PRINCETON UNIVERSITY

Introduction to Differentiable Logic Neural Networks

Mila Bileska, **Lino Gerlach**, Elliott Kauffman, Isobel Ojalvo, Liv Våge

SmartPixels Meeting

March 11, 2026

Introduction

Deep Differentiable Logic Gate Networks

Felix Petersen
Stanford University
University of Konstanz
mail@felix-petersen.de

Christian Borgelt
University of Salzburg
christian@borgelt.net

Hilde Kuehne
University of Frankfurt
MIT-IBM Watson AI Lab
kuehne@uni-frankfurt.de

Oliver Deussen
University of Konstanz
oliver.deussen@uni.kn

Abstract

Recently, research has increasingly focused on developing efficient neural network architectures. In this work, we explore logic gate networks for machine learning tasks by learning combinations of logic gates. These networks comprise logic gates such as "AND" and "XOR", which allow for very fast execution. The difficulty in learning logic gate networks is that they are conventionally non-differentiable and therefore do not allow training with gradient descent. Thus, to allow for effective training, we propose differentiable logic gate networks, an architecture that combines real-valued logics and a continuously parameterized relaxation of the network. The resulting discretized logic gate networks achieve fast inference speeds, e.g., beyond a million images of MNIST per second on a single CPU core.

arXiv:2210.08277

- 2022 NeurIPS paper proposes 'Logic Gate Networks'
  - No weights or biases, only choices of gates
  - Inherently discrete & non-differentiable
- Trick to make them differentiable (compute-intense)
  - Slow training in *differentiable* mode
  - Very fast inference in *discrete* mode
- Very interesting for many applications in HEP

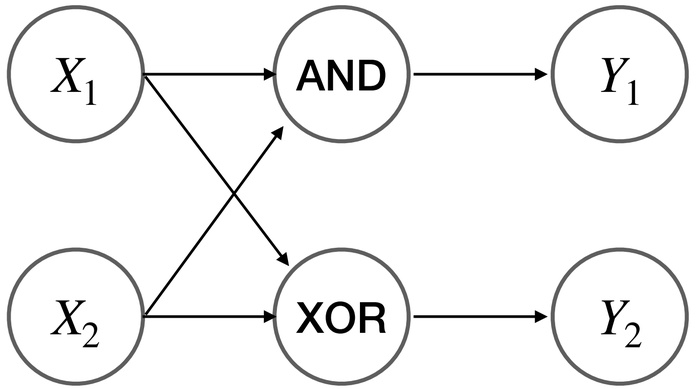
Logic Gate Networks

- Logic Gate: map two binary inputs to one binary output:

$$f : \{0,1\} \times \{0,1\} \rightarrow \{0,1\}$$

- Logic Gate Network (LGN): combination of logic gates
- Minimal example: Adding two binary numbers:

X_1	X_2	Y_1	Y_2
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



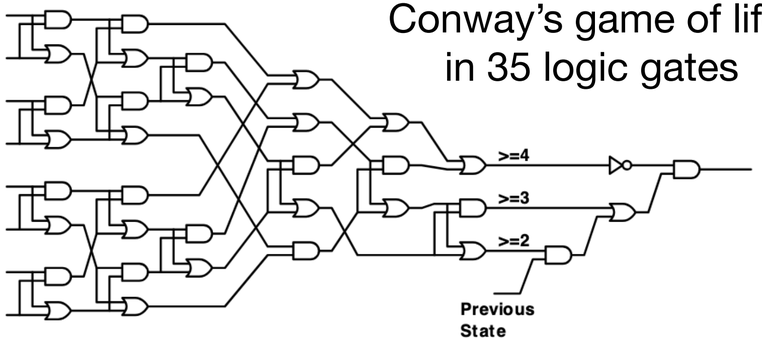
4 possible inputs,
2 outputs each
$\rightarrow 2^4 = 16$ gates

ID	Operator	00	01	10	11
0	False	0	0	0	0
1	$A \wedge B$	0	0	0	1
2	$\neg(A \Rightarrow B)$	0	0	1	0
3	A	0	0	1	1
4	$\neg(A \Leftarrow B)$	0	1	0	0
5	B	0	1	0	1
6	$A \oplus B$	0	1	1	0
7	$A \vee B$	0	1	1	1
8	$\neg(A \vee B)$	1	0	0	0
9	$\neg(A \oplus B)$	1	0	0	1
10	$\neg B$	1	0	1	0
11	$A \Leftarrow B$	1	0	1	1
12	$\neg A$	1	1	0	0
13	$A \Rightarrow B$	1	1	0	1
14	$\neg(A \wedge B)$	1	1	1	0
15	True	1	1	1	1

