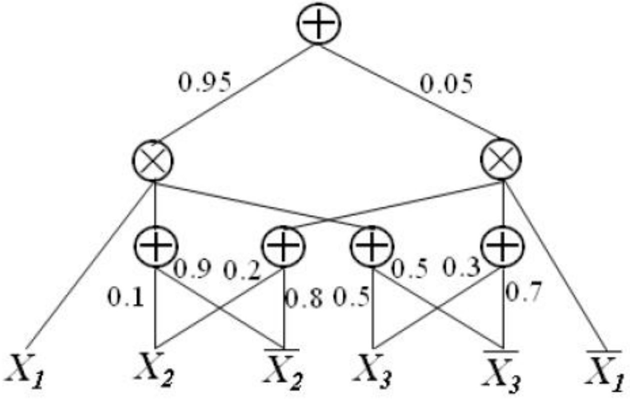
19th January 2022

Sum-Product Networks

What is all about?

Sum-product networks (SPNs) are a probabilistic framework that ***allows building tractable models*** *from data, making it possible to perform various inference tasks in* ***polynomial time***[1]*.*

They can be used alternatively to neural networks to solve similar problems.



Source: *Sum-Product Networks: A New Deep Architecture*, Poon et al., 2011

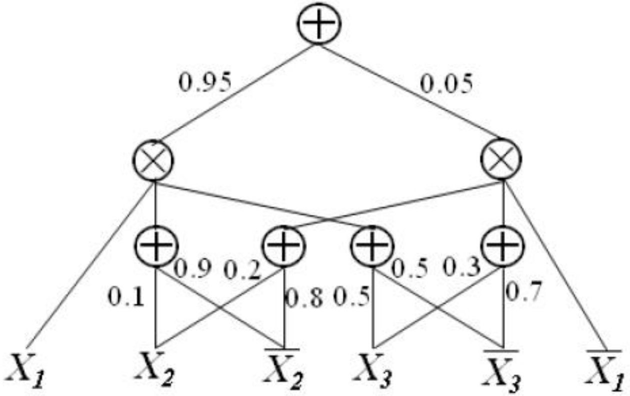
1. Probabilistic graphical models often require exponential time for exact inference.

Outline

- Sum-Product Networks:
  - Definition
  - Purpose
  - Properties
- Inference in SPNs:
  - Marginal probabilities
  - Posterior probabilities
  - Most Probable Explanation (MPE)
- Learning SPNs:
  - Generative parameter learning
  - Discriminative parameter learning
  - Structure learning
- Applications:
  - Image processing
  - NLP
- Sum-Product Networks vs. Neural Networks:
  - Main similarities and differences
  - Learning
  - Final considerations

Sum-Product Networks: definition

- Rooted DAGs
- **Leaves:** input variables indicators
- **Internal nodes:**
  - Sum nodes:
    - Have weights on their edges
    - Value: weighted average of the children’s values
  - Product nodes:
    - Do not have weights
    - Value: product of the children’s values
  - Sum and product nodes arranged in alternating layers



Source: *Sum-Product Networks: A New Deep Architecture*, Poon et al., 2011

Sum-Product Networks: purpose

Representing **probability distributions** as mixtures and factorizations.

- Mixtures: sum nodes
- Factorizations: product nodes

Graphical models represent distributions as a normalized product of factors: $P(\boldsymbol{X} = \boldsymbol{x}) = \frac{1}{Z}\Pi_k \phi_k\left(\boldsymbol{x}_{\{k\}}\right)$

Z is the **partition function**: $Z = \Sigma_{\boldsymbol{x} \in \boldsymbol{X}} \Pi_k \phi_k\left(\boldsymbol{x}_{\{k\}}\right)$ $\longrightarrow$ **Intractable** in many cases (sum of an exponential number of terms)

SPNs make Z tractable, reorganizing it into a computation involving a polynomial number of operations $\longrightarrow$ Setting all the indicators to 1 and performing an upward pass!

