SAND2022-4482C

Adaptive High-Arity Logical Activation Functions

Jed A. Duersch (jaduers@sandia.gov)

Joint Work with Thomas A. Catanach and Niladri Das

SIAM UQ - April 2022 - Atlanta

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-1-1-13525.

Learning Logical Relationships in Binary Data

We would like **data-driven** approaches to **learn plausible logical relationships** between **Boolean claims** (true or false).

For example, we may desire to **fill in missing data** within binary sequences:

$$\text{Training data:} \quad \mathcal{D}$$

$$\text{Binary (Boolean) sequence:} \quad (b_i)_1^N = (01001011?011)$$

$$\text{Solve:} \quad p(b_9|(b_i)_{i\neq 9}, \mathcal{D}).$$

We could design **neural networks** for this, but **what would make them efficient?**

We always worry about **expressivity** and **trainability**.

As there are many, arbitrarily complicated, functions that may explain the training data.

Typical Activation Functions

The commonly used activation functions we might consider are: Rectified Linear Units (ReLU), MinMax[1], MaxAIL[2]

Gidon et al.[3] showed that a single human neuron is capable of learning the exclusive disjunction (**xor**).

None of these activation functions can achieve this in a single layer.

Lowe's **pairwise activation functions**[2] hardcode functional relationships corresponding to **and**, **or**, and **xnor** in a single layer.

Our approach allows **arbitrary n-argument truth functions** in a single layer.

[1] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," in International conference on machine learning. PMLR, 2013, pp. 1319–1327.

[2] S. C. Lowe, R. Earle, J. d'Eon, T. Trappenberg, and S. Oore, "Logical activation functions: Logit-space equivalents of boolean operators," arXiv preprint arXiv:2110.11940, 2021.

[3] A. Gidon, T. A. Zolnik, P. Fidzinski, F. Bolduan, A. Papoutsi, P. Poirazi, M. Holtkamp, I. Vida, and M. E. Larkum, "Dendritic action potentials and computation in human layer 2/3 cortical neurons," Science, vol. 367, no. 6473, pp. 83–87, 2020.

Truth Functions and Logical Arity

Boolean logic is typically formulated in terms of elementary functions, conjunction (**and**), disjunction (**or**), and negation (**not**).

For n arguments, a truth table contains 2^n values. There are 2^{2^n} ways to fill it.

4

Unary

ψ_1	$not(\psi_1)$
0	1
1	0

16

Binary

ψ_2	ψ_1	$or(\psi_1,\psi_2)$	$and(\psi_1,\psi_2)$	$xor(\psi_1,\psi_2)$	$imp(\psi_1,\psi_2)$
0	0	0	0	0	1
0	1	1	0	1	0
1	0	1	0	1	1
1	1	1	1	0	1

Material Implication

256

Ternary

ψ_3	ψ_2	ψ_1	$(\psi_1 \,?\, \psi_2 : \psi_3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Conditioned Disjunction

Notation

It is important to correctly distinguish potential **truth states** from corresponding **belief states (probabilities)** from the **representations** of belief.

	Truth State	Belief State	Logit Representation
Input:	$\xi_j \in \{0,1\}$	$p(\xi_j) \in (0,1)$	$x_j \in \mathbb{R}$
Antecedent:	$\psi_i \in \{0,1\}$	$p(\psi_i) \in (0,1)$	$y_i \in \mathbb{R}$
Consequent:	$\zeta_k \in \{0,1\}$	$p(\zeta_k) \in (0,1)$	$z_k \in \mathbb{R}$

We build towards an algorithm that processes inputs x, or $x^{(\ell)}$, as a linear transformation followed by an activation function,

$$y = Mx \quad \text{and} \quad z = f(y, A)$$

The output may then serve as the input to the next layer, $z = x^{(\ell+1)}$.