arXiv:2210.08277

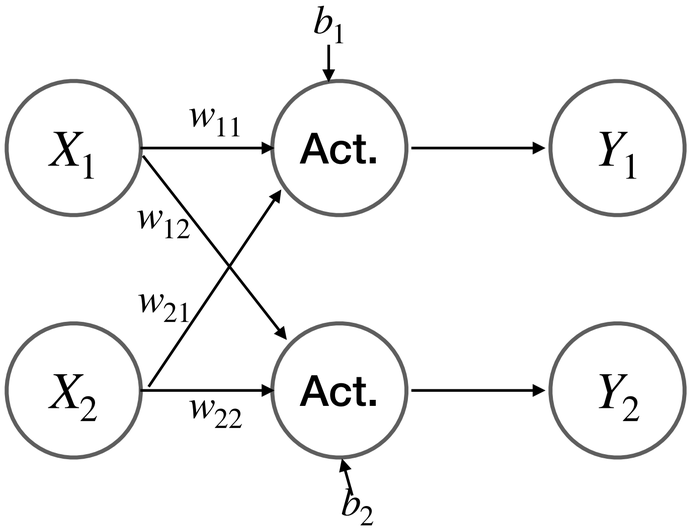
Neural- vs. Logic Gate Nets

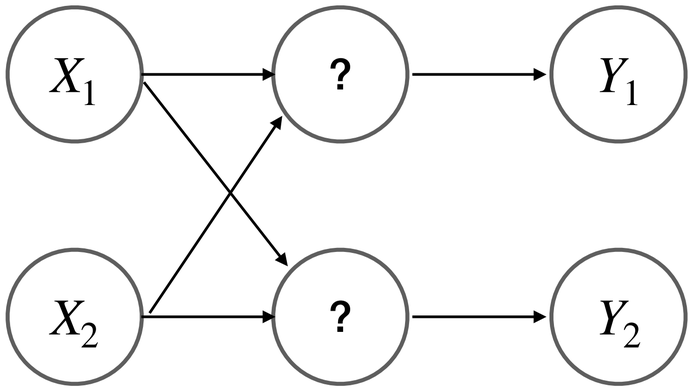
- Like classic NNs, can model any function with sufficient size and suitable architecture
  - NN: Find optimal weights and biases
  - LGN: Find optimal choice of logic gate at each neuron



Conway's game of life in 35 logic gates

https://www.moria.us/old/3/programs/life/





	Classic NN	LGN
Parameter-ization	Weights & Biases	Choice of Logic Gates
Training	**Back-propagation**	?

LGNs are not differentiable by default. Cannot use backprop!

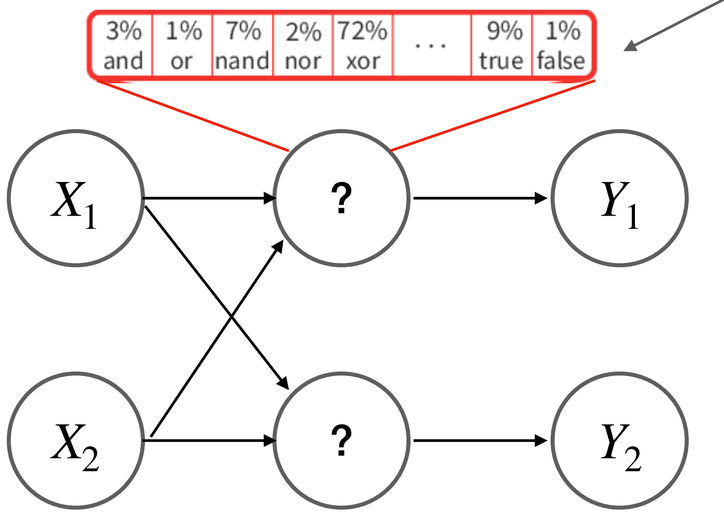
Making LGNs Differentiable

Trick: apply two *relaxations* to make LGNs differentiable:

1.) Real-valued logic

2.) Weighted sum over all Gates

ID	Operator	real-valued	00	01	10	11
0	False	0	0	0	0	0
1	$A \wedge B$	$A \cdot B$	0	0	0	1
2	$\neg(A \Rightarrow B)$	$A - AB$	0	0	1	0
3	A	A	0	0	1	1
4	$\neg(A \Leftarrow B)$	$B - AB$	0	1	0	0
5	B	B	0	1	0	1
6	$A \oplus B$	$A + B - 2AB$	0	1	1	0
7	$A \vee B$	$A + B - AB$	0	1	1	1
8	$\neg(A \vee B)$	$1 - (A + B - AB)$	1	0	0	0
9	$\neg(A \oplus B)$	$1 - (A + B - 2AB)$	1	0	0	1
10	$\neg B$	$1 - B$	1	0	1	0
11	$A \Leftarrow B$	$1 - B + AB$	1	0	1	1
12	$\neg A$	$1 - A$	1	1	0	0
13	$A \Rightarrow B$	$1 - A + AB$	1	1	0	1
14	$\neg(A \wedge B)$	$1 - AB$	1	1	1	0
15	True	1	1	1	1	1