(the number of links in the network is forced to be polynomial)

Importance of being able to compute Z

- Goal of probabilistic models: compute $P(\boldsymbol{X} = \boldsymbol{x}) = \frac{1}{Z} \Pi_k \phi_k\left(\boldsymbol{x}_{\{k\}}\right)$
- Need to compute Z to accurately compute $P(\boldsymbol{X})$
- All marginals are sums of subsets of the terms present in Z
    - If Z can be computed efficiently, it means that those marginals can as well

- Probabilistic graphical models (e.g. Bayesian Networks) deal with this intractability by introducing variational (approximated) inference, but SPNs can compute $P(\boldsymbol{X})$ exactly and efficiently

Further details and definitions on SPNs

- **Scope**: the variables “seen by a node” ⇒ union of the scopes of its children
- **Completeness**: sum node complete ⇒ all children same scope
  - Complete SPN ⇒ all sum nodes are complete
- **Decomposability**: product node decomposable ⇒ children have disjoint scopes
  - Decomposable SPN ⇒ all product nodes are decomposable
- **Consistency**: consistent SPN ⇒ no variable is negated in a child of a product node and non-negated in another
- **Selectivity**: sum node selective ⇒ at most 1 child makes a positive contribution
  - Selective SPN ⇒ all sum nodes are selective

Usually SPNs assumed complete and decomposable

Sum-Product Networks: A New Deep Architecture, Poon & Domingos, 2011
Learning Selective Sum-Product Networks, Peharz, Gens & Domingos, 2014

Inference in SPNs

Marginal and posterior probabilities

- **Marginal:**
  - SPN S with root r, its value is $S(\boldsymbol{x}) = S_r(\boldsymbol{x}) = P(\boldsymbol{x})$
  - The **marginal probability** $S(\boldsymbol{x})$ can be evaluated with a single upward pass from the leaves to the root

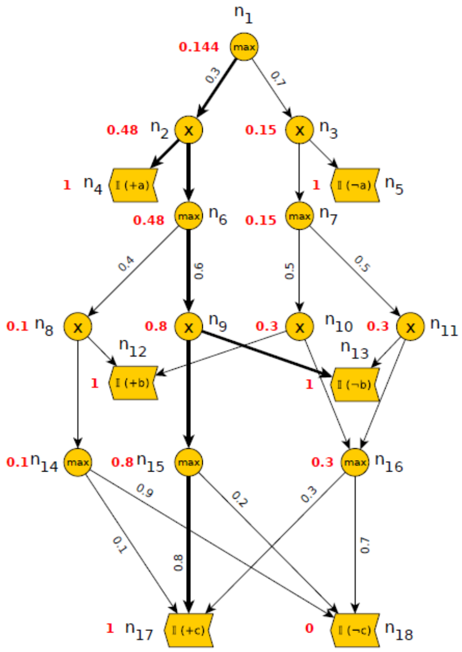
- **Posterior:**
  - $\boldsymbol{e} \in E$ is an **evidence**, $\boldsymbol{X} \cap \boldsymbol{E} = \emptyset$, $P(\boldsymbol{x}|\boldsymbol{e})$ is the **posterior probability**
  - $\boldsymbol{xe}$ denotes the composition of $\boldsymbol{x}$ and $\boldsymbol{e}$
  - $P(\boldsymbol{x}|\boldsymbol{e}) = S(\boldsymbol{xe})/S(\boldsymbol{e}) \rightarrow$ can be evaluated with at most 2 upward passes

Sum-Product Networks: A New Deep Architecture, Poon & Domingos, 2011
Sum-product networks: A survey, Parìs et al., 2020

