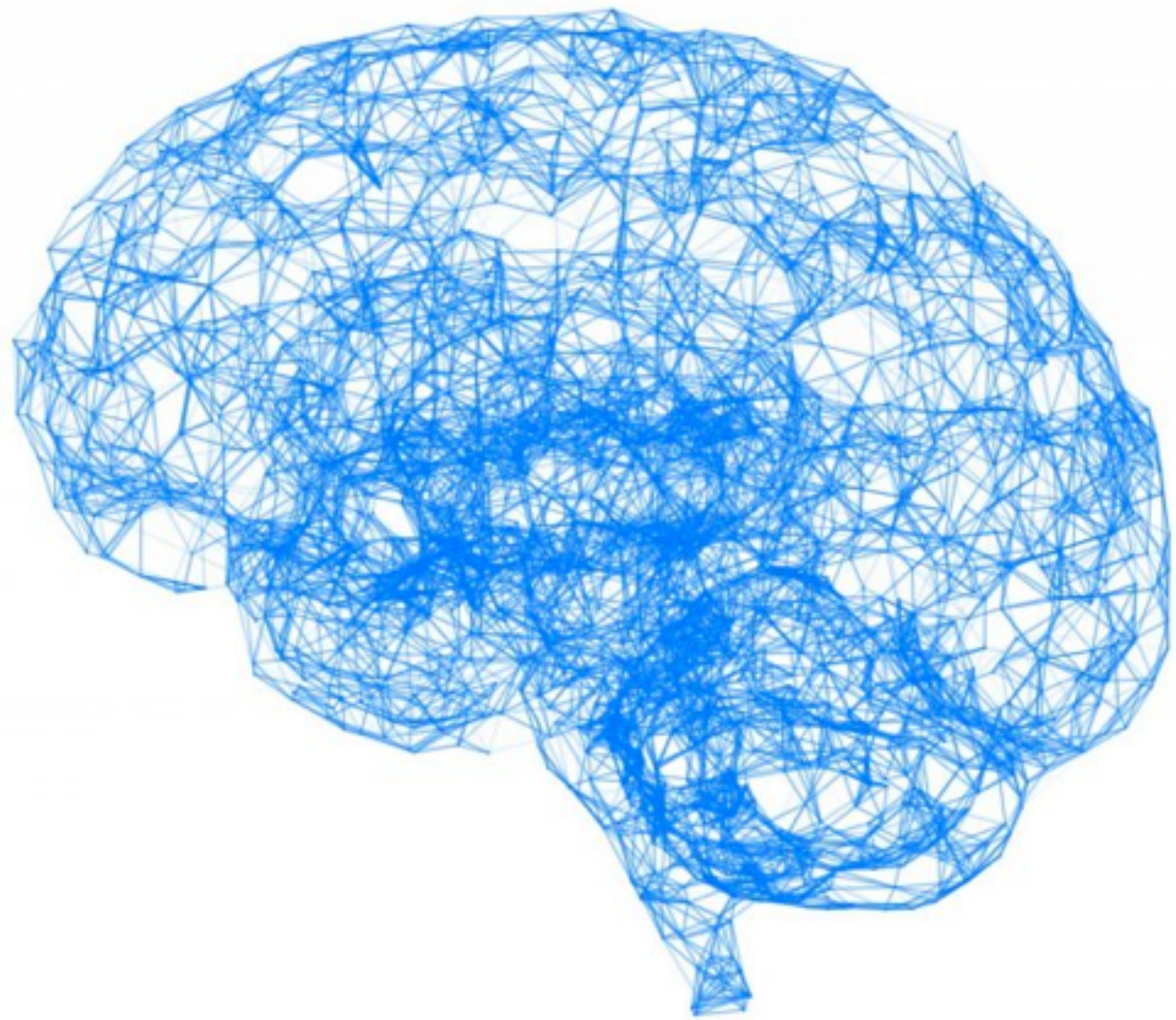
Sum-Product Networks

Jhonatan Oliveira

Research Tasks A, C, and D

Outline

- **The Paper**
    - Motivation
    - Background
    - Critic
- **The Problem**
    - Parallel computation
    - Possible issues
- **The Comparison**
    - Implementation
    - CPU x GPU
- **Conclusion**



The Paper

Sum-Product Networks: A New Deep Architecture

Hoifung Poon and **Pedro Domingos**
Computer Science & Engineering
University of Washington
Seattle, WA 98195, USA
{hoifung,pedrod}@cs.washington.edu

Abstract

The key limiting factor in graphical model inference and learning is the complexity of the partition function. We thus ask the question: what are general conditions under which the partition function is tractable? The answer leads to a new kind of deep architecture, which we call *sum-*

the uniform distribution over vectors with an even number of 1's.) Second, inference is still exponential in the worst case. Third, the sample size required for accurate learning is worst-case exponential in scope size. Fourth, because learning requires inference as a subroutine, it can take exponential time even with fixed scopes (unless the partition function is a known constant, which requires restricting the potentials to be conditional probabilities).