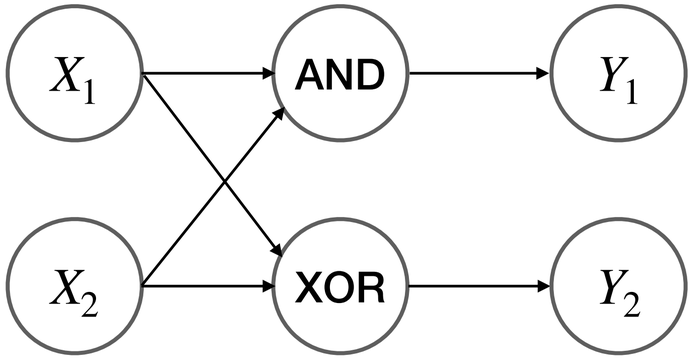
Softmax: $\sum_{i=0}^{15} \frac{e^{w_i}}{\sum_j e^{w_j}} \cdot f_i(a_1, a_2)$

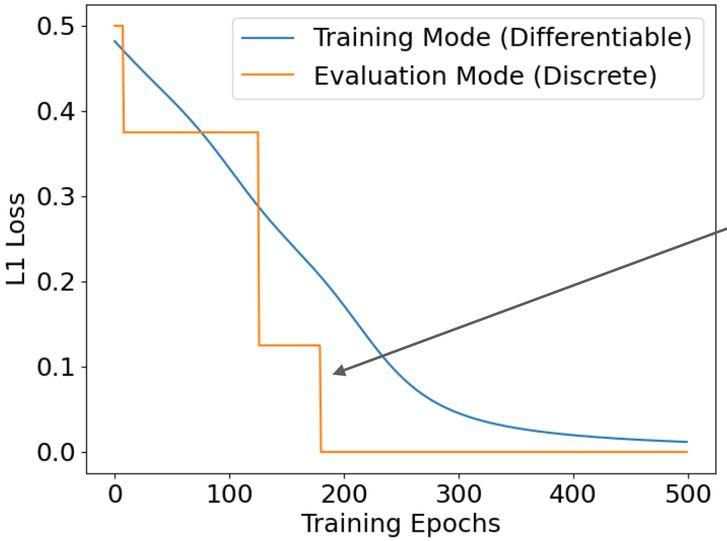
Can now calculate $\frac{\partial \mathcal{L}}{\partial w_i}$ with backprop.!

But very slow: evaluate 16 floating-point ops at each neuron (vs one binary gate)

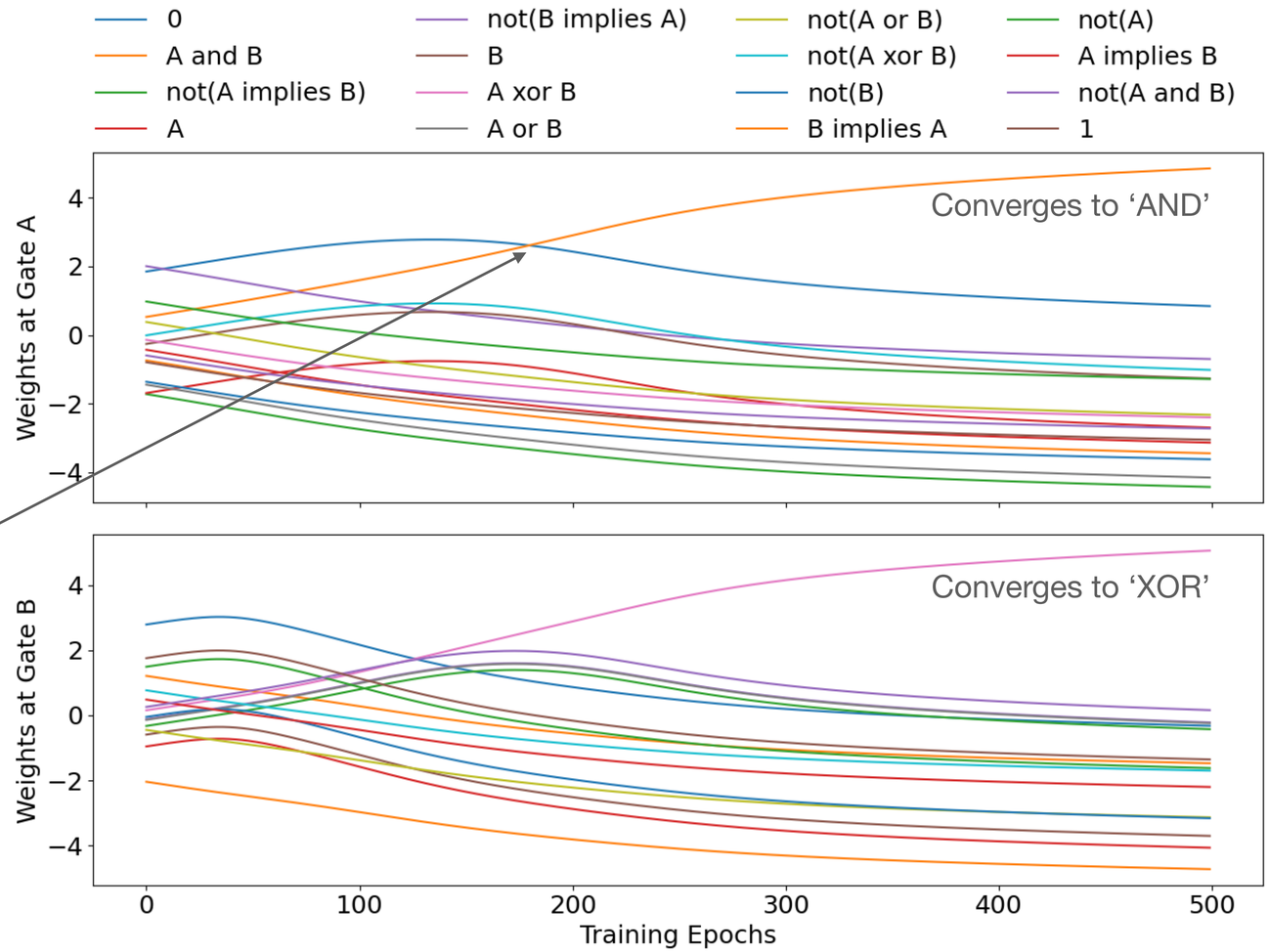
After training, pick gate with highest w_i