Inference in SPNs

Most Probable Explanation (MPE): *Best Tree algorithm*

- **MPE:** configuration of $\boldsymbol{X}$ that **maximizes** the **posterior** probability
  - $MPE(\boldsymbol{e}) = \arg\max_{\boldsymbol{x}} P(\boldsymbol{x}|\boldsymbol{e}) = \arg\max_{\boldsymbol{x}} P(\boldsymbol{xe}) = \arg\max_{\boldsymbol{x}} S(\boldsymbol{xe})$
  - S selective ⇒ MPE found examining all induced trees where $\boldsymbol{e}$ stays fixed and $\boldsymbol{x}$ varies
  - Compare all trees at once by computing $S_i^{max}(\boldsymbol{e})$
    - n_i sum node: $S_i^{max}(\boldsymbol{e}) = \max_{j \in Ch(i)} w_{ij} \cdot S_j^{max}(\boldsymbol{e})$
    - Otherwise: $S_i^{max}(\boldsymbol{e}) = S_i(\boldsymbol{e})$
  - Then **backtracking** the “max” node from the root to the leaves



Source: *Sum-product networks: a survey*, Parìs et al., 2020

Sum-Product Networks: A New Deep Architecture, Poon & Domingos, 2011
Sum-product networks: A survey, Parìs et al., 2020
On the Latent Variable Interpretation in Sum-Product Networks, Peharz et al., 2017

Learning SPNs

- Structure and parameters can be learnt **jointly**:
  - Initial structure: complete and consistent
  - Repeat until convergence:
    - For each example in the dataset:
      - Run inference
      - Update weights: GD / EM
  - Prune edges with zero weight
  - Remove parentless nodes

Algorithm 1 LearnSPN

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

Source: *Sum-Product Networks: A New Deep Architecture*, Poon et al., 2011

Sum-Product Networks: A New Deep Architecture, Poon & Domingos, 2011

Learning SPNs

(Generative) Parameters Learning

Gradient descent (GD):

n_j child n_i: $\frac{\partial S(\boldsymbol{x})}{\partial w_{ij}} = S_j(\boldsymbol{x}) \cdot \frac{\partial S(\boldsymbol{x})}{\partial S_i(\boldsymbol{x})}$

- $\frac{\partial S(\boldsymbol{x})}{\partial S_i(\boldsymbol{x})} = \sum_{k \in Pa(i)} w_{ki} \cdot \frac{\partial S(\boldsymbol{x})}{\partial S_k(\boldsymbol{x})}$ if n_i product node
- $\frac{\partial S(\boldsymbol{x})}{\partial S_i(\boldsymbol{x})} = \sum_{k \in Pa(i)} \frac{\partial S(\boldsymbol{x})}{\partial S_k(\boldsymbol{x})} \prod_{l \in Ch_{-i}(k)} S_l(\boldsymbol{x})$ if n_i sum node

Update weights through gradient step and renormalize

Expectation Maximization (EM):

n_j sum node $\Rightarrow$ can be seen as summing out a hidden variable Y_i:

- Y_i's values are its children

- E step: inference → compute marginals of Y_i
- M step: weight update → add each Y_i's marginal to its sum from previous iteration (and normalize)

Sum-Product Networks: A New Deep Architecture, Poon & Domingos, 2011

Learning SPNs

(Generative) Parameters Learning

Problem:

Both *Gradient Descent* and *Expectation Maximization* **fail** when training deep SPNs due to the **vanishing gradient** phenomenon.

→

Solution: Hard EM

- Replace marginal inference with MPE inference
- Maintain a count for each sum child
- M step: increment count of winning child
- Weights obtained by normalizing the counts

Sum-Product Networks: A New Deep Architecture, Poon & Domingos, 2011

Learning SPNs

Discriminative Parameters Learning

Main idea:

- In machine learning we usually have some **observable input** variables $\boldsymbol{X}$
  - They are given, $P(\boldsymbol{x})$ **not of interest**

- Optimize $P(\boldsymbol{Y}|\boldsymbol{X})$ instead of $P(\boldsymbol{x})$

- $P(\boldsymbol{y}|\boldsymbol{x}) = \frac{\Phi(\boldsymbol{y}|\boldsymbol{x})}{\sum_{\boldsymbol{y}'} \Phi(\boldsymbol{y}'|\boldsymbol{x})} = \frac{\sum_{\boldsymbol{h}} \Phi(\boldsymbol{y},\boldsymbol{h}|\boldsymbol{x})}{\sum_{\boldsymbol{y}',\boldsymbol{h}} \Phi(\boldsymbol{y}',\boldsymbol{h}|\boldsymbol{x})} = \frac{S[\boldsymbol{y},\boldsymbol{1}|\boldsymbol{x}]}{S[\boldsymbol{1},\boldsymbol{1}|\boldsymbol{x}]}$ ➡ Computable in 2 upward passes

