Seminar in Deep Neural Networks

ALG: Math

Sidharth Ramesh 18.05.2021

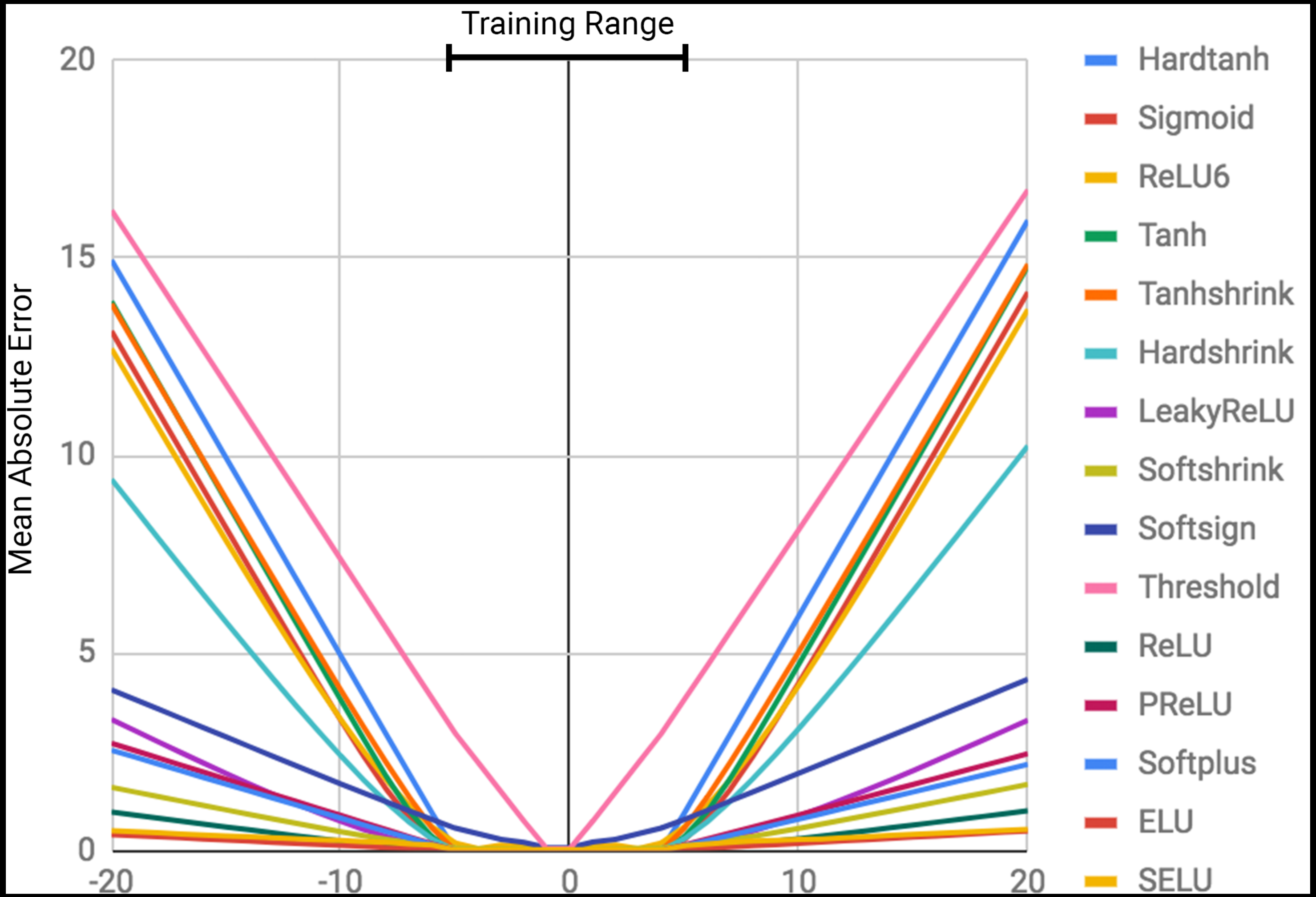
Mentor: Ard Kastrati

Overview

- Introduction
- NALU: Neural Arithmetic Logic Units
- Neural Arithmetic Units
- Neural Power Units
- Neural Status Registers
- LNN
- Discussion

NN fail at Extrapolation

- Scalar identity function
- Autoencoder (same size)



NALU: Neural Arithmetic Logic Units

NAC: The Neural Accumulator, Authers: Andrew Trask et al.