Minimal Training Example



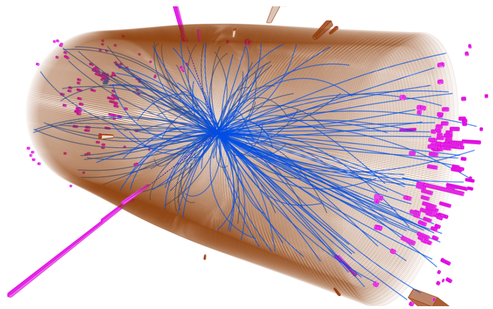


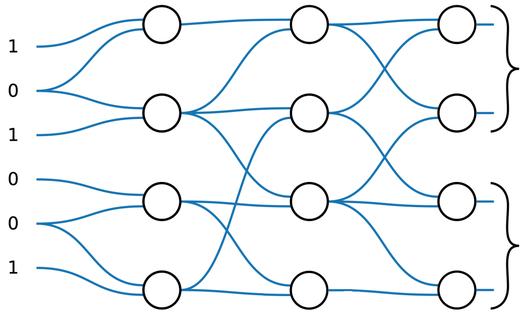
Final order of weights
(= choice of gates)
is reached

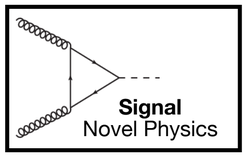


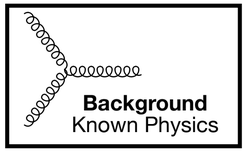
Classification & Regression

- Can aggregate n binary outputs into integer values via **group sum**
- Control scale- and offset via additional parameters (τ, β)
- Train for classification & regression w/ standard loss functions (e.g. CE / MSE)

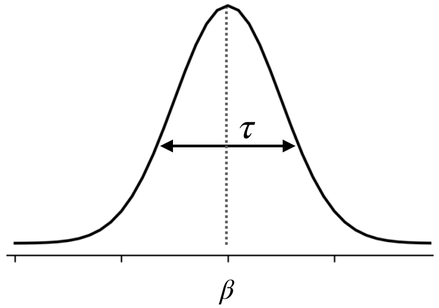








$$\sum_{j=i\cdot n/k+1}^{(i+1)\cdot n/k} a_j \,/\, \tau + \beta$$



With $\tau = 0$ and $\beta = 0$, all output will be between 0 and $\frac{n}{k}$

Why are LGNs so Fast?

Approaches to make **NN**s faster come *for free* with **LGN**s

1.) Quantization

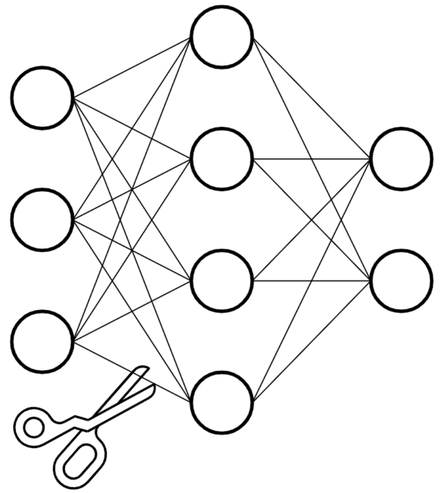
Training finds optimal binary neurons

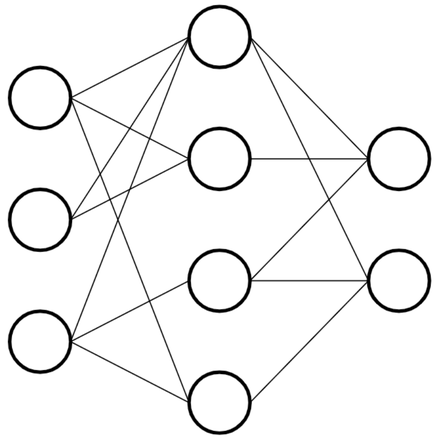
- Maximally quantization aware training

2.) Pruning

Implement entire network in single C routine

- Compiler will remove any redundant logic (w/ optimizations)