Φ is a non-normalized probability distribution

Discriminative Learning of Sum-Product Networks, Gens & Domingos, 2012

Learning SPNs

Discriminative Parameters Learning

Discriminative gradient descent:

$$\frac{\partial L(\boldsymbol{y}|\boldsymbol{x})}{\partial w_{ij}} = \frac{\partial \log P(\boldsymbol{y}|\boldsymbol{x})}{\partial w_{ij}}$$

$$= \frac{1}{S[\boldsymbol{y},\mathbf{1}|\boldsymbol{x}]} \frac{\partial S[\boldsymbol{y},\mathbf{1}|\boldsymbol{x}]}{\partial w_{ij}} - \frac{1}{S[\mathbf{1},\mathbf{1}|\boldsymbol{x}]} \frac{\partial S[\mathbf{1},\mathbf{1}|\boldsymbol{x}]}{\partial w_{ij}}$$

Partial derivatives can be computed as for (generative) gradient descent

Hard gradient descent:

- Based on discriminative GD but marginal inference replaced with MPE inference.
- $M[\boldsymbol{y},\boldsymbol{h}|\boldsymbol{x}]$ defined as SPN computing MPE

$$\frac{\partial \log \tilde{P}(\boldsymbol{y}|\boldsymbol{x})}{\partial w_{ij}} = \frac{\partial \log M[\boldsymbol{y},\mathbf{1}|\boldsymbol{x}]}{\partial w_{ij}} - \frac{\partial \log M[\mathbf{1},\mathbf{1}|\boldsymbol{x}]}{\partial w_{ij}}$$

$$= \frac{c_{ij}{}'}{w_{ij}} - \frac{c_{ij}{}''}{w_{ij}}$$

$c_{ij}{}'$ and $c_{ij}{}''$ are the number of times w_{ij} is traversed by the MPE inference paths in $M[\boldsymbol{y},\mathbf{1}|\boldsymbol{x}]$ and $M[\mathbf{1},\mathbf{1}|\boldsymbol{x}]$

Discriminative Learning of Sum-Product Networks, Gens & Domingos, 2012

Learning SPNs

Structure Learning: 1st example

Theory:

1. Select a **set of subsets** of variables
2. For each subet R, create k sum nodes $S_1^R, \dots, S_k^R$ and select a **set** of ways to decompose R into other subsets $R_1, \dots, R_l$
3. For each decomposition and $\forall R_i$ create a product node with parents S_j^R and children $S_1^{R_1}, \dots, S_l^{R_l}$

Requirement: only polynomial number of subsets and, for each of them, only polynomial number of decompositions

Practical example:

Image data:

- All rectangular regions are selected
- For each of them, select all possible ways to decompose it into 2 other rectangular regions

Sum-Product Networks: A New Deep Architecture, Poon & Domingos, 2011

Learning SPNs

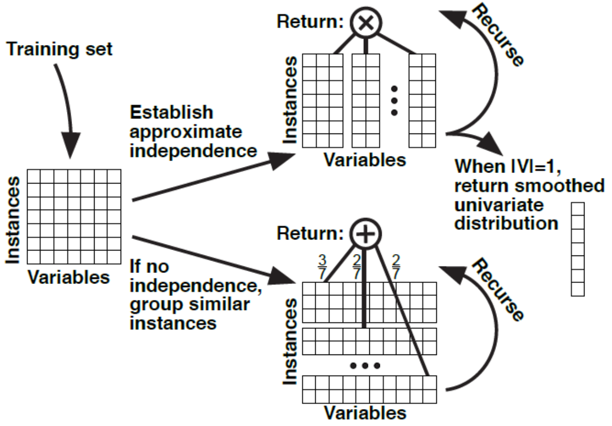
Structure Learning: LearnSPN

Dataset form: matrix (instances x variables)