Motivation

- Best Paper Award in UAI'11
- Cited 162 times
- Significant improvement using deep learning

Sum-Product Networks: A New Deep Architecture

Hoifung Poon and **Pedro Domingos**
Computer Science & Engineering
University of Washington
Seattle, WA 98195, USA
{hoifung,pedrod}@cs.washington.edu

Abstract

The key limiting factor in graphical model inference and learning is the complexity of the partition function. We thus ask the question: what are general conditions under which the partition function is tractable? The answer leads to a new kind of deep architecture, which we call *sum-*

the uniform distribution over vectors with an even number of 1's.) Second, inference is still exponential in the worst case. Third, the sample size required for accurate learning is worst-case exponential in scope size. Fourth, because learning requires inference as a subroutine, it can take exponential time even with fixed scopes (unless the partition function is a known constant, which requires restricting the potentials to be conditional probabilities).

Motivation

Sum-Product Networks: A New Deep Architecture

Hoifung Poon and **Pedro Domingos**
Computer Science & Engineering
University of Washington
Seattle, WA 98195, USA
{hoifung,pedrod}@cs.washington.edu

Abstract

The key limiting factor in graphical model inference and learning is the complexity of the partition function. We thus ask the question: what are general conditions under which the partition function is tractable? The answer leads to a new kind of deep architecture, which we call *sum-*

the uniform distribution over vectors with an even number of 1's.) Second, inference is still exponential in the worst case. Third, the sample size required for accurate learning is worst-case exponential in scope size. Fourth, because learning requires inference as a subroutine, it can take exponential time even with fixed scopes (unless the partition function is a known constant, which requires restricting the potentials to be conditional probabilities).

Shallow vs. Deep Sum-Product Networks

Olivier Delalleau
Department of Computer Science and Operation Research
Université de Montréal
delallea@iro.umontreal.ca

Yoshua Bengio
Department of Computer Science and Operation Research
Université de Montréal
yoshua.bengio@umontreal.ca

Abstract

We investigate the representational power of sum-product networks (computation networks analogous to neural networks, but whose individual units compute either products or weighted sums), through a theoretical analysis that compares deep (multiple hidden layers) vs. shallow (one hidden layer) architectures. We prove there exist families of functions that can be represented much more efficiently with a deep network than with a shallow one, i.e. with substantially fewer hidden units. Such results were not available until now, and contribute to motivate recent research involving learning of deep sum-product networks, and more generally motivate research in Deep Learning.

Introduction and prior work

any learning algorithms are based on searching a family of functions so as to identify one memb said family which minimizes a training criterion. The choice of this family of functions and ho embers of that family are parameterized can be a crucial one. Although there is no universall timal choice of parameterization or family of functions (or "architecture"), as demonstrated b no-free-lunch results [37], it may be the case that some architectures are appropriate (or ina

Motivation

Resource Constrains

- Deep learning:
  - processing
  - data management
- Deep architecture
  - storage
  - inference

Background

Join Probability Distribution

- A multivariate function over a finite set of variables
- Assigns a real number between 0 and 1 to each configuration (combination of variable's values) of the variables
- Summing all assigned real numbers yields 1

X1	X2	P(X1,X2)
0	0	0.1
0	1	0.3
1	0	0.5
1	1	0.1

Background

A	B	P(X1,X2)	
0	0	0.1	$\lambda_{A_0}\lambda_{B_0}$
0	1	0.3	$\lambda_{A_0}\lambda_{B_1}$
1	0	0.5	$\lambda_{A_1}\lambda_{B_0}$
1	1	0.1	$\lambda_{A_1}\lambda_{B_1}$

Network Polynomial

$$f = 0.1\lambda_{A_0}\lambda_{B_0} + 0.3\lambda_{A_0}\lambda_{B_1} + 0.5\lambda_{A_1}\lambda_{B_0} + 0.1\lambda_{A_1}\lambda_{B_1}$$