Initialization and regularization can lead to >90 % trivial gates

Scaling Up LNNs

Convolutional Differentiable Logic Gate Networks

Felix Petersen
Stanford University
InftyLabs Research
mail@felix-petersen.de

Hilde Kuehne
Tuebingen AI Center
MIT-IBM Watson AI Lab
h.kuehne@uni-tuebingen.de

Christian Borgelt
University of Salzburg
christian@borgelt.net

Julian Welzel
InftyLabs Research
welzel@inftylabs.com

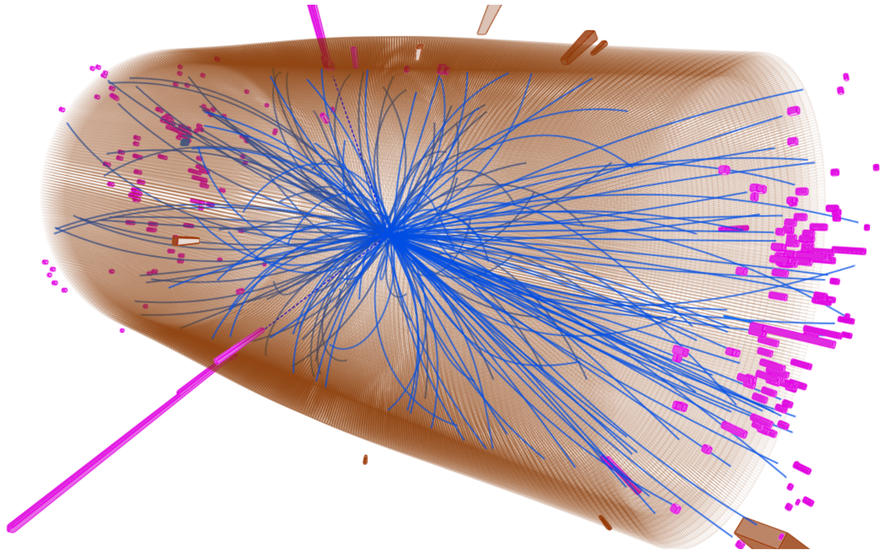
Stefano Ermon
Stanford University
ermon@cs.stanford.edu

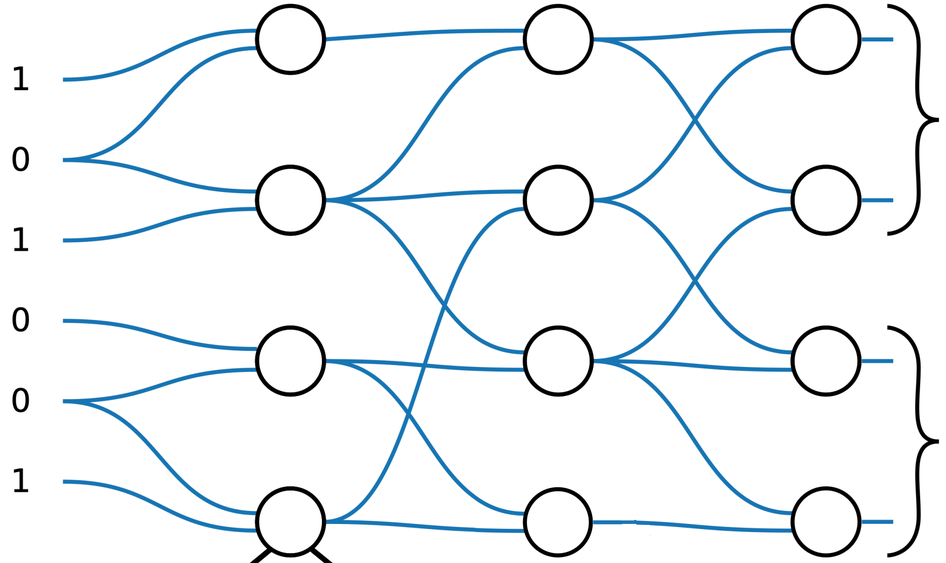
Abstract

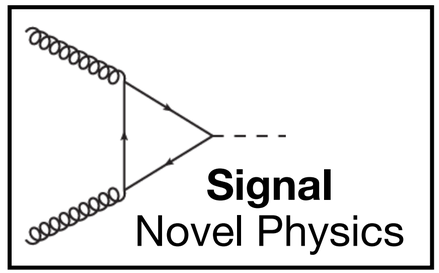
With the increasing inference cost of machine learning models, there is a growing interest in models with fast and efficient inference. Recently, an approach for learning logic gate networks directly via a differentiable relaxation was proposed. Logic gate networks are faster than conventional neural network approaches because their inference only requires logic gate operators such as NAND, OR, and XOR, which are the underlying building blocks of current hardware and can be efficiently executed. We build on this idea, extending it by deep logic gate tree convolutions, logical OR pooling, and residual initializations. This allows scaling logic gate networks up by over one order of magnitude and utilizing the paradigm of convolution. On CIFAR-10, we achieve an accuracy of 86.29% using only 61 million logic gates, which improves over the SOTA while being 29× smaller.