Brief description: recursively split variables into independent subsets ("*chopping*") and then cluster similar instances ("*slicing*")

Every chopping creates a **product** node

Every slicing creates a **sum** node



Source: *Learning the Structure of Sum-Product Networks*, Gens et al., 2013

Learning the Structure of Sum-Product Networks, Gens & Domingos, 2013

Learning SPNs

Structure Learning: LearnSPN

If there is 1 variable

- Consider pairwise independence.
- Create graph containing a node for each variable and no links.
- Perform independence test[3] to each pair of variables and connect dependent ones.
- If the graph has only 1 component, the split fails.

Algorithm 1 LearnSPN(T, V)

input: set of instances T and set of variables V
output: an SPN representing a distribution over V learned from T
if $|V| = 1$ **then**
  return univariate distribution estimated from the variable's values in T
else
  partition V into approximately independent subsets V_j
  if success **then**
    return $\prod_j$ LearnSPN(T, V_j)
  else
    partition T into subsets of similar instances T_i
    return $\sum_i \frac{|T_i|}{|T|} \cdot$ LearnSPN(T_i, V)
  end if
end if

Create terminal node with univariate distribution using MLE

Recurse on the subsets of mutually independent variables

Using hard EM assuming all variables are independent conditioned on the partition

3. G-test of pairwise independence

Source: *Learning the Structure of Sum-Product Networks*, Gens et al., 2013

Applications

Computer Vision: image completion

- **Task:** image completion
- **Dataset:** Caltec-101 and Olivetti faces
- **Algorithm:** minibatch hard EM
  - **Best results:** sums in upward pass and maxes in downward pass
- **Metric:** MSE of completed pixels
- **Comparisons:** DBM, DBN, PCA, Nearest neighbor

- **SPN structure:** hand crafted as explained previously
  - **Multiple resolution levels**: consider coarser decompositions for larger regions and finer for smaller ones
    - Much faster learning
    - Little degradation in performances
  - **Final structure:** 36 layers ($2(d-1)$) for dxd images)

Sum-Product Networks: A New Deep Architecture, Poon & Domingos, 2011

Applications

Computer Vision: image completion

Comparisons:

- **DBM:** Highest MSE on the whole Caltec-101
- **DBN:** Results **not** directly comparable → images preprocessed differently
- **PCA:**
  - 100 principal components
  - Performs well in terms of MSE
  - Blurred completions → linear combination of images
- **Nearest neighbor:**
  - Good on test images similar to training ones
  - Usually poor completions (unless ↑)

Table 1: Mean squared errors on completed image pixels in the left or bottom half. NN is nearest neighbor.

LEFT	SPN	DBM	DBN	PCA	NN
Caltech (ALL)	3551	9043	4778	4234	4887
Face	1992	2998	4960	2851	2327
Helicopter	3284	5935	3353	4056	4265
Dolphin	2983	6898	4757	4232	4227
Olivetti	942	1866	2386	1076	1527

BOTTOM	SPN	DBM	DBN	PCA	NN
Caltech (ALL)	3270	9792	4492	4465	5505
Face	1828	2656	3447	1944	2575
Helicopter	2801	7325	4389	4432	7156
Dolphin	2300	7433	4514	4707	4673
Olivetti	918	2401	1931	1265	1793

Sum-Product Networks: A New Deep Architecture, Poon & Domingos, 2011

Applications

Computer Vision: image completion

Comparisons:

- **DBM:** Highest MSE on the whole Caltec-101
- **DBN:** Results **not** directly comparable → images preprocessed differently
- **PCA:**
  - 100 principal components
  - Performs well in terms of MSE
  - Blurred completions → linear combination of images
- **Nearest neighbor:**
  - Good on test images similar to training ones
  - Usually poor completions (unless ↑)

