Practical Implementation of Deep Differentiable Logic Gate Networks: Design and Benchmarks

CS Olympiad Projects

Sofia Maragò, Andrea Coato, Zeno Orglmeister, Jonathan Braun

Google Zurich, August 21, 2025

Introduction

Neural network models have become an integral part of everyday life. However, their inference often comes with high computational and memory demands, requiring specialized and expensive hardware.

One promising solution to this challenge is to simplify the model's operations, shifting the resource-intensive workload to a centralized training phase. The optimization we have researched works through a network fully comprised of logic gates. This approach can enable faster, more accessible, and widespread deployment of neural networks, benefiting both large tech companies and small-scale businesses alike.

Previous work and project structure

We used previous research on LGNs[1] as a starting point for the creation of a deep differentiable network for classification tasks with relaxation to real-valued logic for training.

We mainly experimented with the choice of input mapping to boolean values, node quantity, network architecture, output aggregation, optimizer and other heuristics. We built a framework to generate different types of networks and tweak training parameters easily, it is comprised of:

- MNIST-to-bitset extractor
- network architecture generator
- gradient descent trainable gates
- converter into AIG
- accuracy evaluator

[1] Felix Petersen et al. *Deep Differentiable Logic Gate Networks*. 2022. arXiv: 2210.08277 [cs.LG]. URL: https://arxiv.org/abs/2210.08277.

Inputs

MNIST input

We found that for the MNIST dataset a simple conversion with rounding is sufficient. This result was confirmed giving a rounded input to a "standard" neural network for MNIST, and the accuracy decrease was negligible. We also tried increasing the input information with various heuristics and pre-processing the image with additional activations.

For further work on image classification with LGNs, a new approach for efficient discretizations of the inputs has to be found, possible methods could be categorizing brightness and vibrancy of each pixel.

Network creation

Network structure generator

Every network structure has a pair of nodes connected to each node in a layer, in addition to the input. We started with a simple deep network with a constant number of nodes in each layer and randomized connections.

Another approach we tried was a convolution-like network, which was promising but required a large number of nodes and a deeper network and specialized gate-binarization tactics.

On both approaches we tried applying heuristics, such as encouraging connections to the central parts of the image.

Network gate training

Differentiable logic gates

Logic gates are, by definition, non differentiable, therefore in order to train a neural network we used a probabilistic relaxation. Every gate type, described as a function $f : \{0,1\} \times \{0,1\} \to \{0,1\} \times \{0,1\}$, is mappable to real valued logic.

Every gate type has an associated probability in such a way that, through softmax, it is mapped to a probability simplex in 16 dimensions in each node. The gate choice is then decided as the most likely parameter over the distribution.

ID	Operator	Real-values
0	$False$	0
1	$A \wedge B$	$A \cdot B$
2	$\overline{A \Rightarrow B}$	$A - AB$
3	A	A
4	$\overline{A \Leftarrow B}$	$B - AB$
5	B	B
6	$A \dot{\vee} B$	$A + B - 2AB$
7	$A \vee B$	$A + B - AB$
8	$\overline{A \vee B}$	$1 - (A + B - AB)$
9	$\overline{A \dot{\vee} B}$	$1 - (A + B - 2AB)$
10	$\overline{B}$	$1 - B$
11	$A \Leftarrow B$	$1 - B + AB$
12	$\overline{A}$	$1 - A$
13	$A \Rightarrow B$	$1 - A + AB$
14	$\overline{A \wedge B}$	$1 - AB$
15	$True$	1

Gradient descent training

The inference calculation for each gate, starting from the input gates, layer by layer, can be described as:

$$a = \sum_{i=0}^{15} p_i \cdot f_i(a_1, a_2)$$

where p_i is the softmaxed probability at gate a, and a_1, a_2 its input values. For backpropagation we used the Jax library.

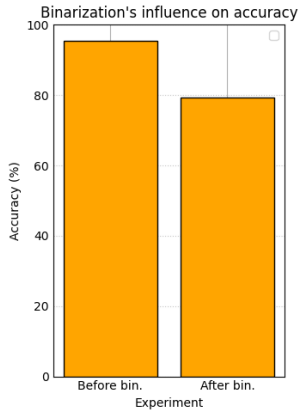
Loss and optimizers

The number of output nodes is designed to significantly exceed 10, every classification group has a set of output nodes allowing for a final choice through the highest activation mean. The network has been trained with AdamW optimizer. We tried various configurations of learning rate and weight decay based on different architectures and heuristic setups.

Gate choice

Although we observed that most nodes the network naturally converge into a hard choice when given enough training epochs, the percentage loss from this forced conversion can be big, in our case 16-20% on the 96% (with approx. 15000 nodes).

A possible way to tackle this problem was the integration of an entropy cost over the probability distributions into the loss function (approach which we observed is particularly hard to balance with the actual learning phase). Other approaches would be to convert nodes with an indecisive trend early in the training to let the other nodes fit around.



Additional heuristics

As additional heuristics we encouraged pass-through gates with an initial parametrization much higher than the other gates, instead of randomly choosing over a gaussian distribution.

Secondly we tried slightly encouraging or-pooling in order to minimize wash out of values in shallower layers. We also experimented with a sigmoid function which would reduce indecisive gate values during training but found out it was not improving performance significantly, a gradual increase of the temperature of the softmax instead had almost the same effect.

Conversion and evaluation

From 16 gate types to AIG

Before evaluating the final accuracy of the LGN we converted it into a simpler architecture with only and inverter gates, for faster execution and possibility of comparison to other networks with the AIGER format (common binary and ASCII format for AIG).

This also gives us the possibility to run our final network through optimizers which get rid of additional gates for even faster computation.

Further development ideas

Convolutional structures would be our main point of further development, for example enforcing gate probability distribution equivalence in the first layers instead of only an inspiration for the network structure.

The most important issue to address as to accuracy and therefore possibly application to more complex datasets is the big gap between real-valued network accuracy and logic gate relaxation.

Other possibilities include residual connections and further research into logic gate specific heuristics. In addition it could be possible to also train the connections dinamically, in parallel to the values.

Thanks for listening!