From Logic to Learning: A Framework for Converting Sentential Decision Diagrams into Differentiable Neural Networks

Author: Matheus Sanches Jurgensen (IME - USP)

Advisors: Denis Deratani Mauá, Jonas Rodrigues Lima Gonçalves

INTRODUCTION

Artificial Intelligence currently faces a dichotomy between the interpretability of symbolic reasoning and the generalization capabilities of deep neural networks. This work addresses this gap within the **Neuro-Symbolic AI** domain, specifically contributing to the **dPASP** project. The primary objective is to design a software framework capable of converting **Sentential Decision Diagrams (SDDs)** into **Neural Networks**. This framework allows efficient probabilistic inference and enables optimization of probabilistic parameters from data using standard gradient-based deep learning methods.

IMPORTANT DEFINITIONS

Sentential Decision Diagrams (SDDs) A highly structured type of logic circuit—a directed acyclic graph representing a Boolean function where leaf nodes are variables and internal nodes are logical operations. SDDs possess specific structural properties to ensure tractability:

- **Decomposability:** Subcircuits in an AND node do not share variables.
- **Determinism:** Subcircuits in an OR node are logically mutually exclusive.

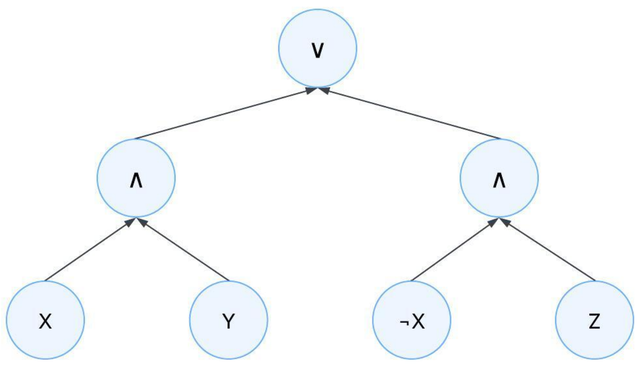

Figure 1: *SDD circuit representing* $f(x, y, z) = (x \wedge y) \vee (\neg x \wedge z)$

METHODOLOGY

Circuit Transformation The framework converts the logical SDD into a **Smooth Arithmetic Circuit** compatible with neural architectures.

- **AND Nodes** $\rightarrow$ **Product Nodes** ($\times$).
- **OR Nodes** $\rightarrow$ **Sum Nodes** ($+$).

Inference Implementation This logic is implemented directly in **PyTorch**, treating the circuit as a computational graph.

- **The Goal:** Compute the conditional probability of specific query variables (Q) assuming that certain evidence variables (E) have occurred.
- **The Process:** This is calculated as $P(Q|E) = P(Q, E)/P(E)$, meaning that the system performs two forward passes through the network.

Training Implementation The system learns from data using standard backpropagation and **Stochastic Gradient Descent (SGD)**.

- **Probabilistic Parameters:** These are learnable weights that represent the probability that a specific variable is true.
- **Training Data:** Datasets where we observe the logical consequences, but the root causes (probabilistic variables) may be unobserved.
- **Loss Function:** Minimize the **Negative Log-Likelihood**, to maximize the likelihood that our learned parameters explain the observed data.

RESULTS

Case Study: The framework was validated using the "Alarm" Bayesian network (5 variables), utilizing a dataset where probabilistic causes (*Burglary*, *Earthquake*, *Hears Alarm*) are hidden, and only logical consequences (*Alarm*, *Calls*) are observed.

Learning Accuracy We evaluated learning by varying dataset size and training epochs. Results show that increasing sample size significantly reduces Mean Absolute Error (MAE) and variance, effectively recovering the latent probabilities (0.1, 0.2, and 0.7, respectively).

Num Samples	Num Epochs	Avg. MAE	Std. Error	Avg. Time (ms)
$1,000$	$1,000$	0.0179	0.0054	$1,010$
$10,000$	$5,000$	0.0063	0.0016	$19,321$
$100,000$	$10,000$	0.0013	0.0004	$107,696$

Table 1: *MAE, variance, and time metrics for parameter learning.*

Samples / Epochs	Burglary (0.1)	Earthquake (0.2)	Hears Alarm (0.7)
$1,000$ / $1,000$	0.114 ± 0.019	0.197 ± 0.019	0.697 ± 0.020
$100,000$ / $10,000$	0.100 ± 0.001	0.199 ± 0.001	0.699 ± 0.001

Table 2: *Learned probabilities for the probabilistic variables.*

CONCLUSION & NEXT STEPS

The final results of this work include:

- A system that unifies probabilistic inference and parameter learning into a single architecture optimized via backpropagation.
- A modular, extensible, and rigorously tested (149 tests) framework based on SOLID principles.

Regarding future work, the next steps involve supporting complex primitives like Annotated Disjunctions, handling non-stratified programs, and implementing architectural optimizations for GPU parallelization.