Original

SPN

DBM

DBN

PCA

Nearest neighbor

Sum-Product Networks: A New Deep Architecture, Poon & Domingos, 2011

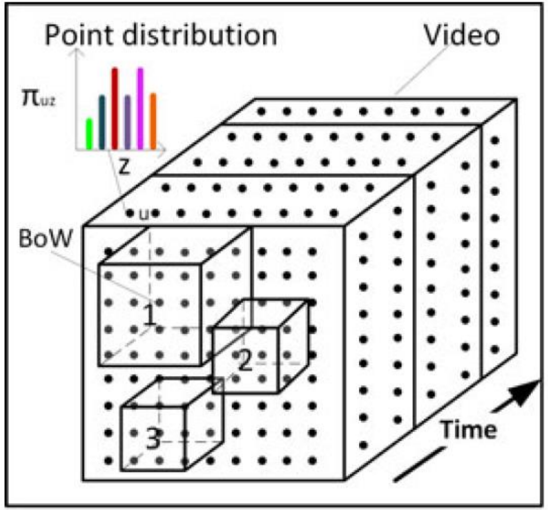
Applications

Activity recognition from videos

- *Visual word*: each meaningful part of an image
  - Extracted using a **neural network**

- Visual words placed on a 3D grid (width x height x time)

- Every window in the grid is modeled as a *Bag of Words* (BoW):
  - Histogram of occurrences of visual words in the window



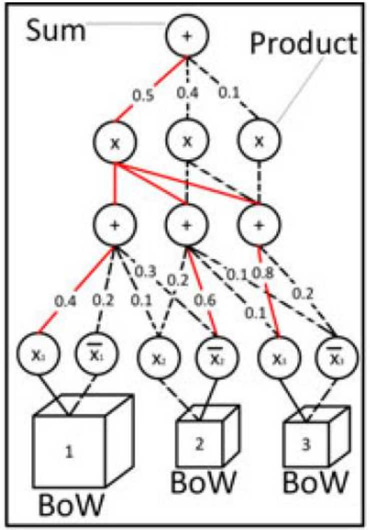
Source: *Sum Product Networks for Activity Recognition*, Amer & Todorovic, 2016

Sum Product Networks for Activity Recognition, Amer & Todorovic, 2016

Applications

Activity recognition from videos

- Each **BoW** treated as **random variable**:
  - 2 possible states: foreground and background
- **Product** nodes: **combinations** of sub-activities into more complex ones
- **Sum** nodes: **variations** of the same activity
- **Initial structure** of the SPN:
  - Almost completely connected graph
  - Gets pruned after parameters are learnt
- **Parameters learning**: iteratively
  - Learn SPN's weights from BoW's parameters
  - Learn BoW's parameters from SPN's weights



Source: *Sum Product Networks for Activity Recognition*, Amer & Todorovic, 2016

Sum Product Networks for Activity Recognition, Amer & Todorovic, 2016

Applications

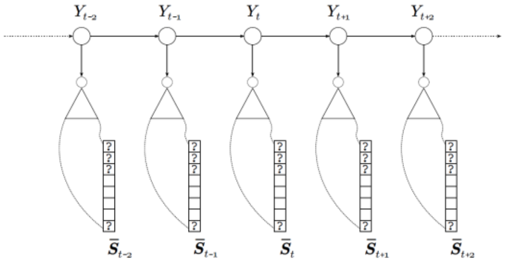
Natural Language Processing: bandwidth extension

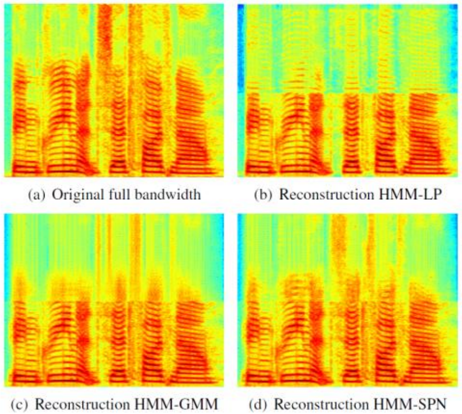
- **Task:** retrieve lost audio frequencies in telephone communications
- Real-time inference is crucial