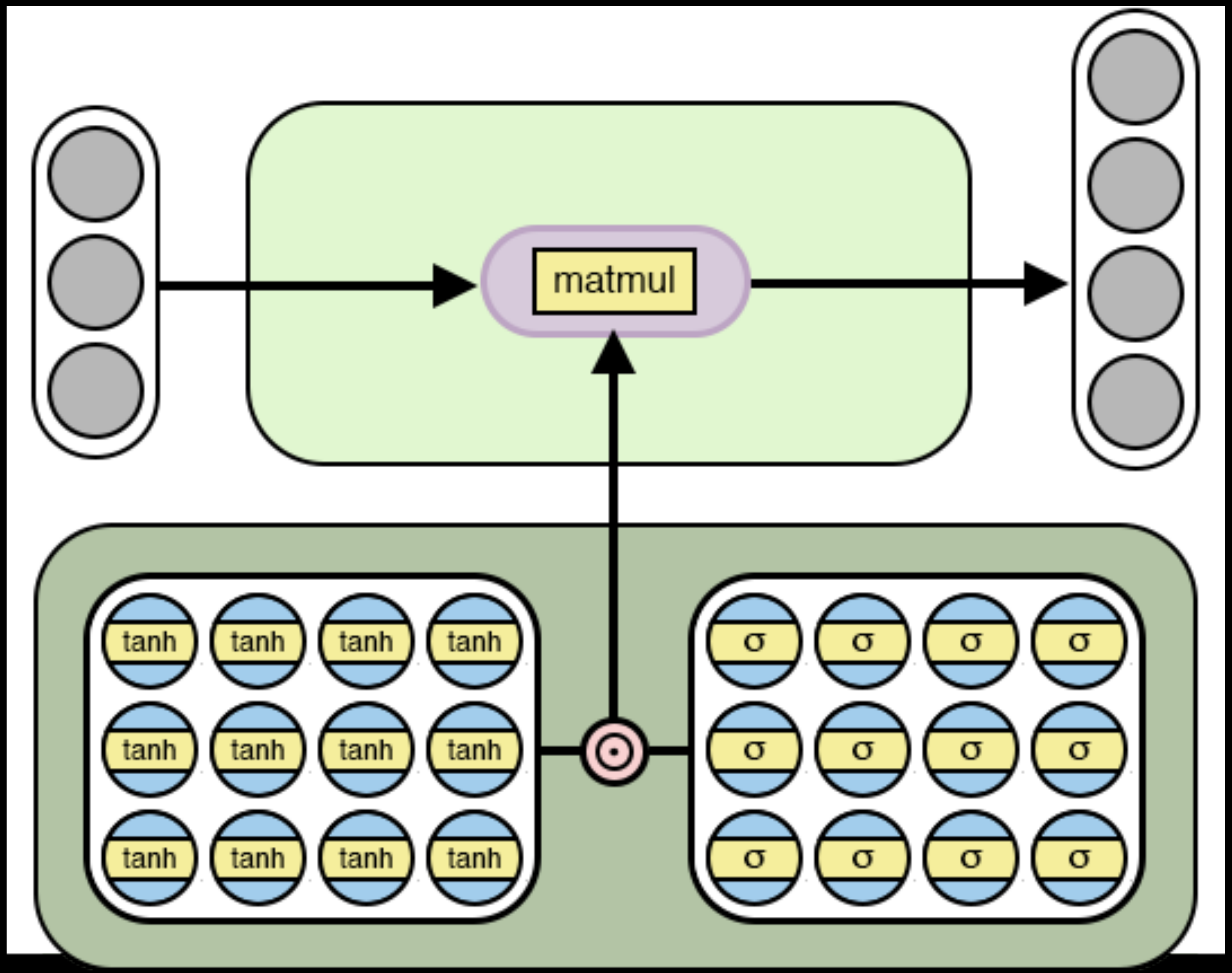
- Introduce inductive bias for linear extrapolation
- Idea:
    - $a = Wx$ (linear layer)
    - Transformation matrix W consists of values $\{-1, 0, 1\}$
    - Introduce a form which is easy to learn with gradient descent