Generalizing Truth Functions to Belief Functions

To **fully incorporate uncertainty** into Boolean logic, we need to account for uncertainty in **antecedents**, **consequents**, and the truth table (**belief table**). Evaluating the resulting belief function is the usual marginalization,

Belief Hypercube

$$p(\zeta) = \begin{aligned} & p(\zeta \mid \psi_1, \psi_2)p(\psi_1, \psi_2) && + p(\zeta \mid \psi_1, \neg\psi_2)p(\psi_1, \neg\psi_2) \\ & + p(\zeta \mid \neg\psi_1, \psi_2)p(\neg\psi_1, \psi_2) && + p(\zeta \mid \neg\psi_1, \neg\psi_2)p(\neg\psi_1, \neg\psi_2). \end{aligned}$$

For n antecedents, the belief table is a hypercube with 2^n vertices. This example shows marginalization for a **general binary belief function**. If we hold antecedents to be independent, we can easily construct any truth function.

$$p_{\text{xor}}(\zeta) = p(\psi_1)\,(1 - p(\psi_2)) + (1 - p(\psi_1))\,p(\psi_2).$$

Products and Vanishing Gradients

Suppose we represent the belief table in conjunctive normal form.

$$\text{Let} \quad \psi = (\psi_1, \psi_2, \ldots, \psi_n) \qquad \text{so that} \quad p(\zeta) = \sum_{\psi \in \{0,1\}^n} p(\zeta \mid \psi) \prod_{i=1}^{n} p(\psi_i).$$

$$\text{Backpropagation gives} \quad \frac{\partial J}{\partial p(\zeta \mid \psi)} = \frac{\partial J}{\partial p(\zeta)} \prod_{i=1}^{n} p(\psi_i)$$

The gradient can change by orders of magnitude during training!

$$\text{and} \quad \frac{\partial J}{\partial p(\psi_i)} = \frac{\partial J}{\partial p(\zeta)} p(\zeta \mid \psi) \prod_{j \neq i}^{n} p(\psi_j).$$

Within each product, **every factor suppresses the gradient**, thus creating a sensitivity problem (ill-conditioning). If we turn the learning rate up, then once a vertex becomes plausible, the parameter updates become **unstable**.

Logit Representations

There is also an issue with **ensuring coherent antecedent probabilities**.

We can fix both of these problems by working with logit representations.

$$\text{Let} \quad y_i = \log\left(\frac{p(\psi_i)}{p(\neg\psi_i)}\right) \quad \text{so that} \quad p(\psi_i) = \frac{1}{1+\exp(-y_i)}.$$

Inputs

Outputs

$$\text{Likewise} \quad z = \log\left(\frac{p(\zeta)}{p(\neg\zeta)}\right) \quad \text{so that} \quad p(\zeta) = \frac{1}{1+\exp(-z)}.$$

For example, binary truth functions with only antecedent and consequent uncertainty (certain conditionals) would be

$$p(\zeta \mid \text{and}) = p(\psi_1)p(\psi_2) \qquad z = -\log\left(\exp(-y_1 - y_2) + \exp(-y_1) + \exp(-y_2)\right)$$

$$p(\zeta \mid \text{or}) = 1 - p(-\psi_1)p(-\psi_2) \qquad z = \log\left(\exp(y_1 + y_2) + \exp(y_1) + \exp(y_2)\right)$$

$$p(\zeta \mid \text{xor}) = p(\psi_1)p(-\psi_2) + p(-\psi_1)p(\psi_2) \qquad z = \log\left(\frac{\exp(y_1) + \exp(y_2)}{1 + \exp(y_1 + y_2)}\right)$$

Generalized Unary Belief Function

Rather than using fixed truth functions, we would like to **parameterize belief tables** so that our **data can drive any logic** that is useful. The logit construction, including conditional uncertainty, of the **general unary belief function**

General Unary Marginalization: $p(\zeta) = p(\zeta \mid \neg\psi_1)p(\neg\psi_1) + p(\zeta \mid \psi_1)p(\psi_1)$

$$\text{becomes} \quad z = \log\left(\frac{e^{\frac{a_0-a_1-y_1}{2}} + e^{\frac{a_0+a_1-y_1}{2}} + e^{\frac{-a_0+a_1+y_1}{2}} + e^{\frac{a_0+a_1+y_1}{2}}}{e^{\frac{-a_0+a_1-y_1}{2}} + e^{\frac{-a_0-a_1-y_1}{2}} + e^{\frac{-a_0-a_1+y_1}{2}} + e^{\frac{a_0-a_1+y_1}{2}}}\right)^{*}$$

Belief Vertex

$$\text{where} \quad a_0 = \log\left(\frac{p(\zeta \mid \neg\psi_1)}{p(\neg\zeta \mid \neg\psi_1)}\right) \quad \text{and} \quad a_1 = \log\left(\frac{p(\zeta \mid \psi_1)}{p(\neg\zeta \mid \psi_1)}\right).$$

This result is mildly interesting, but mostly **horrifying**. Such functions are very expensive to evaluate (and differentiate), and this is **just the unary case**!