- HMM to represent evolution of the spectrum
- Cluster the data (spectrum coefficients of frames)

- Train an SPN for each cluster:
  - Trained following *Poon & Domingos (2011)*

- Generate lost frequencies through MPE inference




(a) Original full bandwidth (b) Reconstruction HMM-LP
(c) Reconstruction HMM-GMM (d) Reconstruction HMM-SPN

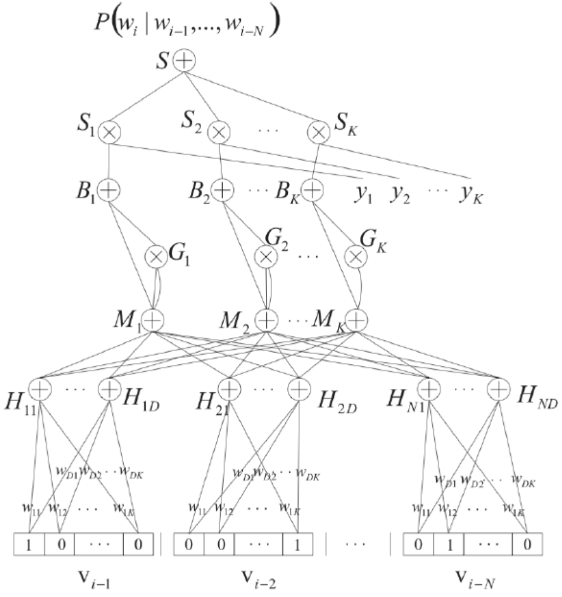
Modeling speech with sum-product networks: Application to bandwidth extension, Peharz et al., 2014

Applications

Natural Language Processing: language modelling

- **Task:** predict probability of next word in a sequence
  - $P(\boldsymbol{w}_{1:m}) \approx \prod_{k=1}^{m} P(w_k | \boldsymbol{w}_{k-n+1:k-1})$

- **Discriminative SPN** to model the above distribution:
  - Leaves: 1-hot vectors representing N previous words
  - Next layer compresses them into continuous values vectors
  - Penultimate layer connected to indicators representing the word we are predicting



Architecture of an SPN for language modelling

Language Modeling with Sum-Product Networks, Cheng et al., 2014

Applications

Natural Language Processing: language modelling

- Trained with **discriminative** gradient descent
- **Metric:** perplexity score (PPL)
  - $PPL = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1},\dots,w_{i-N})}}$
- SPNs outperformed all other models
  - They haven't been tested against transformers, which were developed later

Model	Individual PPL
TrainingSetFrequency	528.4
KN5 [3]	141.2
Log-bilinear model [4]	144.5
Feedforward neural network [5]	140.2
Syntactical neural network [8]	131.3
RNN [6]	124.7
LDA-augmented RNN [9]	113.7
SPN-3	**104.2**
SPN-4	**107.6**
SPN-4'	**100.0**

The suffix indicates the number of previous words considered

Source: *Language Modelling with Sum-Product Networks*, Cheng et al., 2014

Language Modeling with Sum-Product Networks, Cheng et al., 2014

Sum-Product Networks vs. Neural Networks

Main similarities and differences

SPNs can be see as a particular type of NNs:

- Flow of info from leaves to root
- They can be used alternatively for similar tasks

Main difference: *SPNs are grounded in a probabilistic interpretation, NNs do not have an obvious one.*

SPNs:

- Input: RVs (indicators)
- Output: probabilities
- Internal nodes: each computes a probability

NNs:

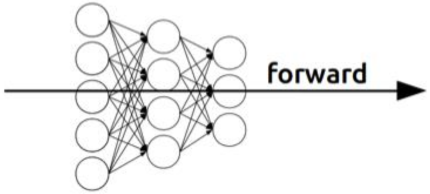
- Input: usually raw numbers
- Output: might be framed probabilistically (e.g. sigmoid for binary classification)
- Hidden units: lack direct probabilistic interpretation