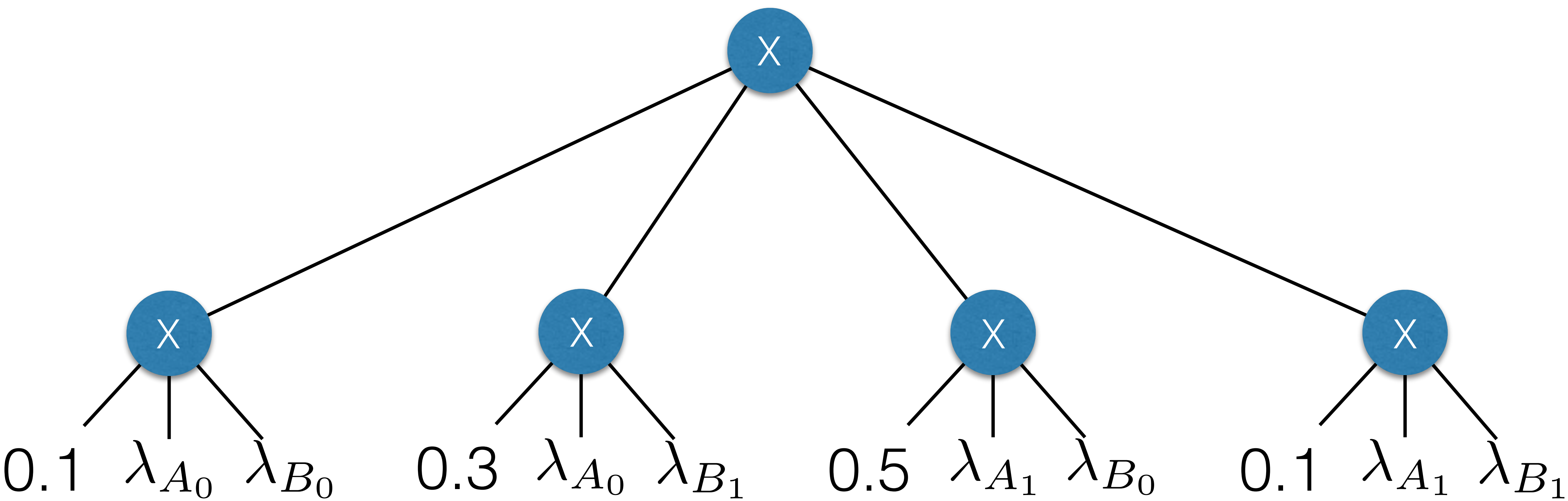
Background

A	B	P(X1,X2)	
0	0	0.1	$\lambda_{A_0}\lambda_{B_0}$
0	1	0.3	$\lambda_{A_0}\lambda_{B_1}$
~~1~~	~~0~~	~~0.5~~	$\cancel{\lambda_{A_1}\lambda_{B_0}}$
~~1~~	~~1~~	~~0.1~~	$\cancel{\lambda_{A_1}\lambda_{B_1}}$

Network Polynomial

$$f = 0.1\lambda_{A_0}\lambda_{B_0} + 0.3\lambda_{A_0}\lambda_{B_1} + \cancel{0.5\lambda_{A_1}\lambda_{B_0}} + \cancel{0.1\lambda_{A_1}\lambda_{B_1}}$$

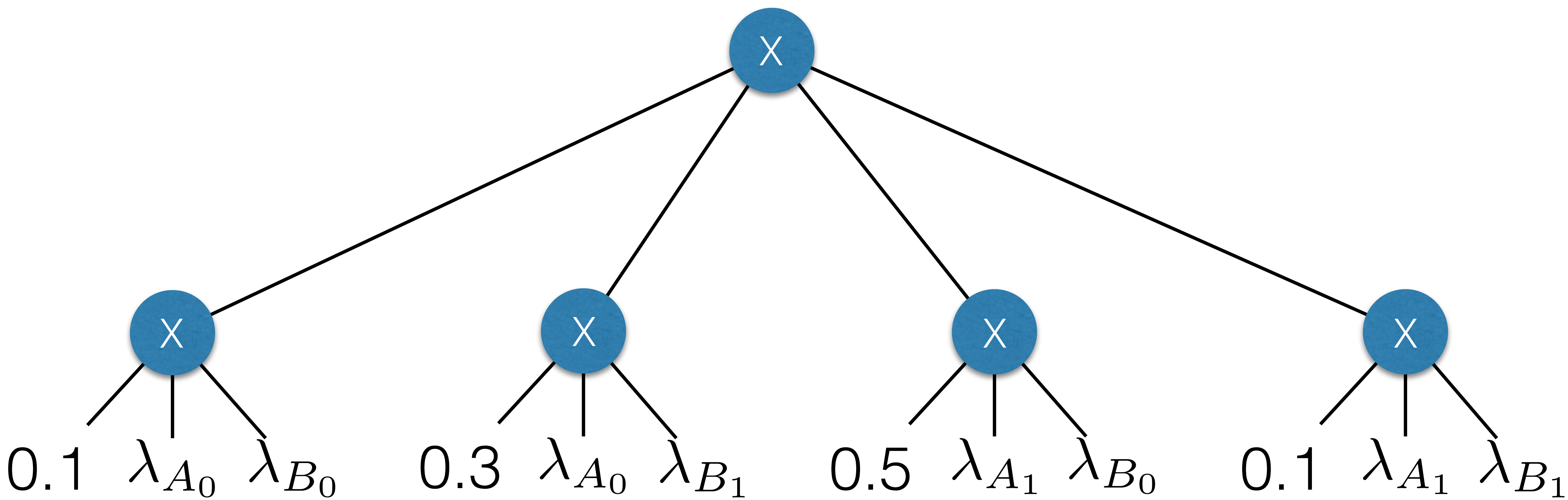
Background



Network Polynomial

$$f = 0.1\lambda_{A_0}\lambda_{B_0} + 0.3\lambda_{A_0}\lambda_{B_1} + 0.5\lambda_{A_1}\lambda_{B_0} + 0.1\lambda_{A_1}\lambda_{B_1}$$

Background



Sum-Product Network

a directed acyclic graph formed by sum and product nodes.