* This probably isn't even numerical stable because we are moving the most important arithmetic into the exponent and backing out.

Log Sum Exp Max Approximation

Just as **the log is a functor**, mapping **multiplicative composition** to **additive composition,**

$$\log\left(\prod_{i=1}^{n} x_i\right) = \sum_{i=1}^{n} \log(x_i)$$

The log also approximately maps **additive composition** to **maximization.**

$$\log\left(\sum_{i=1}^{n} x_i\right) \approx \max_{i=1}^{n} \log(x_i)$$

$$\text{provided} \quad x_i > 0 \; \forall \; i \in [n].$$

This approximation is particularly good when the maximizing term dominates by an order of magnitude.

$$\log\left(\sum_{i=1}^{n} \exp(x_i)\right) \approx \max_{i=1}^{n} x_i \quad \text{provided} \quad x_i \in \mathbb{R} \; \forall \; i \in [n].$$

Unary Activation

Applying LSEM to the generalized unary marginalization gives the simple formula,

LSEM Approximation for Unary Activation

$$z = \frac{1}{2}\left[\max(|y_1| + s, |d + y_1|) - \max(|y_1| - s, |d - y_1|)\right],$$

where $s = a_0 + a_1$ and $d = a_1 - a_0$.

Sparse gradients with no attenuation!

Algorithm 1 Adaptive Unary Activation

Input: $\boldsymbol{y}$ is a $w_{\text{out}} \times 1$ vector of activation inputs, or antecedent logits.

Implied by the shape of $\boldsymbol{y}$, w_{out} is number of channels or output width.

$\boldsymbol{A}$ is a $w_{\text{out}} \times 2$ matrix of activation parameters, belief-table logits.

Output: $\boldsymbol{z}$ is a $w_{\text{out}} \times 1$ vector of activation outputs, consequent logits.

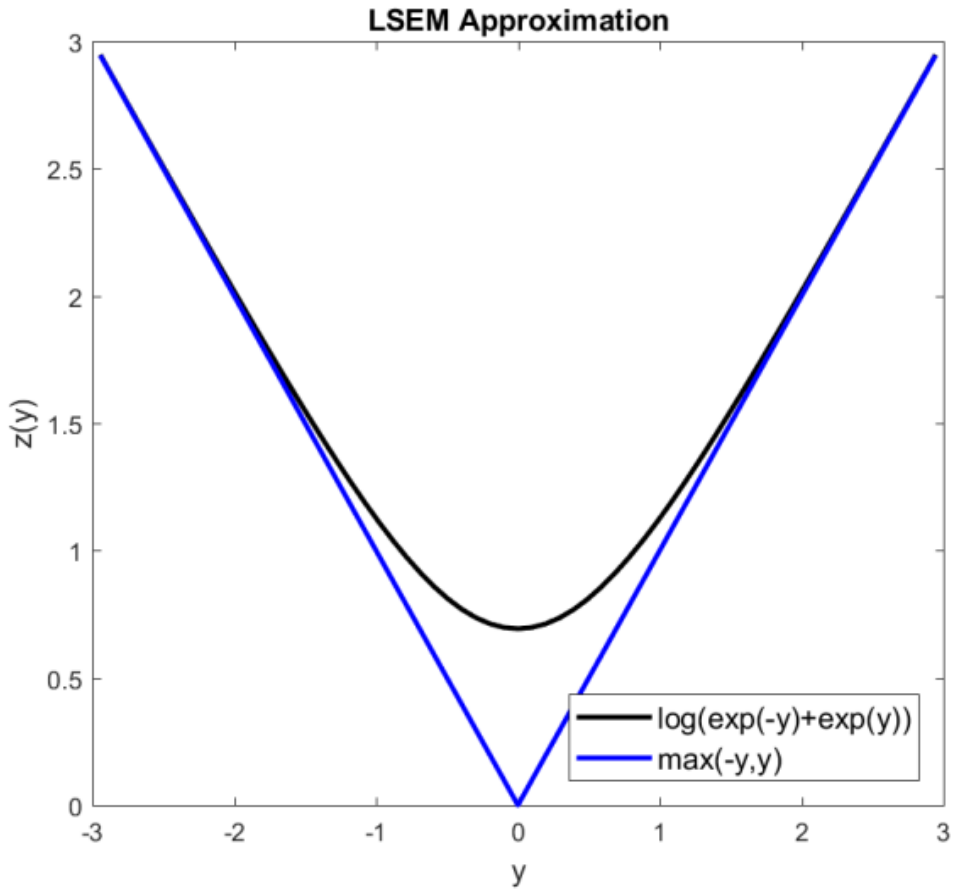
1: **function** $\boldsymbol{z} = \textbf{unary}(\boldsymbol{y}, \boldsymbol{A})$