Sum-Product Networks vs. Neural Networks

Inference

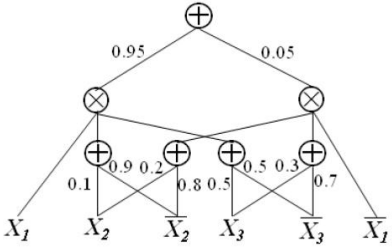
Neural Networks:

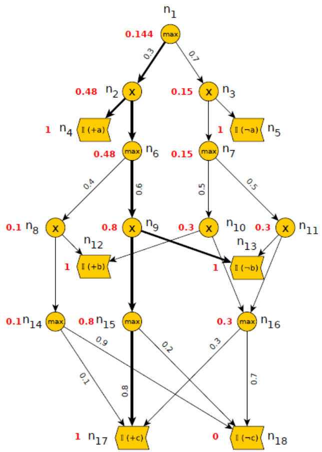
- Always one forward pass
- Need all inputs to be observable



Sum-Product Networks: depends on what it's computing

- Marginals $P(X_i)$: one upward pass
- Posteriors $P(\boldsymbol{X}|\boldsymbol{E})$: two upward passes
- MPE: requires backtracking
- Can perform inference with partial info





Sum-Product Networks vs. Neural Networks

Learning

Parameters learning:
- **NNs**:
  - based on discriminative gradient descent
- **SPNs**:
  - Discriminative gradient descent
    - Hard gradient descent
  - Generative approaches
    - Generative gradient descent
    - Expectation Maximization
    - Hard EM

Structure learning:
- **NNs**:
  - Designed by hand
  - Trial-and-error approach (very inefficient)
- **SPNs**:
  - Can learn structure from data
    - Either starting from a dense architecture and then pruning (e.g. *Poon & Domingos 2011*)
    - Or learning form zero (e.g. *LearnSPN*)

Sum-Product Networks vs. Neural Networks

Some final considerations

- Despite many advantages of SPNs over NNs, the latter tend to be superior in many tasks:
  - CIFAR-10 classification: SPNs got 84% accuracy in 2012, but NNs are over 99%[1]
  - However RAT-SPNs[2] reached comparable performance on MNIST
    - RAT-SPNs: randomized architecture and trained with NN-style methods
- SPNs are a younger class of models → less well-established learning practices and fewer heuristics
- SPNs and NNs could be used complementary:
  - e.g. Wang & Wang[3] used a convolutional NN to isolate parts of images before using SPNs for activity recognition

1. *https://paperswithcode.com/sota/image-classification-on-cifar-10*
2. *Probabilistic Deep Learning using Sum-Product Networks*, Peharz et al., 2018
3. *Hierarchical Spatial Sum-Product Networks for Action Recognition in Still Images*, Wang & Wang, 2015

Bibliography

- *Sum-Product Networks: A New Deep Architecture*, Poon & Domingos, 2011
- *Learning Selective Sum-Product Networks*, Peharz, Gens & Domingos, 2014
- *Sum-product networks: A survey*, Parìs et al., 2020
- *On the Latent Variable Interpretation in Sum-Product Networks*, Peharz et al., 2017
- *Discriminative Learning of Sum-Product Networks*, Gens & Domingos, 2012
- *Learning the Structure of Sum-Product Networks*, Gens & Domingos, 2013
- *Modeling speech with sum-product networks: Application to bandwidth extension*, Peharz et al., 2014
- *Language Modeling with Sum-Product Networks*, Cheng et al., 2014
- *Probabilistic Deep Learning using Sum-Product Networks*, Peharz et al., 2018
- *Hierarchical Spatial Sum-Product Networks for Action Recognition in Still Images*, Wang & Wang, 2015
- Caltec-101 dataset http://www.vision.caltech.edu/Image_Datasets/Caltech101/
- Olivetti faces: *Parameterisation of a stochastic model for human face identification*, Samaria & Harter1994
- MNIST: *The MNIST database of handwritten digits*, LeCun & Cortes, 2005

Sum-Product Networks: an alternative to Neural Networks?

Thank you for your attention

Alex Pasquali

Technische Universität München

alex.pasquali@tum.de