Critic

Good Points

- Novel model
- Well written
- Logical flow
- Experiments
- Relate to other field

6 SUM-PRODUCT NETWORKS AND THE CORTEX

The cortex is composed of two main types of cells: pyramidal neurons and stellate neurons. Pyramidal neurons excite the neurons they connect to; most stellate neurons inhibit them. There is an interesting analogy between these two types of neurons and the nodes in SPNs, particularly when MAP inference is used. In this case the network is composed of max nodes and sum nodes (logs of products). (Cf. Riesenhuber and Poggio [23], which also uses max and sum nodes, but is not a probabilistic model.) Max nodes are analogous to inhibitory neurons in that they select the highest input for further propagation. Sum nodes are analogous to excitatory neurons in that they compute a sum of their inputs. In SPNs the weights are at the inputs of max nodes, while the analogy with the cortex suggests having them at the inputs of sum (log product) nodes. One can be mapped to the other if we let max nodes ignore their children's weights and consider only their values. Possible justifications for this include: (a) it potentially reduces computational cost by allowing max nodes to be merged; (b) ignoring priors may improve discriminative performance [11]; (c) priors may be approximately encoded by the number of units representing the same pattern, and this may facilitate online hard EM learning. Unlike SPNs, the cortex has no single root node, but it is straighforward to extend SPNs to have multiple roots, corresponding to simultaneously computing multiple distributions with shared structure. Of course, SPNs are still biologically unrealistic in

Critic

Points to improve

- Related works
- Background

1 INTRODUCTION

The goal of probabilistic modeling is to represent probability distributions compactly, compute their marginals and modes efficiently, and learn them accurately. Graphical models [22] represent distributions compactly as normalized products of factors: $P(X = x) = \frac{1}{Z}\prod_k \phi_k(x_{\{k\}})$, where $x \in \mathcal{X}$ is a d-dimensional vector, each *potential* ϕ_k is a function of a subset $x_{\{k\}}$ of the variables (its scope), and $Z = \sum_{x\in\mathcal{X}} \prod_k \phi_k(x_{\{k\}})$ is the *partition function*. Graphical models have a number of important limitations. First, there are many distributions that admit a compact representation, but not in the form above. (For example,

This can be seen as follows. The partition function Z is intractable because it is the sum of an exponential number of terms. All marginals are sums of subsets of these terms; thus if Z can be computed efficiently, so can they. But Z itself is a function that can potentially be compactly represented using a deep architecture. Z is computed using only two types of operations: sums and products. It can be computed efficiently if $\sum_{x\in\mathcal{X}} \prod_k \phi_k(x_{\{k\}})$ can be reorganized using the distributive law into a computation involving only a polynomial number of sums and products. Given a graphical model, the inference problem in a nutshell is to perform this reorganization. But we can instead learn from the outset a model that is already in efficiently computable form,

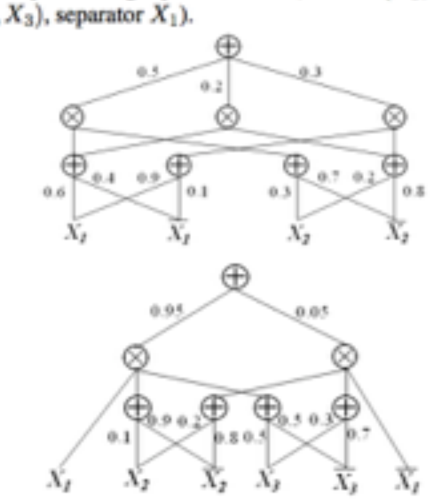
viewing sums as implicit hidden variables. This leads naturally to the question: what is the broadest class of models that admit such an efficient form for Z?

We answer this question by providing conditions for tractability of Z, and showing that they are more general than previous tractable classes. We introduce *sum-product networks (SPNs)*, a representation that facilitates this treatment and also has semantic value in its own right. SPNs can be viewed as generalized directed acyclic graphs of mixture models, with sum nodes corresponding to mixtures over subsets of variables and product nodes corresponding to features or mixture components. SPNs lend themselves naturally to efficient learning by backpropagation or EM. Of course, many distributions cannot be represented by polynomial-sized SPNs, and whether these are sufficient for the real-world problems we need to solve is an empirical question. Our experiments show they are quite promising.

