JUICE: A Julia Package for Logic and Probabilistic Circuits

Meihua Dang, Pasha Khosravi, Yitao Liang, Antonio Vergari, Guy Van den Broeck

Computer Science Department
University of California, Los Angeles
{mhdang, pashak, yliang, aver, guyvdb}@cs.ucla.edu

Abstract

JUICE is an open-source Julia package providing tools for logic and probabilistic reasoning and learning based on logic circuits (LCs) and probabilistic circuits (PCs). It provides a range of efficient algorithms for probabilistic inference queries, such as computing marginal probabilities (MAR), as well as many more advanced queries. Certain structural circuit properties are needed to achieve this tractability, which JUICE helps validate. Additionally, it supports several parameter and structure learning algorithms proposed in the recent literature. By leveraging parallelism (on both CPU and GPU), JUICE provides a fast implementation of circuit-based algorithms, which makes it suitable for tackling large-scale datasets and models.

Introduction

Decision making in the real world requires the ability to compactly represent and easily learn complex models, such as probability distributions and logical knowledge bases. It also requires the ability to perform logic and probabilistic reasoning about complex models. Moreover, in many sensitive domains, these computations need to be carried out exactly and efficiently. Classical models like Bayesian and Markov networks, and recent neural density estimators like variational autoencoders (Kingma and Welling 2013) and normalizing flows (Papamakarios et al. 2019) can capture complex distributions, but at the cost of being highly intractable. They only offer approximate answers to inference queries.

In contrast, probabilistic circuits (PCs) (Choi, Vergari, and Van den Broeck 2020) are expressive, deep, yet tractable probabilistic models that permit exact inference for many types of queries, in time linear in the size of their representation. They are increasingly used in downstream applications that require efficient probabilistic inference: algorithmic fairness (Choi, Dang, and Van den Broeck 2020), missing data (Khosravi et al. 2019, 2020; Correia, Peharz, and de Campos 2020), graphical model inference (Shih and Ermon 2020), probabilistic programming (Holzten, Van den Broeck, and Millstein 2020; Skryagin et al. 2020), activity recognition (Galindez Olascoaga et al. 2019), vision (Stelzner, Peharz, and Kersting 2019), and explainability (Nourani et al. 2020; Wang, Khosravi, and Van den Broeck 2020).

PCs are the probabilistic counterpart to tractable logic circuits (LCs) (Darwiche and Marquis 2002). They are compact representations of Boolean functions that are successfully employed in many knowledge representation and reasoning scenarios, often as a target for compilation (Darwiche and Marquis 2002; Van den Broeck and Suciu 2017).

We introduce JUICE, a Julia package that offers researchers and practitioners efficient routines to construct, compile, learn and reason with LCs and PCs. It provides a wide range of functionality, while being easy to extend and customize. Next, we give brief theoretical background, and discuss the high-level design and implementation of JUICE.

Background on Tractable Circuits

Representation

Circuits, including LCs and PCs, are computational graphs where nodes are computation units and edges define an order of execution. A LC $G$ is a directed acyclic graph encoding a logical formula where each inner node represents either disjunction or conjunction, and each input node is a logical literal, e.g., $X = 1$. Examples of LCs are $d$-DNNF circuits, binary, and sentential decision diagrams (Darwiche and Marquis 2002; Darwiche 2011).

A PC $(G, \theta)$ over random variables (RVs) $X$ represents a joint probability distribution $p(X)$ with parameters $\theta$. PCs are a family of tractable probabilistic models that includes arithmetic circuits (Darwiche 2002), sum-product networks (Poon and Domingos 2011), and cutset networks (Rahman, Kothalkar, and Gogate 2014). A PC structure $G$ is similar to a LC structure, except that conjunction nodes are written as products, disjunction nodes are written as sums.

1 JUICE source code is available at https://github.com/Juice-jl
and input nodes $n$ are associated with some tractable distribution $f_n$—for example, an indicator function for discrete variables, or a Gaussian for continuous variables. Parameters $\theta$ are associated with each sum node and input distribution.

Concretely, let $\text{in}(n)$ be the set of inputs of an inner node $n$. Then $p(X)$ is defined as follows:

$$p_n(x) = \begin{cases} f_n(x) & \text{if } n \text{ is an input node} \\ \prod_{c \in \text{in}(n)} p_c(x) & \text{if } n \text{ is a product node} \\ \sum_{c \in \text{in}(n)} \theta_{n,c} p_c(x) & \text{if } n \text{ is a sum node} \end{cases}$$

Intuitively, a product node $n$ defines a factorized distribution, and a sum node $n$ defines a mixture model parameterized by weights $\theta_{n,c}$. See Figure 1 for an example of a PC.