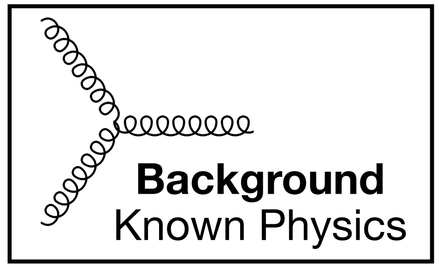
- 2024 NeurIPS paper introduced convolutions for LGNs
- SOTA performance on small benchmarks until now
- ~2 orders of magnitude smaller models than optimized, matrix-multiplication-based approaches
- But: no source code available

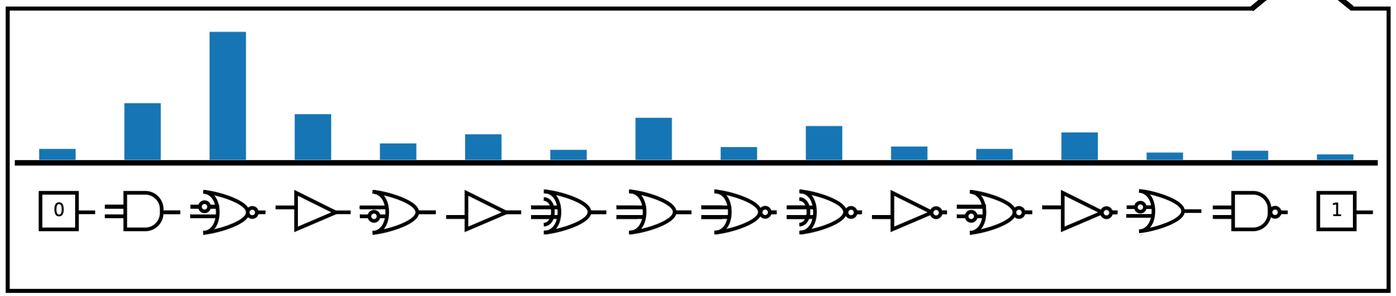
Intermediate Recap











Instead of weights and biases, **LGNs directly learn combinations of logic gates** (e.g. “AND”, “XOR”) via probability distributions over all gates. During inference, only the most likely gate is used at each node, leading to **extremely efficient deployment on hardware.** This allows signal-vs-background **classification in real time** for HEP.

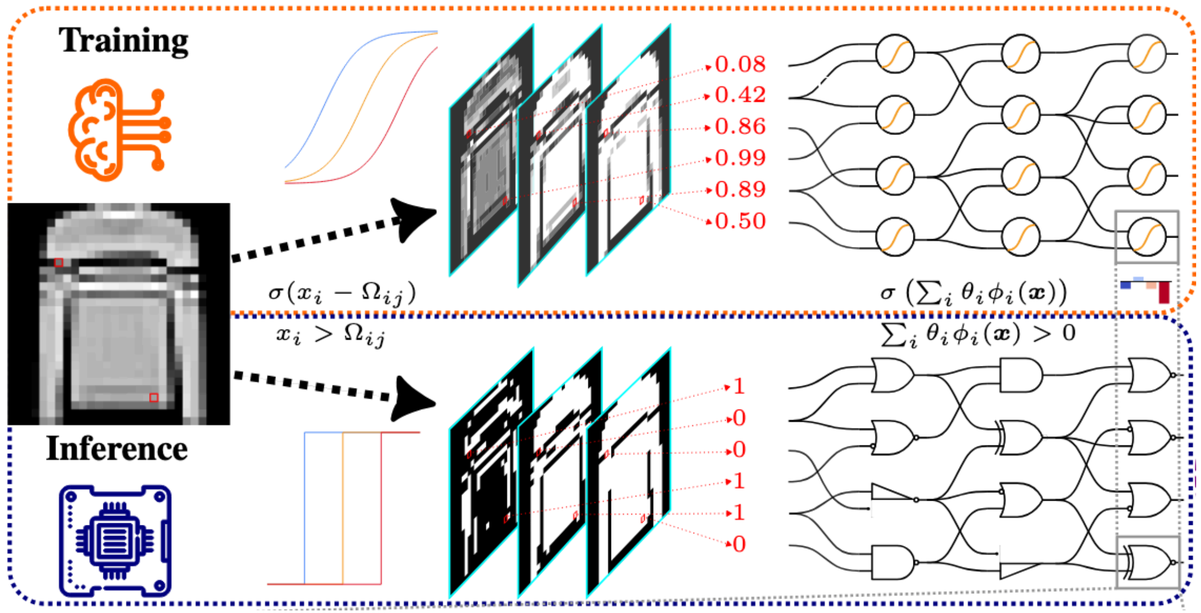
- LGNs promise a leap in computational efficiency for small ML tasks
  - Very interesting for on-chip, real-time AI (E.g. SmartPixels!)