2 SUM-PRODUCT NETWORKS

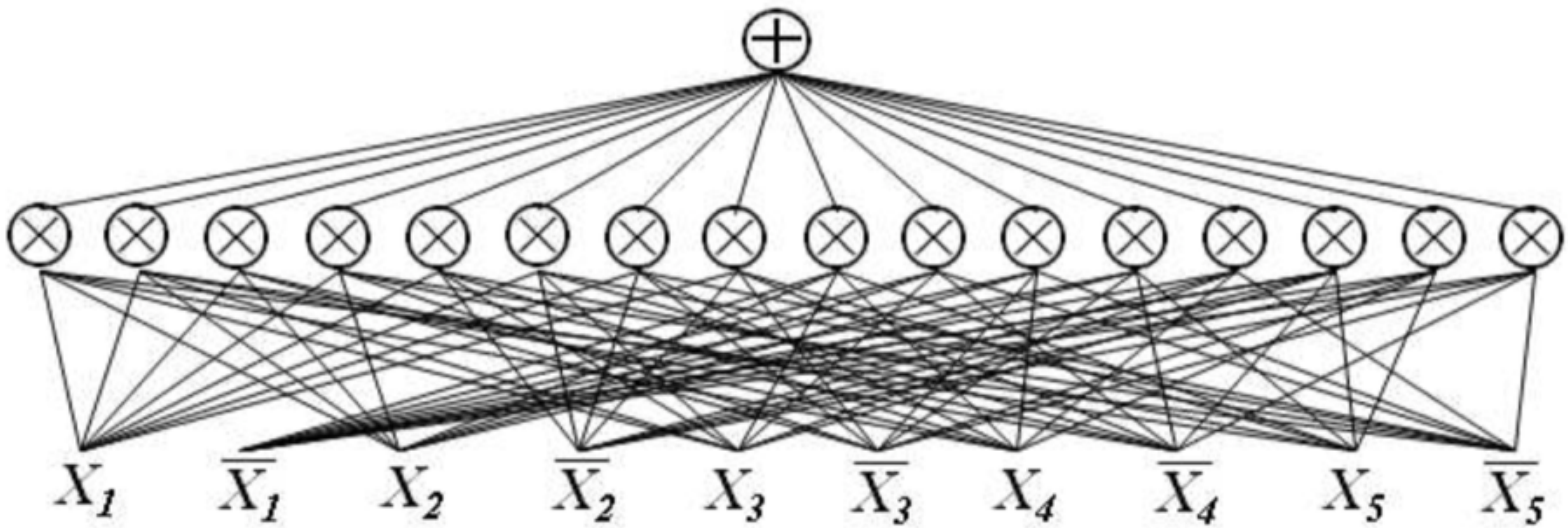
For simplicity, we focus first on Boolean variables. The extension to multi-valued discrete variables and continuous

Figure 1: Top: SPN implementing a naive Bayes mixture model (three components, two variables). Bottom: SPN implementing a junction tree (clusters (X_1, X_2) and (X_1, X_3), separator X_1).