### Inference

Given certain structural properties, PCs support tractable probabilistic inference. JUICE makes it easy to test for these properties and construct circuits that have them.

A circuit is said to be smooth if for every sum node all of its inputs depend on the same set of RVs; it is decomposable if for every product node its inputs depend on disjoint sets of RVs (Darwiche and Marquis 2002). Given a smooth and decomposable PC, computing marginal probabilities (MAR) becomes tractable for any partial evidence. The computation reduces a customized feed forward evaluations of the circuit. This also implies tractable computation of conditional probabilities (CON), which are ratios of marginals.

A circuit is deterministic if for every complete input assignment $x$, at most one output of every sum node has a non-zero output. Decomposability and determinism enables tractable exact inference of most-probable explanations (MPE) (Choi and Darwiche 2017), also called MAP inference.

Another useful property is structured decomposability; it means that product nodes with the same scope decompose their variables in the same way (Kisa et al. 2014). Forms of structured decomposability guarantee tractable inference for advanced queries and manipulations: multiplying PCs (Shen, Choi, and Darwiche 2016), or computing expectations (Choi, Van den Broeck, and Darwiche 2015; Khosravi et al. 2019) and KL divergences (Liang and Van den Broeck 2017).

### An Overview of the JUICE Package

Table 1 summarizes the main compilation, reasoning and learning functionality implemented in the JUICE package.

### Learning

JUICE supports highly efficient parameter learning given complete or incomplete data (Kisa et al. 2014; Peharz et al. 2016). For example, the following code snippet learns the parameters of a PC on GPU and evaluates the likelihoods given training examples. We report runtimes for the training set PLANTS (Lowd and Davis 2010) with 17,412 examples and a PC with around 150 thousand nodes.

```
estimate_parameters(pc, data) # MLE; 88 ms
EVI(pc, data) # Full Evidence; 83 ms
```

JUICE also implements several structure learners that adopt hill-climbing strategies (Liang, Bekker, and Van den Broeck 2017; Dang, Vergari, and Van den Broeck 2020).

### Design

We model circuits as linked node structures. Inference routines iterate over the circuit forward or backward, passing results from node to node. Arbitrary inference algorithms can be implemented by providing different lambda functions, corresponding to different computations, to a general-purpose, optimized circuit traversal and propagation infrastructure. When modifying circuits, we leverage Julia’s automatic garbage collection, and the fact that nodes only link to their inputs, not the consumers of their output.

### Parallel computing on CPU and GPU

A linked node representation is an intuitive data structure for circuits. However, it has the drawback that it makes computations sparse, making it harder to leverage parallelism to speed up computation. To optimize performance during inference and learning, we translate the circuit’s DAG into a layered computational graph, starting with the input layer, and each layer only depending on the previous layers. Since the computations on the nodes in the same layer can be cached in one large vector, we can simultaneously parallelize our computation over the nodes in the layer on the one hand, and training examples or inference task data on the other hand. Additionally, we leverage Julia’s multiple dispatch to provide customized kernels to accelerate computation on both CPUs and GPUs (using SIMD and CUDA kernels respectively). Experiments show that CPU parallelism gives significant speed-ups, which even become an order of magnitude faster with GPU parallelism, all using the same underlying data structures.

### Other circuit types

Finally, because JUICE separates structures (LCs and PCs) and algorithms (e.g., inference and learning), making custom circuit models with specific semantics is relatively easy. For example, JUICE also provides an implementation of logistic circuits (Liang and Van den Broeck 2019) which is a type of discriminative PC for classification.

---

<table>
<thead>
<tr>
<th>Models</th>
<th>Logic Circuits</th>
<th>Probabilistic Circuits</th>
<th>Pairs of Circuits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithms</td>
<td>forward &amp; backward traversal</td>
<td>EVI, MAR, CON, MPE (conditional) sampling</td>
<td>multiply</td>
</tr>
<tr>
<td></td>
<td>smooth, condition, split, merge, clone</td>
<td>MLE/EM parameter learning</td>
<td>KL divergence</td>
</tr>
<tr>
<td></td>
<td>(weighted) model counting</td>
<td>hill climbing structure learning</td>
<td>expectations</td>
</tr>
<tr>
<td></td>
<td>compilation, SAT</td>
<td></td>
<td>expectations</td>
</tr>
</tbody>
</table>

Table 1: The list of major functionalities that JUICE supports. Many routines benefit from SIMD/GPU parallelization.
References