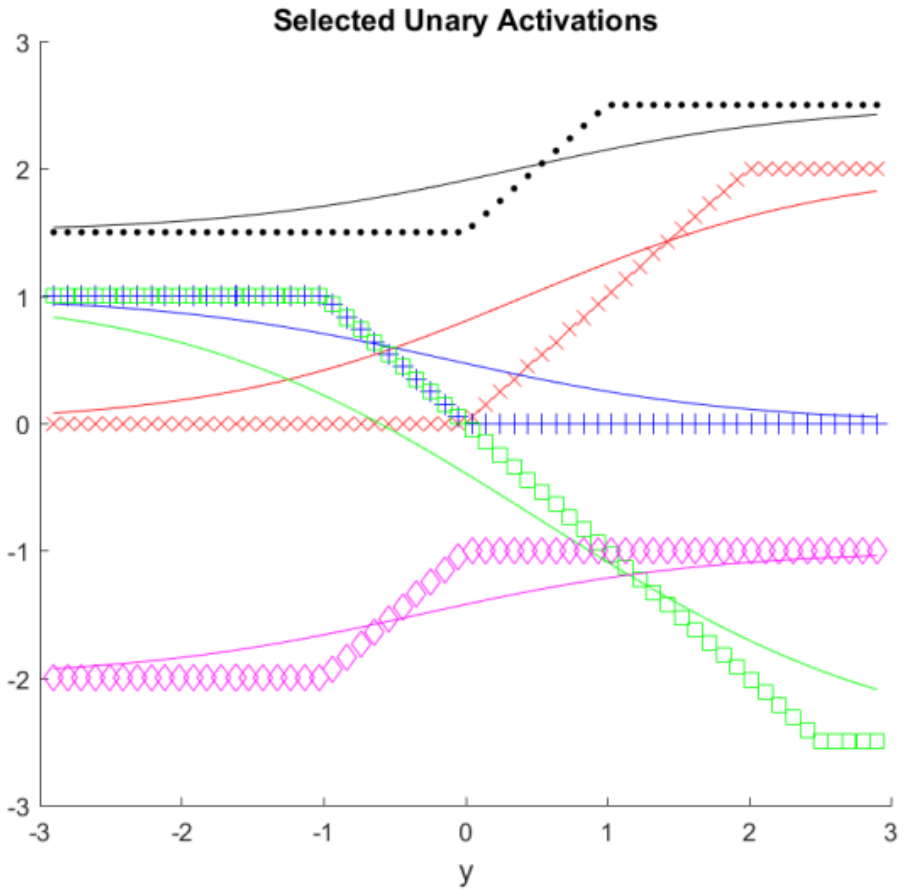
2: Compute belief-table sums, $\boldsymbol{s}$, and differences, $\boldsymbol{d}$, as $\begin{bmatrix}\boldsymbol{s} & \boldsymbol{d}\end{bmatrix} = \boldsymbol{A}\begin{bmatrix}1 & -1\\ 1 & 1\end{bmatrix}$.

3: Compute activation outputs, $\boldsymbol{z} = \frac{1}{2}\left[\max(|\boldsymbol{y}| + \boldsymbol{s}, |\boldsymbol{d} + \boldsymbol{y}|) - \max(|\boldsymbol{y}| - \boldsymbol{s}, |\boldsymbol{d} - \boldsymbol{y}|)\right]$.