NAC

$$a = Wx$$

$$W = tanh(\hat{W}) \cdot \sigma(\hat{M})$$

Elements in range $[-1,1]$ with bias towards $-1,0,1$



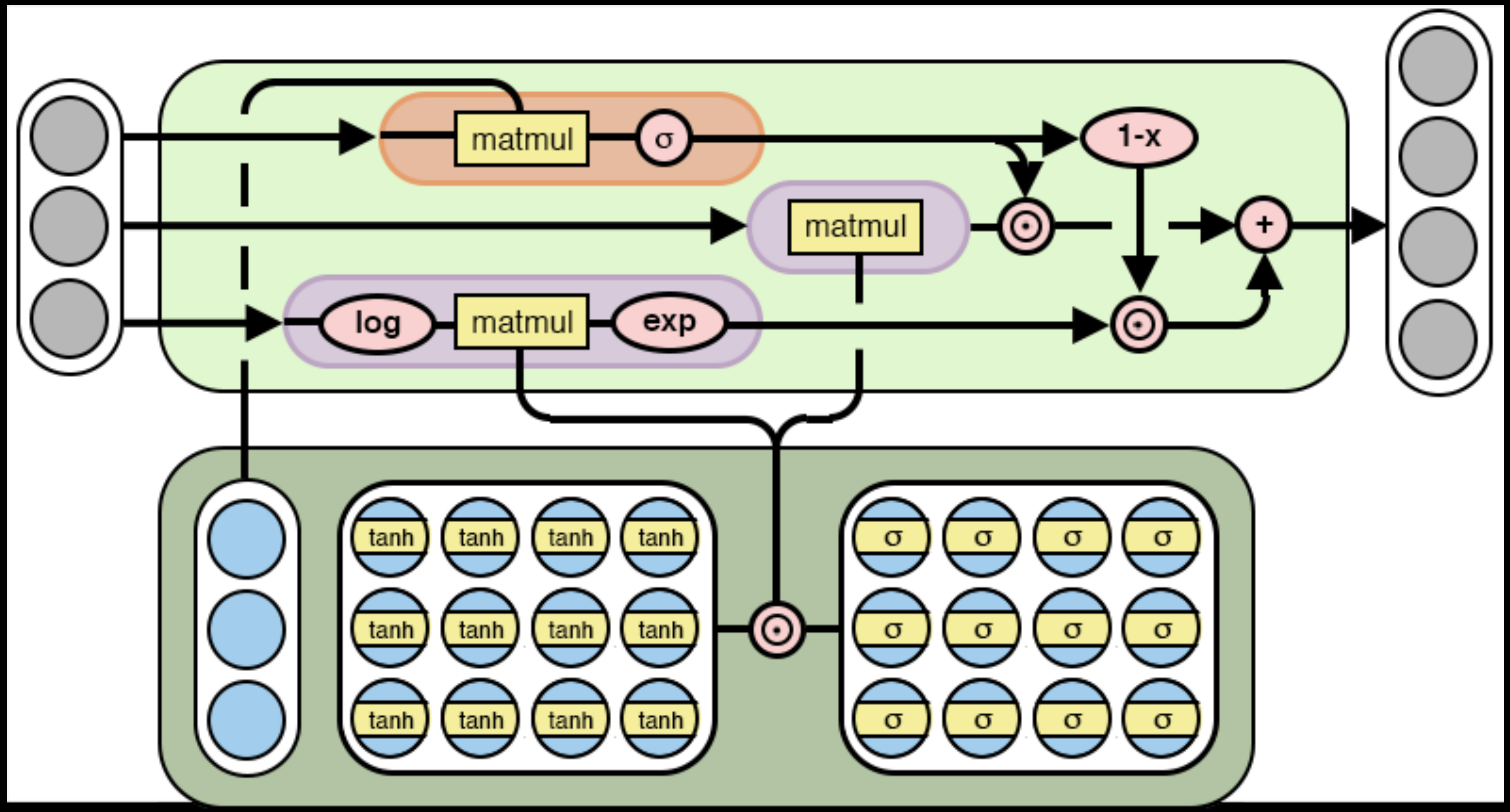
NALU

Idea: Gating between add/sub cell and mul/div cell

$$y = g \cdot a \; + \; (1 - g) \cdot m$$

$$m = \exp W(\log(|x| + \epsilon))$$

$$g = \sigma(Gx)$$



		Static Task (test)				Recurrent Task (test)			
		Relu6	None	NAC	NALU	LSTM	ReLU	NAC	NALU
Interpolation	$a+b$	0.2	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
	$a-b$	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
	$a \times b$	3.2	20.9	21.4	**0.0**	**0.0**	**0.0**	1.5	**0.0**
	a/b	**4.2**	35.0	37.1	5.3	**0.0**	**0.0**	1.2	**0.0**
	a^2	0.7	4.3	22.4	**0.0**	**0.0**	**0.0**	2.3	**0.0**
	$\sqrt{a}$	0.5	2.2	3.6	**0.0**	**0.0**	**0.0**	2.1	**0.0**
Extrapolation	$a+b$	42.6	**0.0**	**0.0**	**0.0**	96.1	85.5	**0.0**	**0.0**
	$a-b$	29.0	**0.0**	**0.0**	**0.0**	97.0	70.9	**0.0**	**0.0**
	$a \times b$	10.1	29.5	33.3	**0.0**	98.2	97.9	88.4	**0.0**
	a/b	37.2	52.3	61.3	**0.7**	**95.6**	863.5	>999	>999
	a^2	47.0	25.1	53.3	**0.0**	98.0	98.0	123.7	**0.0**
	$\sqrt{a}$	10.3	20.0	16.4	**0.0**	95.8	34.1	>999	**0.0**

Neural Arithmetic Units

Improving upon NALU, Authors: Andreas Madsen and Alexander Rosenberg Johansen

- NALU doesn’t support negative values or large hidden input-size
- Improving NAC_+ and $NAC_\cdot$ based on a theoretical analysis
  - Simplification of the weight matrix ($W = tanh(\hat{W}) \cdot \sigma(\hat{M})$)
  - Sparsity regulariser
  - NAU (neural addition unit) and NMU (neural multiplication unit)

Neural Arithmetic Units

Expectation of the gradient

- Glorot & Bengio, 2010: $E[z_{h_l}] = 0$ is desired

- In NALU this leads to $E[tanh(W_{h_{l-1},h_l})] = 0$

  Reminder: $W_{h_{l-1},h_l} = tanh(\hat{W}_{h_{l-1},h_l}) \cdot \sigma(\hat{M})$

- Causes the expectation of the gradient to be zero

$$E\left[\frac{\delta \mathscr{L}}{\delta \hat{M}_{h_{l-1},\, h_l}} \right] = E\left[\frac{\delta \mathscr{L}}{\delta W_{h_{l-1},\, h_l}} \right] \cdot E[\ \tanh(\ \hat{W}_{h_{l-1},\, h_l})\ ] \cdot E[\ \sigma'(\ \hat{M}_{h_{l-1},\, h_l}\ )\ ] = 0$$

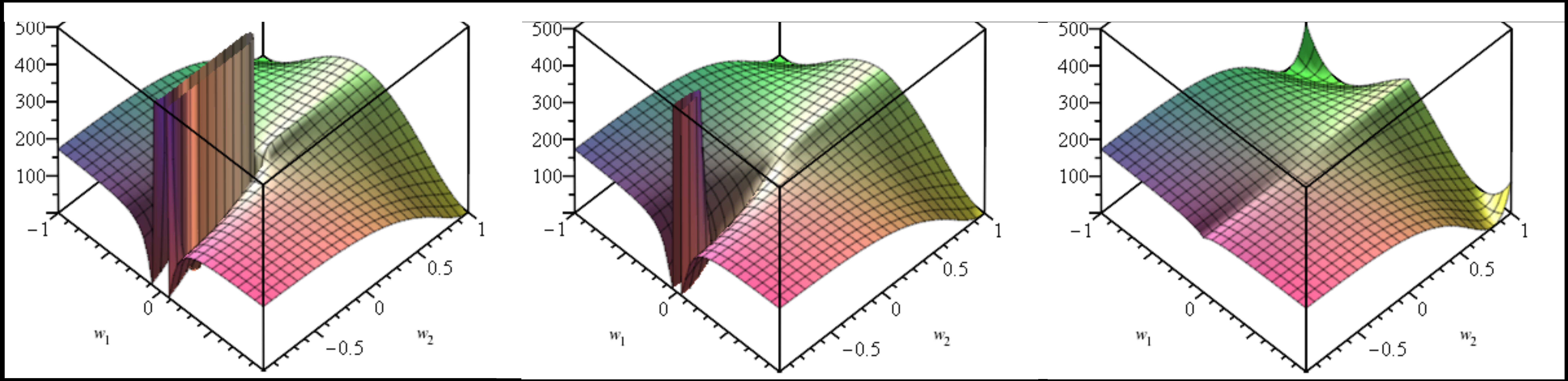
Neural Addition Unit

- Simplified Weight Matrix
  - $W_{h_{l-1},h_l} = min(max(W_{h_{l-1},h_l}, -1),1)$ clamping the elements to [-1,1]
- Sparsity regulariser
  - $\mathcal{R}_{l,sparse} = \frac{1}{(H_l \cdot H_{l-1})} \sum_{h_l=1}^{H_l} \sum_{h_{l-1}=1}^{H_{l-1}} min(|W_{h_{l-1},h_l}|, 1 - |W_{h_{l-1},h_l}|)$
  - $\mathcal{L} = \hat{\mathcal{L}} + \lambda_{sparse} \mathcal{R}_{l,sparse}$
- NAU: $z_{h_l} = \sum_{h_{l-1}=1}^{H_{l-1}} W_{h_l,h_{l-1}} z_{h_{l-1}}$