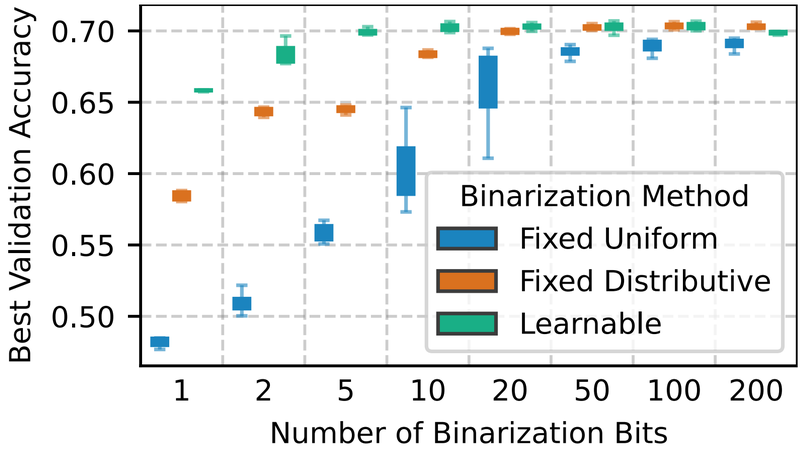
Our Contributions

- Conducted feasibility studies for LGNs in HEP:
  - *Rapid Inference of Logic Gate Neural Networks for Anomaly Detection in High Energy Physics* (arxiv.org/abs/2511.01908)
  - *Evaluation of Novel Fast Machine Learning Algorithms for Knowledge-Distillation-Based Anomaly Detection at CMS* (arxiv.org/abs/2510.15672)
- Fundamental ML research:
  - WARP Logic Neural Networks (arxiv.org/abs/2602.03527)
- Developed open source python package for training and inference of LNNs

Learnable Thresholding

- LNNs operate on boolean data -> Transform continuous inputs first
  - *Thermometer thresholding* expands dims along new *bit axis*
  - E.g. 3 thresholds: 28x28 grey-scale image -> 3x28x28 bit tensor
  - Thresholds [0.25, 0.5, 0.75]: 0.2 -> (0, 0, 0), 0.6 -> [1,1,0], …





Learning thresholds during training greatly improves performance

Open Source Python Package

README MIT license

torchlogix

pypi package 0.1.1 | unit-test.yml passing | docs online | license MIT | DOI 10.5281/zenodo.18800428 | NSF 2323298

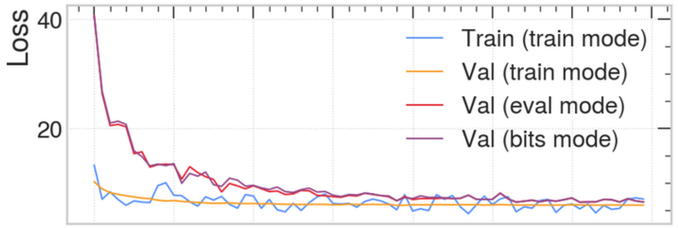
`torchlogix` is a `PyTorch`-based library for training and inference of **logic neural networks**. These solve machine learning tasks by learning combinations of boolean logic expressions. As the choice of boolean expressions is conventionally non-differentiable, relaxations are applied to allow training with gradient-based methods. The final model can be discretized again, resulting in a fully boolean expression with extremely efficient inference, e.g., beyond a million images of MNIST per second on a single CPU core.

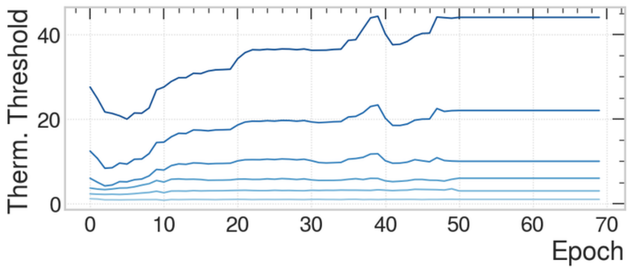
- Based on *difflogic* (code of original LGN paper)
- Extends by new features & optimizations:
  - Learnable Thermometer Thresholding
  - WARP Parametrization
  - Transpiler: trained model -> verilog (WIP)
  - Convolutions (2D & 3D)
    - As described in arxiv.org/abs/2411.04732