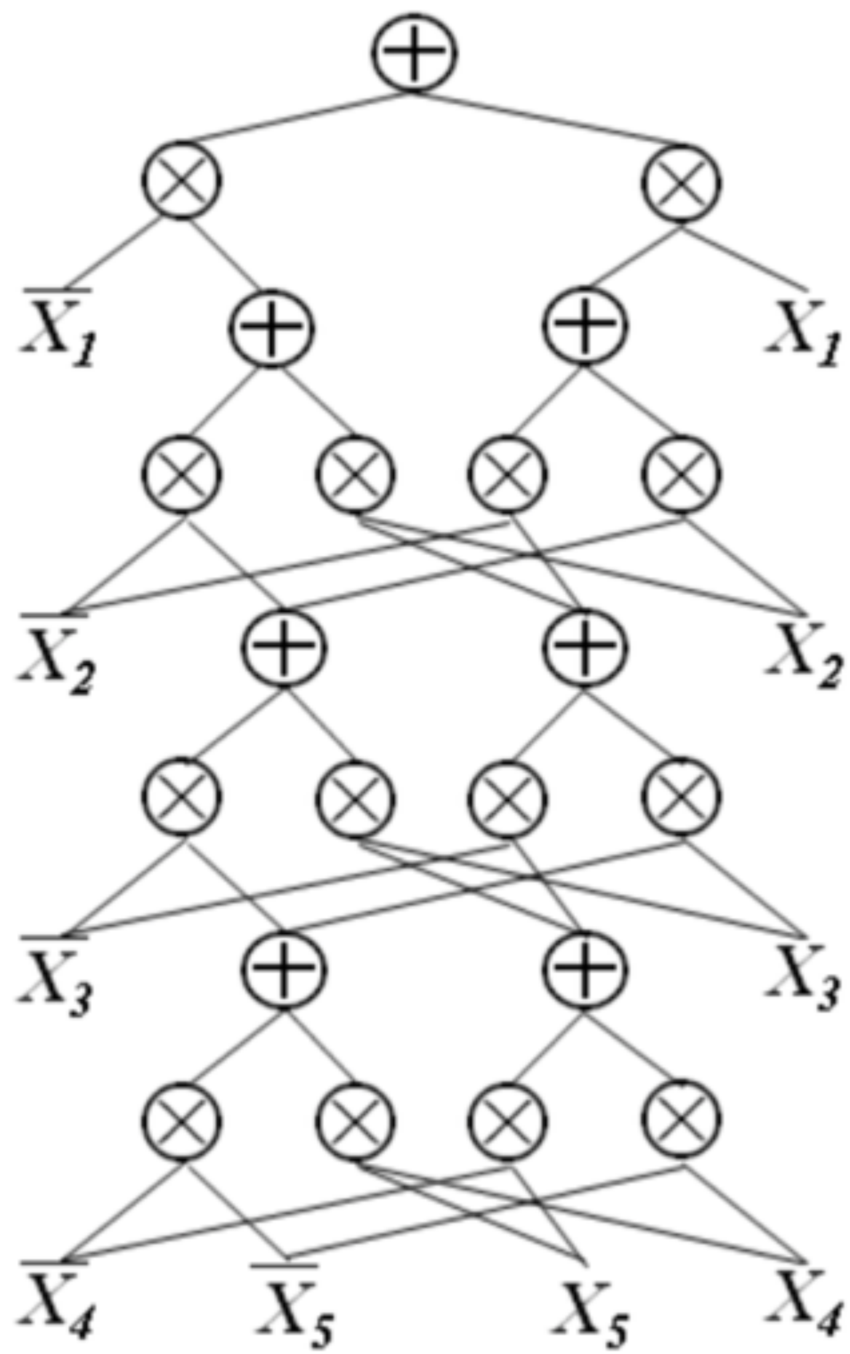


$x_1, \ldots, x_d$ *is a rooted directed acyclic graph whose leaves*

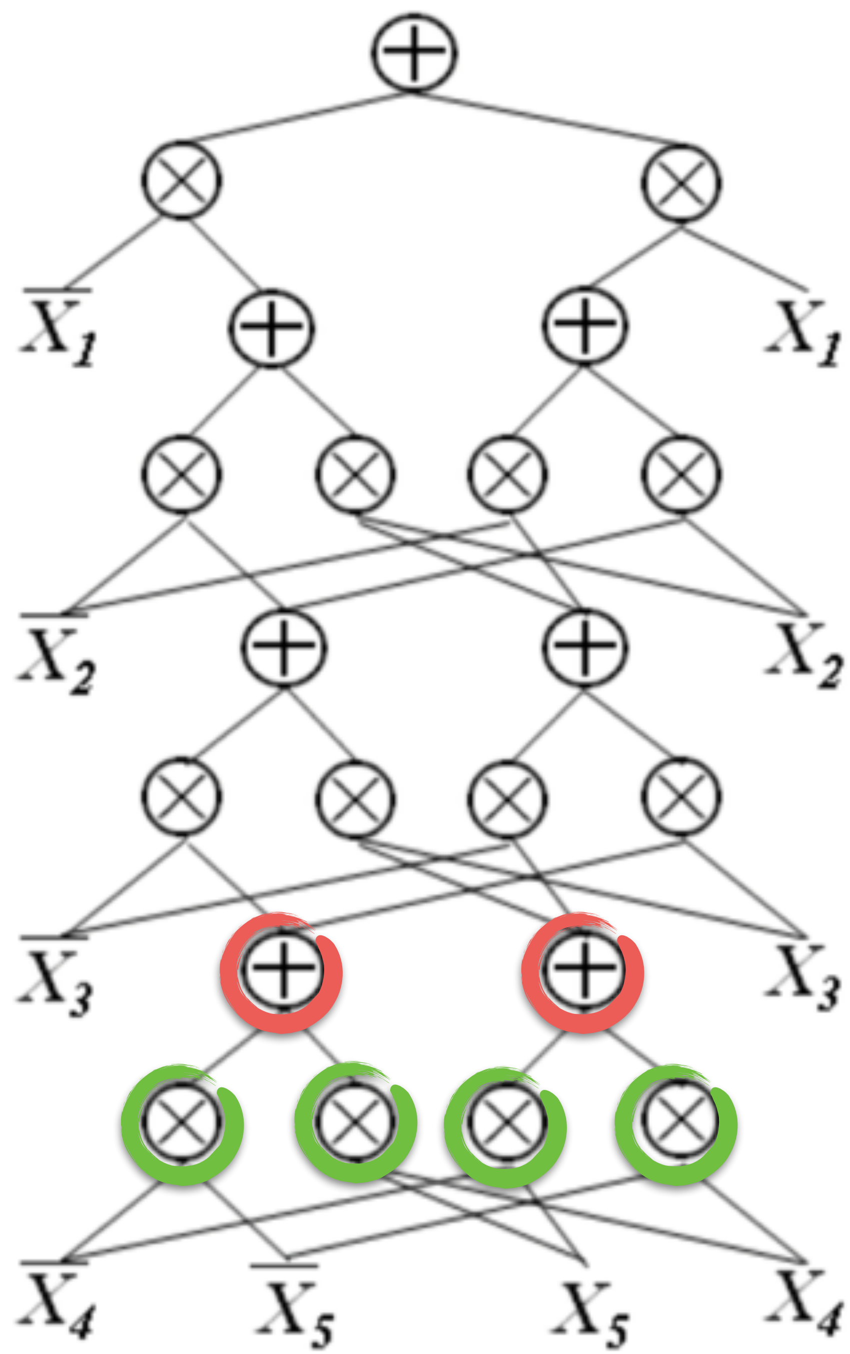
The Problem


X_1
$\overline{X_1}$
X_2
$\overline{X_2}$
X_3
$\overline{X_3}$
X_4
$\overline{X_4}$
X_5
$\overline{X_5}$