4: **end function**

What do these function look like?





Even just **the unary case subsumes ReLU activation**. Note that the approximation error overestimates consequent uncertainty when the antecedent y is uncertain. As y increases, z reaches the asymptote early.

Applying the LSEM to Binary Functions

Any of the 16 binary truth tables can be obtained by a negation pattern of arguments for one of the functions below.

$$
\begin{aligned}
z_{\text{true}}(x_1, x_2) &= \text{logit}(p_{\text{true}}) \\
z_{\text{false}}(x_1, x_2) &= -\text{logit}(p_{\text{true}}) \\
z_1(x_1, x_2) &= x_1 \\
z_2(x_1, x_2) &= x_2 \\
z_{\text{and}}(x_1, x_2) &= \min(x_1, x_2, x_1 + x_2) \\
z_{\text{or}}(x_1, x_2) &= \max(x_1, x_2, x_1 + x_2) \\
z_{\text{xor}}(x_1, x_2) &= \frac{1}{2}\left(|x_1 - x_2| - |x_1 + x_2|\right)
\end{aligned}
$$

We could certainly hard-code a collection of these **unary** and **pairwise activations**, but **can we do better?**

Adaptive N-ary Activation

If antecedent probabilities are independent, then **every higher arity marginalization** has a **simple recursive structure**,

$$p(z) = \sum_{\psi_n=0}^{1} \cdots \left[\sum_{\psi_2=0}^{1} \left[\sum_{\psi_1=0}^{1} p(Z \mid \psi) p(\psi_1) \right] p(\psi_2) \right] \cdots p(\psi_n).$$

This means any higher-arity logic can be built by **composing unary functions**.

Algorithm 2 Adaptive n-ary Activation

Input: $\boldsymbol{Y}$ is a $w_{\text{out}} \times n$ matrix of activation inputs, antecedent logits.
Output width w_{out} and activation arity n are implied by the shape of $\boldsymbol{Y}$.
$\boldsymbol{\Theta}$ is a $w_{\text{out}} \times 2^n$ matrix of activation parameters.

Output: z is a $w_{\text{out}} \times 1$ vector of activation outputs, consequent logits.

1: **function** $z = \text{n\_ary}(\boldsymbol{Y}, \boldsymbol{\Theta})$

Remark: In this algorithm, we index columns of a matrix as

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{y}_1 & \boldsymbol{y}_2 & \cdots & \boldsymbol{y}_n \end{bmatrix} \quad \text{and} \quad \boldsymbol{A}^{(i)} = \begin{bmatrix} \boldsymbol{a}_1^{(i)} & \boldsymbol{a}_2^{(i)} & \cdots & \boldsymbol{a}_{2^{n-i}}^{(i)} \end{bmatrix} \quad \text{for} \quad i \in \{0, 1, ..., n\}.$$

2: Change basis from parameter representations to belief-table logits,

$$\boldsymbol{A}^{(0)} = \boldsymbol{\Theta} \left(\bigotimes_{i=1}^{n} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \right),$$ where $\otimes$ is the Kronecker product.

3: **for** $i = 1, 2, \ldots, n$ **do**
4: **for** $j = 1, 2, \ldots, 2^{n-i}$ **do**
5: $\boldsymbol{a}_j^{(i)} = \text{unary}\left(\boldsymbol{y}_i, \begin{bmatrix} \boldsymbol{a}_{2j-1}^{(i-1)} & \boldsymbol{a}_{2j}^{(i-1)} \end{bmatrix} \right)$
6: **end for**
7: **end for**
8: Return output, $z = \boldsymbol{A}^{(n)}$.
9: **end function**

Okay, so what's this about?