Challenges of Division

- $m = \exp W(\log(|x| + \epsilon))$

- Small x, the output explodes



$NAC_{\cdot}$ with $\epsilon = 10^{-7}$ | $NAC_{\cdot}$ with $\epsilon = 0.1$ | $NAC_{\cdot}$ with $\epsilon = 1$

Neural Multiplication Unit

- $W_{h_{l-1},h_l} = min(max(W_{h_{l-1},h_l}, 0), 1)$
- $\mathcal{R}_{l,sparse} = \frac{1}{(H_l \cdot H_{l-1})} \sum_{h_l=1}^{H_l} \sum_{h_{l-1}=1}^{H_{l-1}} min(|W_{h_{l-1},h_l}|, 1 - |W_{h_{l-1},h_l}|)$
- NMU: $z_{h_l} = \prod_{h_{l-1}=1}^{H_{l-1}} W_{h_l,h_{l-1}} z_{h_{l-1}} + 1 - W_{h_l,h_{l-1}}$

Op	Model	Success Rate	Solved at iteration step Median	Solved at iteration step Mean	Sparsity error Mean
$\times$	$\mathrm{NAC}_{\bullet}$	$31\%\,^{+10\%}_{-8\%}$	$2.8\cdot10^{6}$	$3.0\cdot10^{6}\,^{+2.9\cdot10^{5}}_{-2.4\cdot10^{5}}$	$5.8\cdot10^{-4}\,^{+4.8\cdot10^{-4}}_{-2.6\cdot10^{-4}}$
	NALU	$0\%\,^{+4\%}_{-0\%}$	—	—	—
	NMU	$\mathbf{98\%}\,^{+1\%}_{-5\%}$	$\mathbf{1.4\cdot10^{6}}$	$\mathbf{1.5\cdot10^{6}}\,^{+5.0\cdot10^{4}}_{-6.6\cdot10^{4}}$	$\mathbf{4.2\cdot10^{-7}}\,^{+2.9\cdot10^{-8}}_{-2.9\cdot10^{-8}}$
$+$	NAC_{+}	$\mathbf{100\%}\,^{+0\%}_{-4\%}$	$2.5\cdot10^{5}$	$4.9\cdot10^{5}\,^{+5.2\cdot10^{4}}_{-4.5\cdot10^{4}}$	$2.3\cdot10^{-1}\,^{+6.5\cdot10^{-3}}_{-6.5\cdot10^{-3}}$
	Linear	$\mathbf{100\%}\,^{+0\%}_{-4\%}$	$6.1\cdot10^{4}$	$\mathbf{6.3\cdot10^{4}}\,^{+2.5\cdot10^{3}}_{-3.3\cdot10^{3}}$	$2.5\cdot10^{-1}\,^{+3.6\cdot10^{-4}}_{-3.6\cdot10^{-4}}$
	NALU	$14\%\,^{+8\%}_{-5\%}$	$1.5\cdot10^{6}$	$1.6\cdot10^{6}\,^{+3.8\cdot10^{5}}_{-3.3\cdot10^{5}}$	$1.7\cdot10^{-1}\,^{+2.7\cdot10^{-2}}_{-2.5\cdot10^{-2}}$
	NAU	$\mathbf{100\%}\,^{+0\%}_{-4\%}$	$\mathbf{1.8\cdot10^{4}}$	$3.9\cdot10^{5}\,^{+4.5\cdot10^{4}}_{-3.7\cdot10^{4}}$	$\mathbf{3.2\cdot10^{-5}}\,^{+1.3\cdot10^{-5}}_{-1.3\cdot10^{-5}}$
$-$	NAC_{+}	$\mathbf{100\%}\,^{+0\%}_{-4\%}$	$9.0\cdot10^{3}$	$3.7\cdot10^{5}\,^{+3.8\cdot10^{4}}_{-3.8\cdot10^{4}}$	$2.3\cdot10^{-1}\,^{+5.4\cdot10^{-3}}_{-5.4\cdot10^{-3}}$
	Linear	$7\%\,^{+7\%}_{-4\%}$	$3.3\cdot10^{6}$	$1.4\cdot10^{6}\,^{+7.0\cdot10^{5}}_{-6.1\cdot10^{5}}$	$1.8\cdot10^{-1}\,^{+7.2\cdot10^{-2}}_{-5.8\cdot10^{-2}}$
	NALU	$14\%\,^{+8\%}_{-5\%}$	$1.9\cdot10^{6}$	$1.9\cdot10^{6}\,^{+4.4\cdot10^{5}}_{-4.5\cdot10^{5}}$	$2.1\cdot10^{-1}\,^{+2.2\cdot10^{-2}}_{-2.2\cdot10^{-2}}$
	NAU	$\mathbf{100\%}\,^{+0\%}_{-4\%}$	$\mathbf{5.0\cdot10^{3}}$	$\mathbf{1.6\cdot10^{5}}\,^{+1.7\cdot10^{4}}_{-1.6\cdot10^{4}}$	$\mathbf{6.6\cdot10^{-2}}\,^{+2.5\cdot10^{-2}}_{-1.9\cdot10^{-2}}$

Neural Power Units

Authors: Niklas Heim, Tomas Pevny, Vaclav Smidl

- Expands Neural Arithmetic Logic Units to operate on full domain of real numbers
- Adds capability to learn any arbitrary power function (therefore also square root and division)
- Improve convergence by introducing a relevance gate
- Highly transparent model

NaiveNPU

$$m = \exp W(\log(|x| + \epsilon))$$

Idea: use complex log and allow W to be complex as well

- $y = \exp(W \log_{complex}(x)) = \exp((W_r + iW_i)\ \log_{complex}(x))$
- Allowing W to be complex results in complex gradients which effectively doubles the number of parameters
- We only consider the real part of y
  - $y = \exp(W_r \log(r) - \pi W_i k) \cdot \cos(W_i \log(r) + \pi W_r\ k)$