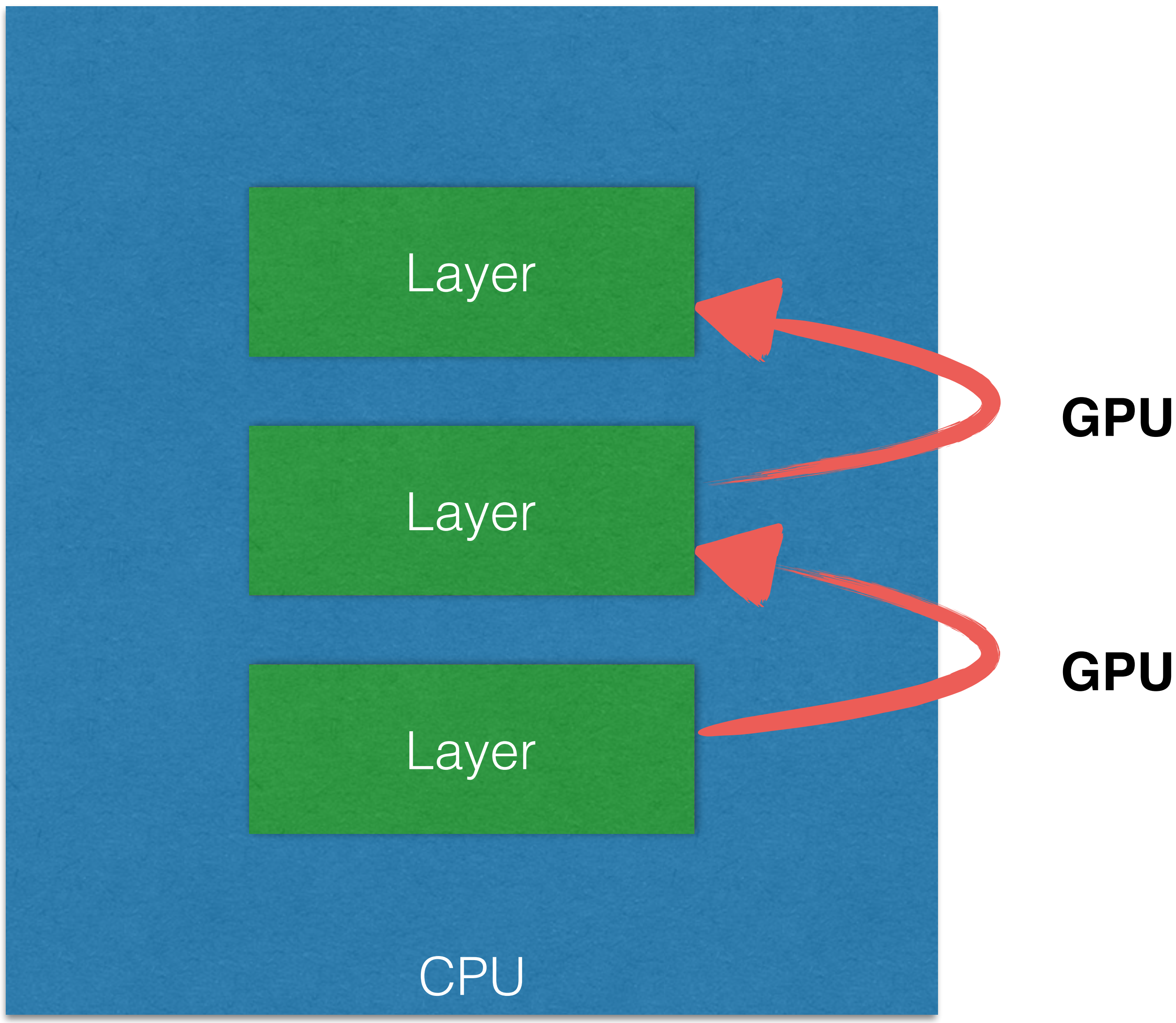
Parallel Computation



Possible Issues



The Comparison



Implementation

torch

Caffe

TensorFlow

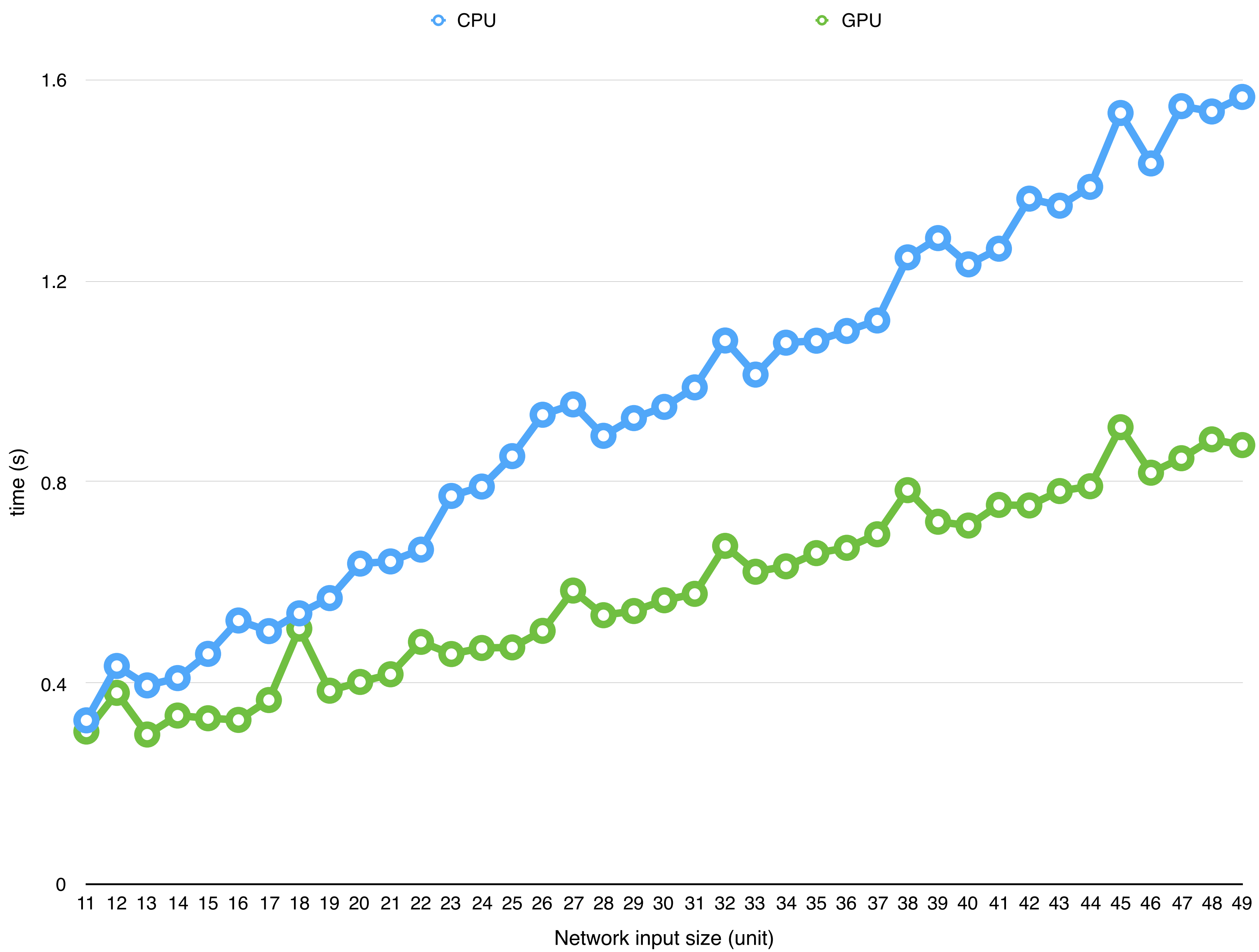
theano

Implementation

```python
import numpy as np
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Layer, Activation
import time
import tensorflow as tf

f = open("results.csv", "w")

for i in range(10, 50):

    print("--> Running at %d <--" % i)
    INPUT_SIZE = i
    OUTPUT_SIZE = INPUT_SIZE
    nb_class = 3

    # global parameters
    batch_size = 128
    nb_epoch = 40

    np.random.seed(123)

    X_train = np.random.rand(INPUT_SIZE, nb_class)
    Y_train = np.random.rand(OUTPUT_SIZE, nb_class)

    X_test = np.random.rand(INPUT_SIZE)
    Y_test = np.random.rand(OUTPUT_SIZE)

    print("--> Building model...")
    model = Sequential()
    model.add(Dense(INPUT_SIZE, input_shape=(nb_class,)))
    model.add(Activation('linear'))
    model.add(Dense(OUTPUT_SIZE))
    model.add(Activation('linear'))

    print("--> Compiling model...")

    start_time = time.time()

    model.compile(loss='categorical_crossentropy', optimizer='rmsprop')

    final_time = time.time()
    diff_time = final_time - start_time
    print("--> Exec time:")
    print(diff_time)

    f.write(str(i)+","+str(diff_time)+","+"\n")

f.close()
```

The Comparison

CPU x GPU



Conclusions

- Sum-product networks are a new deep architecture for modelling and inference
- Paper is well done with good ideas and nice presentation
- Inference and learning can be parallelized per layer
- GPUs can speed up SPN learning and inference