Logical Complexity and Sparsity

With the basic construction, then for 2^n parameters, we can access 2^{2^n} qualitatively distinct functions. For example, 4-ary logic requires **16 parameters,** opening **65,536 truth tables** for each output **within a single layer.**

But if we change the parameter basis, then a powerful property emerges:

$$B = \bigotimes_{i=1}^{n} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \quad \text{so that} \quad A = \Theta B$$

Kronecker product (power)

Theorem 1 ***Irrelevant Antecedents and Parameter Zeros.*** *Let* $\mathbf{p}(\zeta_k \mid \psi_k)$ *represent the* k^{th} *n-ary belief table. Evaluating the tuple of antecedents,* $\psi_k = (\psi_{k1}, \psi_{k2}, \ldots, \psi_{kn})$*, at any vertex of the hypercube,* $\psi_k \in \{\mathbf{0}, \mathbf{1}\}^n$*, gives the probability of the consequent* ζ_k*. Let* $\boldsymbol{a}_k \in \mathbb{R}^{2^n}$ *be the row vector of logit representations, i.e.* $\boldsymbol{a}_{kj} = logit(\mathbf{p}(\zeta_k \mid \psi_k))$ *where the column index* j *maps to* $\psi_k = bits(j)$*, counting from zero. The subset of irrelevant antecedents is defined as*

$$\mathcal{I} = \{\psi_{ki} \mid \mathbf{p}(\zeta_k \mid \psi_k) = \mathbf{p}(\zeta_k \mid (\psi_{k1}, \ldots, \neg\psi_{ki}, \ldots, \psi_{kn})) \quad for\ all \quad \psi_k \in \{\mathbf{0}, \mathbf{1}\}^n\}.$$

Using the change of basis, $\boldsymbol{A} = \boldsymbol{\Theta B}$*,* *if we have* $\psi_{ki} \in \mathcal{I}$ *then* $\boldsymbol{\theta}_{kj} = 0$ *for all* $bit_i(j) = 1$.

Reducing Complexity via Effective Arity

Lower-arity logic embedded in a high-arity activation function retains the same number of nonzeros, $n_{nz} \leq 2^m$, where m is the effective arity. For example, a soft **and** can be parameterized by

Binary Representation

$$a = \begin{bmatrix} -\alpha & -\alpha & -\alpha & \alpha \end{bmatrix} = \alpha \begin{bmatrix} \frac{-1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 \end{bmatrix} = \theta B^{(2)}.$$

Writing the same function using arguments 1 and 3 of a ternary function gives

Ternary Representation

$$a = \alpha \begin{bmatrix} -\frac{1}{2} & \frac{1}{2} & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \end{bmatrix} B^{(3)}$$

Reducing Complexity by Increasing Uncertainty

We can also **reduce complexity** by **increasing consequent uncertainty**. This is **a foundationally-desirable property** for principled uncertainty quantification in automatic abstract learning algorithms.

Table 1: Representations for Selected Binary Activation Functions

k	Operation	n_{nz}	a_{k00}/α	a_{k01}/α	a_{k10}/α	a_{k11}/α	θ_{k1}/α	θ_{k2}/α	θ_{k3}/α	θ_{k4}/α
1	**true**	1	1	1	1	1	1	0	0	0
2	$\mathbf{arg_1}$	1	−1	1	−1	1	0	1	0	0
3	$\mathbf{not_2}$	1	1	1	−1	−1	0	0	−1	0
4	**xor**	1	−1	1	1	−1	0	0	0	−1
5	$\mathbf{relu_1}$	2	0	1	0	1	$1/2$	$1/2$	0	0
6	$\mathbf{relu_{\neg 2}}$	2	1	1	0	0	$1/2$	0	$-1/2$	0
7	$\mathbf{relu_{xor}}$	2	0	1	1	0	$1/2$	0	0	$-1/2$
8	**imply**	4	1	−1	1	1	$1/2$	$-1/2$	$1/2$	$1/2$
9	$\mathbf{imply^*}$	2	0	−1	0	1	0	0	$1/2$	$1/2$
10	**and**	4	−1	−1	−1	1	$-1/2$	$1/2$	$1/2$	$1/2$
11	**or**	4	−1	1	1	1	$1/2$	$1/2$	$1/2$	$-1/2$
12	$\mathbf{and^*}$	2	−1	0	0	1	0	$1/2$	$1/2$	0

Counting zeros, there are actually 3^{2^n} qualitatively distinct functions we can access.