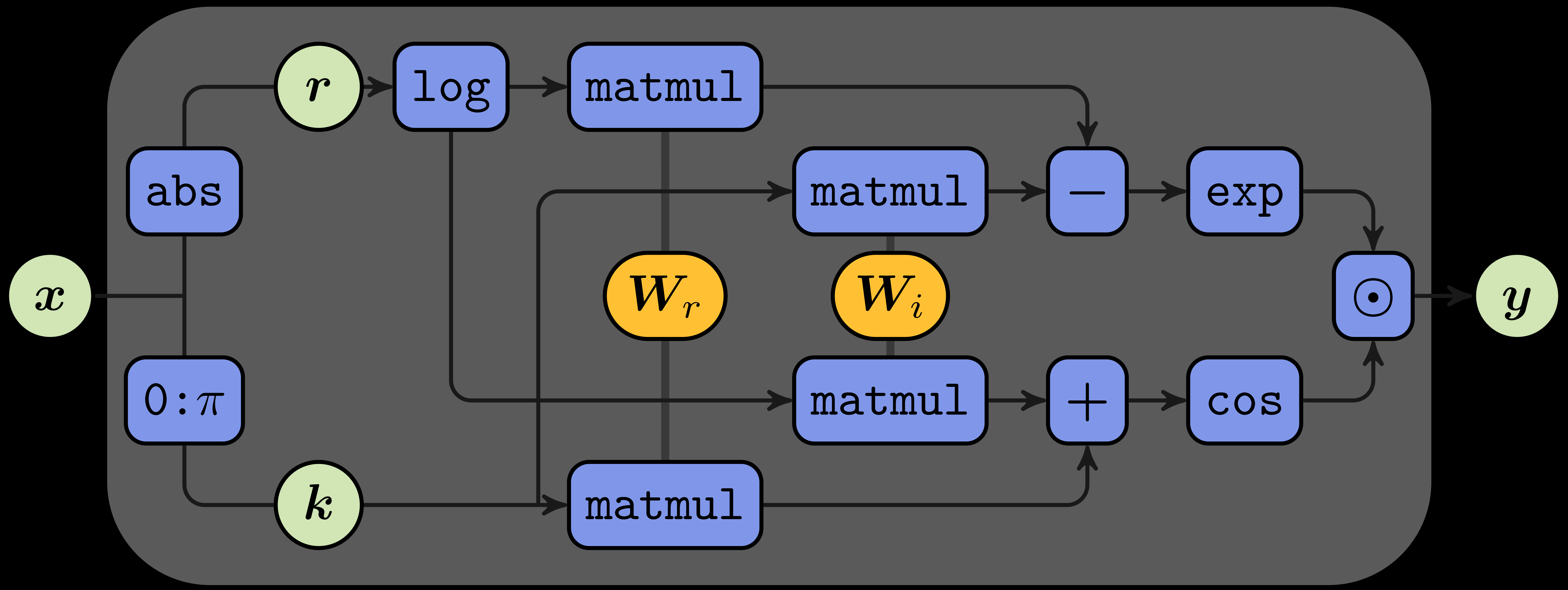
Definition NaiveNPU

The naive Neural Power Unit, with Matrices W_r and W_i representing real and imaginary part of the complex number, is defined as:

$$y = exp(W_r \log(r) - \pi W_i k) \cdot \cos(W_i \log(r) + \pi W_r k)$$, where

$$r = |x| + \epsilon \, , \qquad k_i = \begin{cases} 0 \; if \; x_i \leq 0 \\ 1 \; if \; x_i > 0 \end{cases}$$

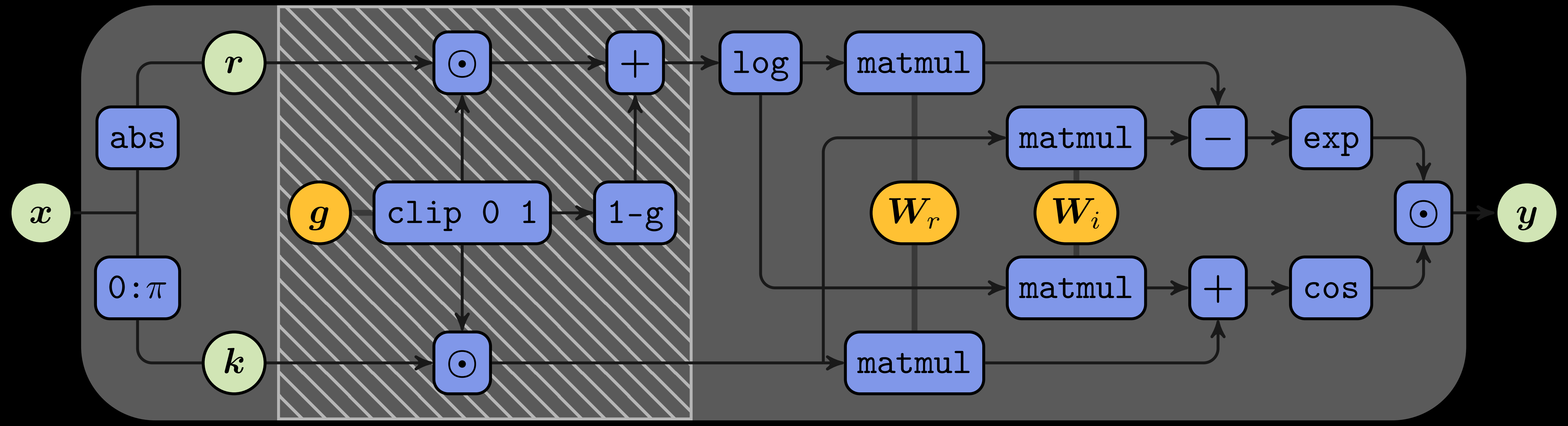
with inputs x, machine epsilon ϵ and learn parameters W_r and W_i .