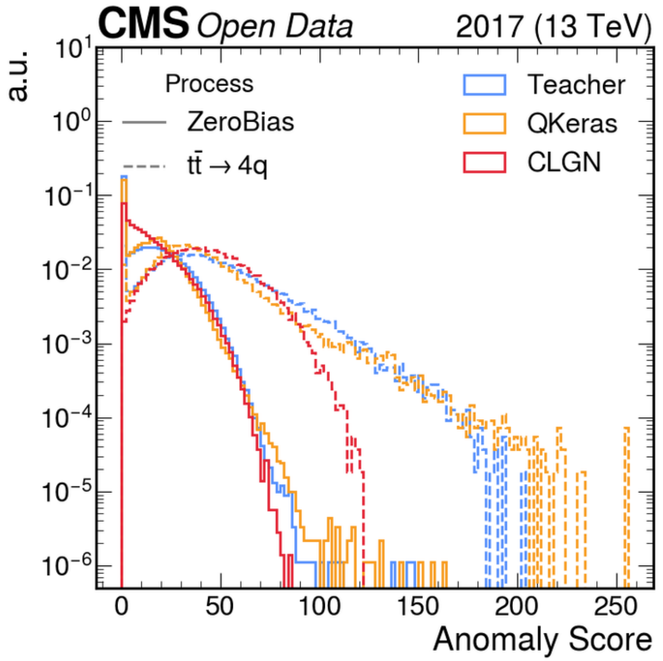
Backup

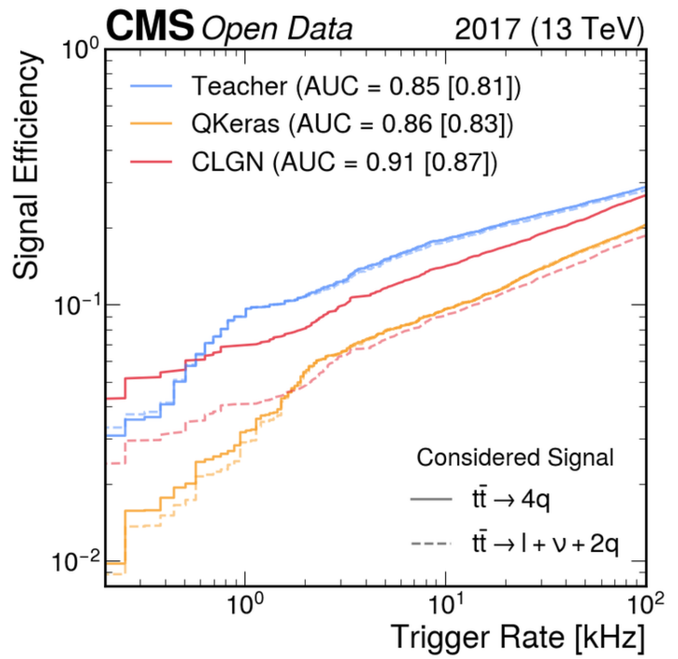
LGN-Based CICADA Student

- Trained LGN-based CICADA student on 2017 open data
- Convolutional LGN model learns to mimic the teacher nicely
- ROC-Curves also look promising
- Bit-wise encoding of Energy proved crucial
  - Breakthrough w/ *Learnable Thermometer Thresholding*





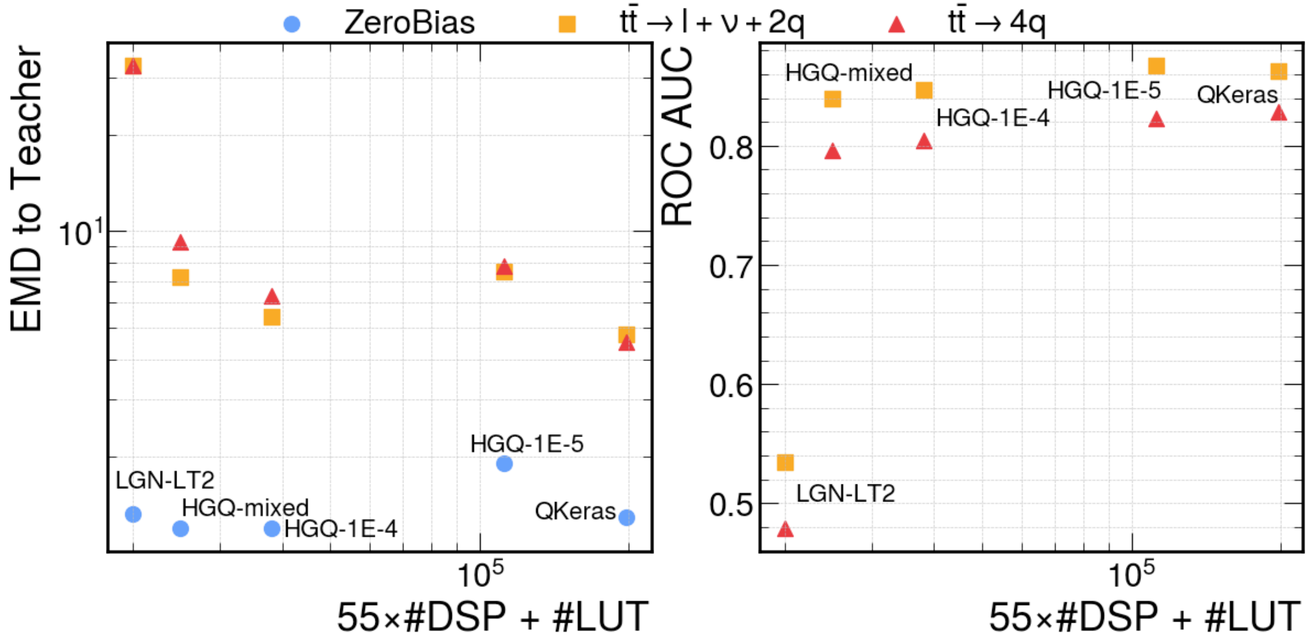




From Liv's ACAT talk. Proceedings on arxiv

LGN-based Student - Synthesis

- Initial FPGA synthesis emulation shows promising results: ~ x10 resource reduction
- But: Only possible for dense, non-convolutional model
- Still working on synthesizing convolutional LGNs



From our 4-page NeurIPS workshop paper. Preprint on arxiv

- Going from PyTorch -> Boolean C-code -> HLS -> RTL
  - Others go from PyTorch directly to RTL with impressive results. But how?
- We are no real FPGA-experts. Helping hands very welcome if someone's interested!