Experiment 1 Setup

Ground Truth:

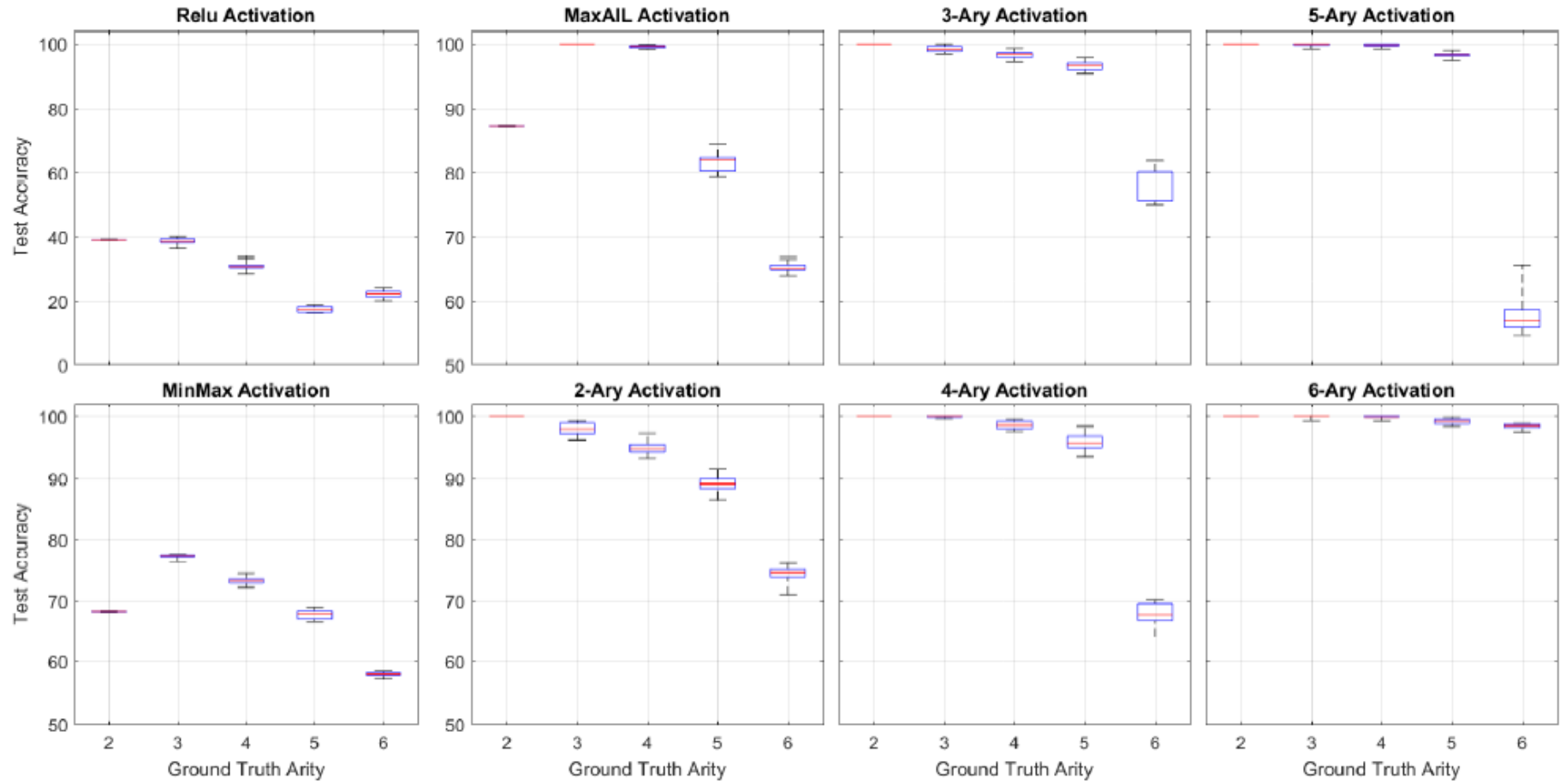
- Each training case consists of a realization of 32 Bernoulli random variables (inputs).
- The ground truth is formed by selecting n antecedents (uniformly) as well as a random truth table.
- We do this for 32 consequents (outputs) independently.
- The dataset consists of applying the ground truth to 6000 input realizations.

Architecture:

- If the effective arity of an activation function is m, then hidden layer ℓ may take up to m^{ℓ} inputs. Thus we need at least $L = \lceil \log_m n \rceil$ hidden layers (including output).
- We keep enough intermediate neurons for each consequent to have an independent learning pathway.