NaiveNPU diagram, with input x and output y. Vectors in green, trainables in orange, functions in blue

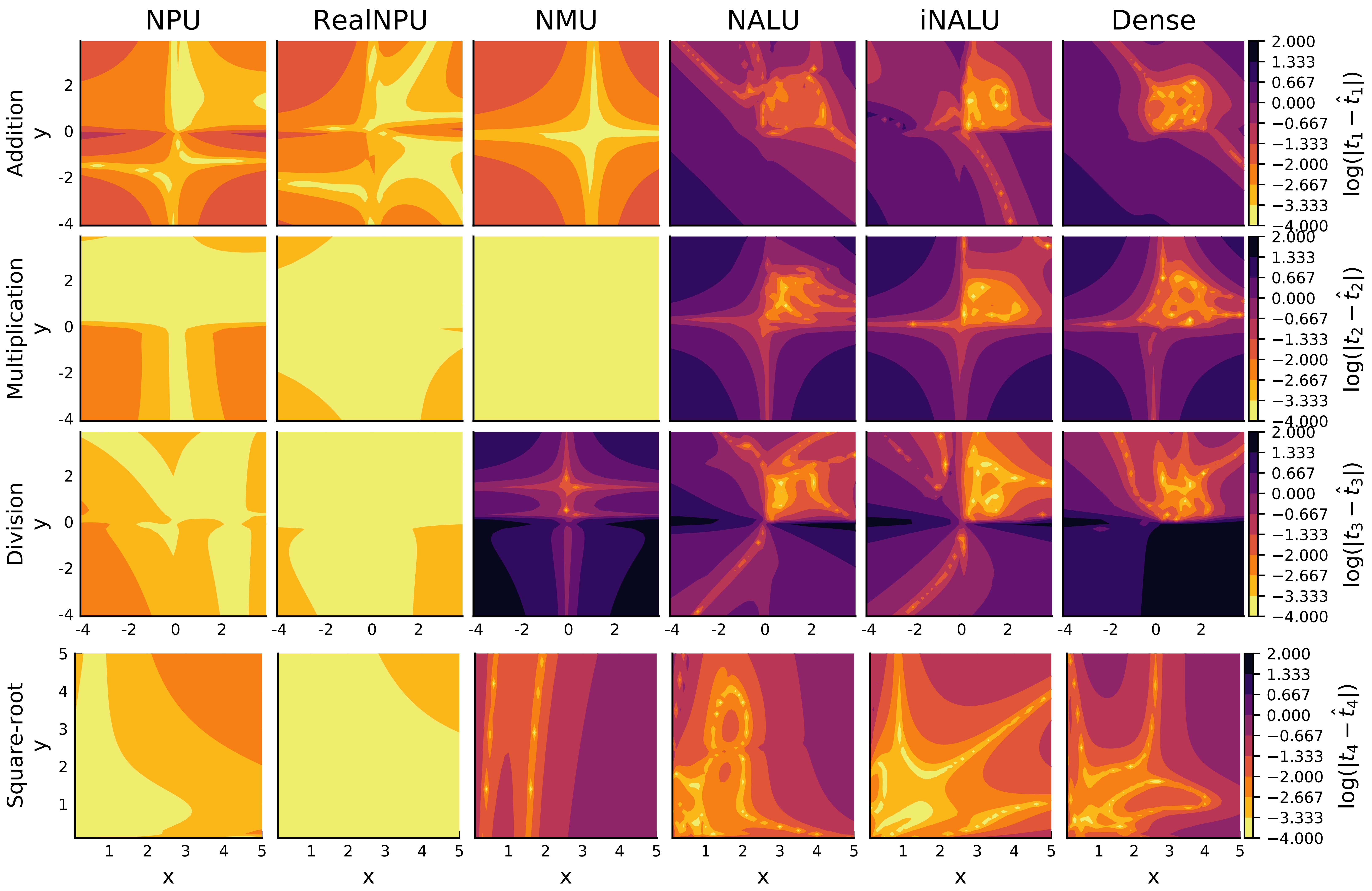
Relevance Gate

- NaiveNPU has difficulties converging on large scale tasks
- If an input x_i is close to zero (i.e irrelevant, the whole row will be zero —> we lose gradient information for all other inputs x_i



$$r = \hat{g} \cdot (|x| + \epsilon) + (1 - \hat{g})\,, \qquad k_i = \begin{cases} 0 \;\; if \; x_i \leq 0 \\ \hat{g}_i \;\; if \; x_i > 0 \end{cases}, \qquad \hat{g}_i = min(max(g_i, 0), 1)$$