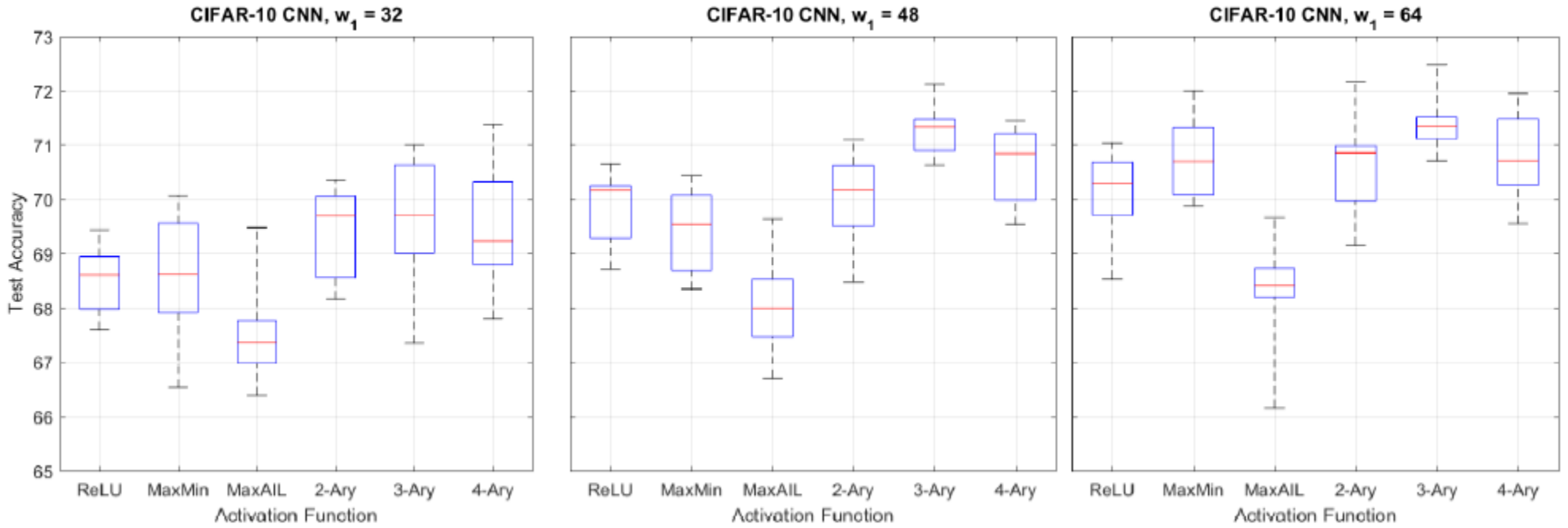
Single-Layer High-Arity Logic

Our activation functions can learn arbitrary logic in a single layer, provided the activation arity is greater than or equal to that of the ground truth



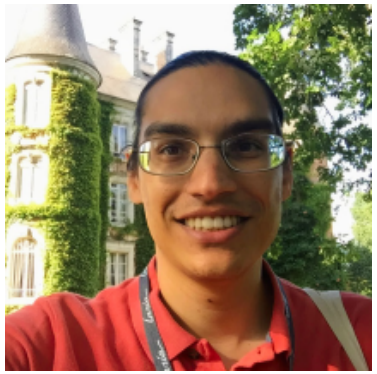
Early Competitive Results

Substituting our activation functions in standard networks with standard training often outperforms other activation functions. **Ternary logic seems to be a good choice.** Yet, we know that **quaternary logic subsumes ternary logic**, thus it remains to be understood **how to efficiently exploit complexity suppression** during training.



Summary

- Belief function provide the **mathematically correct domain** to evaluate **plausible latent relationships** between truth claims.
- High-arity activation functions **allow dimensionality to work in our favor**. Although the parameters increase approximately linearly (for $n \leq 6$), we gain access to $\mathbf{3^{2^n}}$ **qualitatively distinct functions** per channel.
- Our activations are **efficient to evaluate and differentiate**, passing undamped gradient dependencies to relevant antecedents and belief table parameters.
- This framework opens **new pathways to extract sophisticated logic** from data.
- The **concrete connection between sparsity and logical complexity** is needed to build a framework for **complexity-based uncertainty quantification**.


Thank you!

arXiv: 2203.08977

Email: jaduers@sandia.gov