$$f(x, y) = (x + y, xy, x/y, \sqrt{x})^T =: t$$

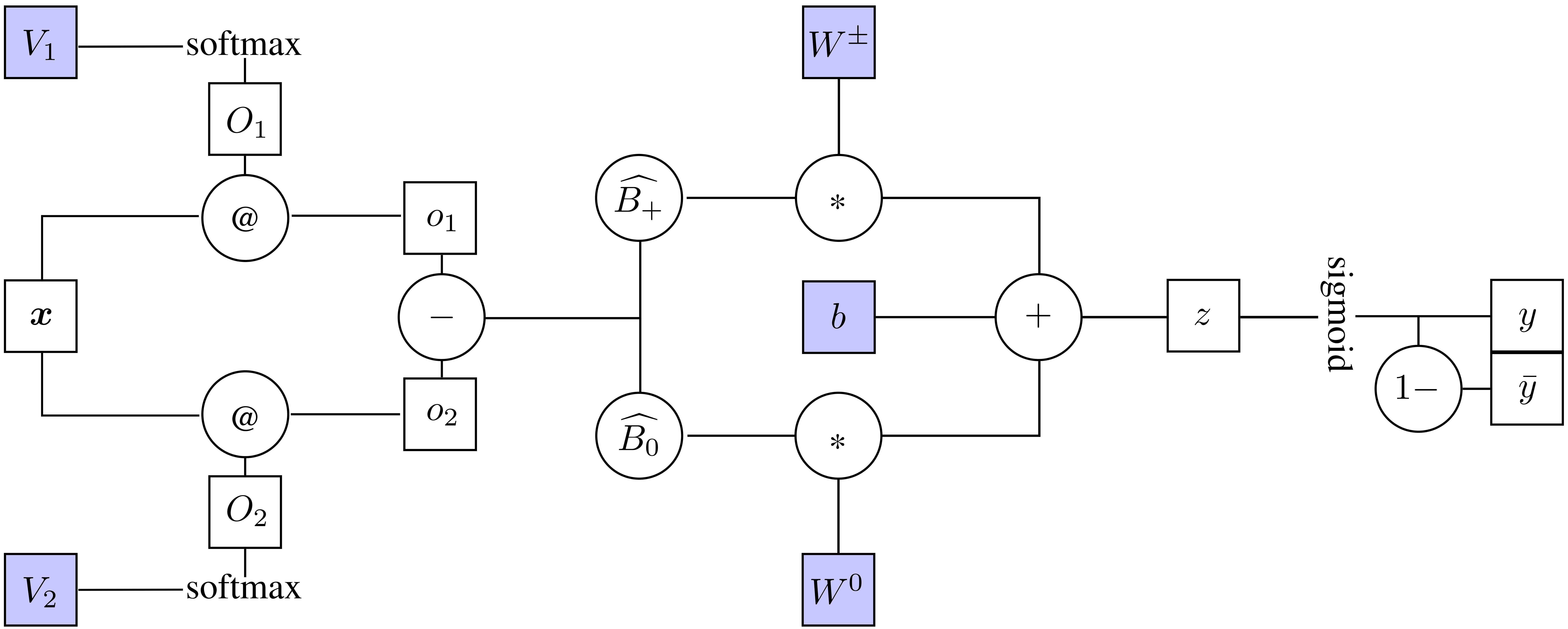


Neural Status Registers

Authors: Lukas Faber and Roger Wattenhofer

- The before mentioned architectures deal with extrapolation
- Quantitative Reasoning —> (if and while)
- Inspired by ALU (Arithmetic Logic Units)
  - Computes difference of 2 inputs
    - Difference is positive —> sign bit B_+ is set
    - Difference is zero —> zero bit B_0 is set
  - Combining these bits with logical operations (&, /, !) we can perform comparisons such as ($>$, $<$, $\leq$, $\geq$, $=$, $\neq$)

High-Level NSR architecture



Continous Relaxation and Floating Point Comparison

1. $\frac{\delta y}{\delta b} = y(1 - y)$
2. $\frac{\delta y}{\delta W^{+}} = y(1 - y)B_{+}$
3. $\frac{\delta y}{\delta W^{0}} = y(1 - y)B_{0}$
4. $\frac{\delta y}{\delta O_{1}} = y(1 - y)(B_{+}^{'}W^{+} \ + \ B_{0}^{'}W^{0})$
5. $\frac{\delta y}{\delta O_{2}} = - y(1 - y)(B_{+}^{'}W^{+} \ + \ B_{0}^{'}W^{0})$

- The Values B_{+} and B_{0} can take on the value 0. Results in zero gradients for Equation (2) & (3)
    - Adjust them:

$$\hat{B}_{+} = tanh(x) \quad \text{and } \hat{B}_{0} = 1 - 2(tanh(x))^{2}$$

- Difference $x = 0.5$ then $\hat{B}_{0} = 0.57$ which is incorrect
    - We fix this by introducing a hyper parameter λ

$$\hat{B}_{+\lambda} = tanh(\lambda x) \quad \text{and} \quad \hat{B}_{0\lambda} = 1 - 2(tanh(\lambda x))^{2}$$

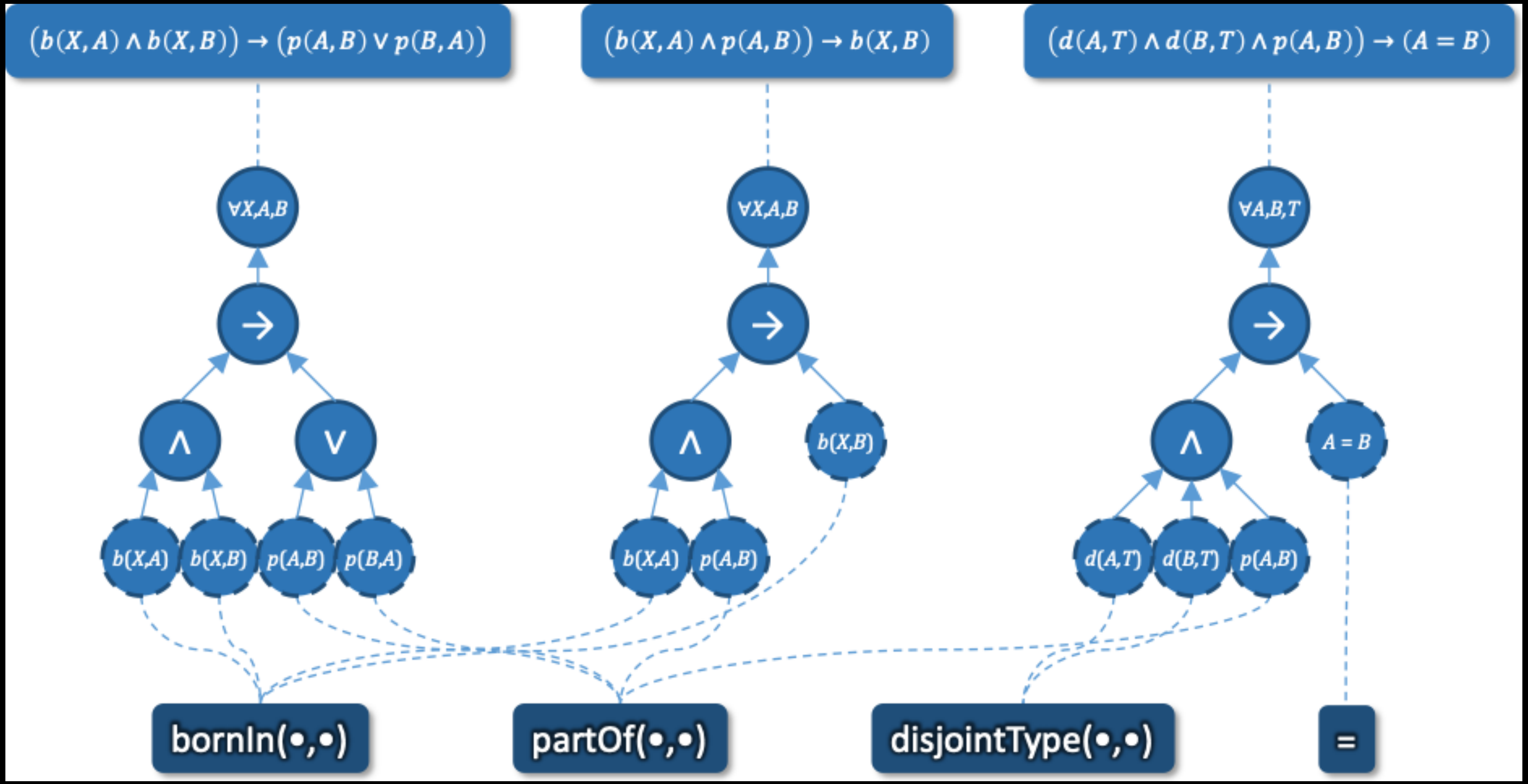
Learning comparisons on data from $[-10; 9]$

Task	Model	Train	10^2	10^3	10^4	10^5	10^6	10^7	10^8	10^9	10^{10}	10^{11}	10^{12}	10^{13}
MLP	$>$	**1.0**	**1.0**	0.96	0.71	0.49	0.48	0.48	0.48	0.48	0.48	0.48	0.48	0.48
	$<$	**1.0**	**1.0**	0.93	0.7	0.49	0.48	0.48	0.48	0.48	0.48	0.48	0.48	0.48
	$\geq$	**1.0**	**1.0**	0.94	0.71	0.49	0.48	0.48	0.48	0.48	0.48	0.48	0.48	0.48
	$\leq$	**1.0**	**1.0**	0.94	0.68	0.49	0.48	0.48	0.48	0.48	0.48	0.48	0.48	0.48
	$=$	0.95	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
	$\neq$	0.95	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
NSR (ours)	$>$	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
	$<$	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
	$\geq$	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
	$\leq$	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
	$=$	**0.99**	**0.93**	**0.93**	**0.93**	**0.93**	**0.93**	**0.93**	**0.93**	**0.93**	**0.93**	**0.93**	**0.93**	**0.93**
	$\neq$	**0.99**	**0.92**	**0.92**	**0.92**	**0.92**	**0.92**	**0.92**	**0.92**	**0.92**	**0.92**	**0.92**	**0.92**	**0.92**

LNN: Logical Neural Networks

Authors: Ryan Riegel, Alexander Gray, et al.

- This is not your standard Neural Network
- LNN are able to do neural network-style ***learning*** and classical AI-style ***reasoning***
- Neurons model a weighted real-valued logic



LNN: Logical Neural Network

Main Ideas

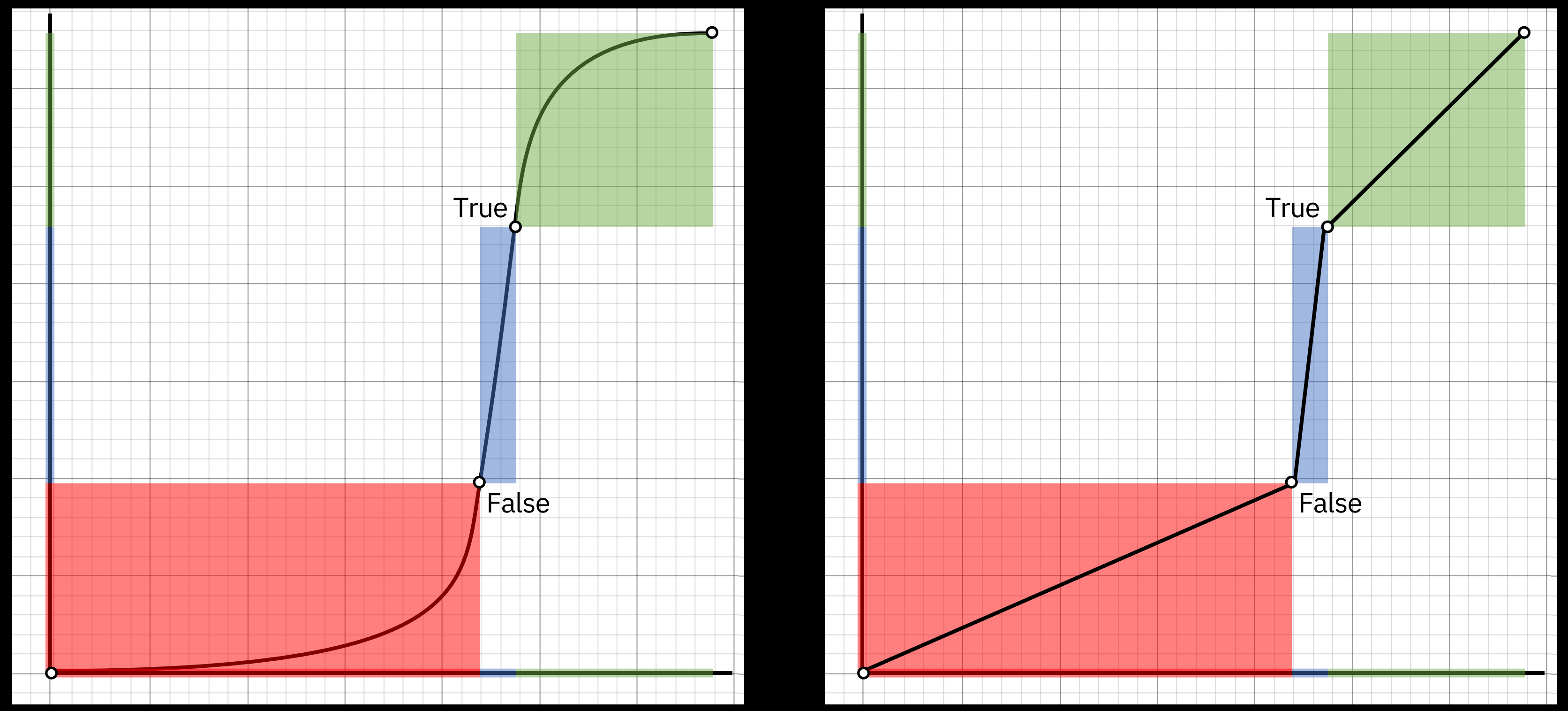
1. Logical Constraints

2. Bounds on Truth Values

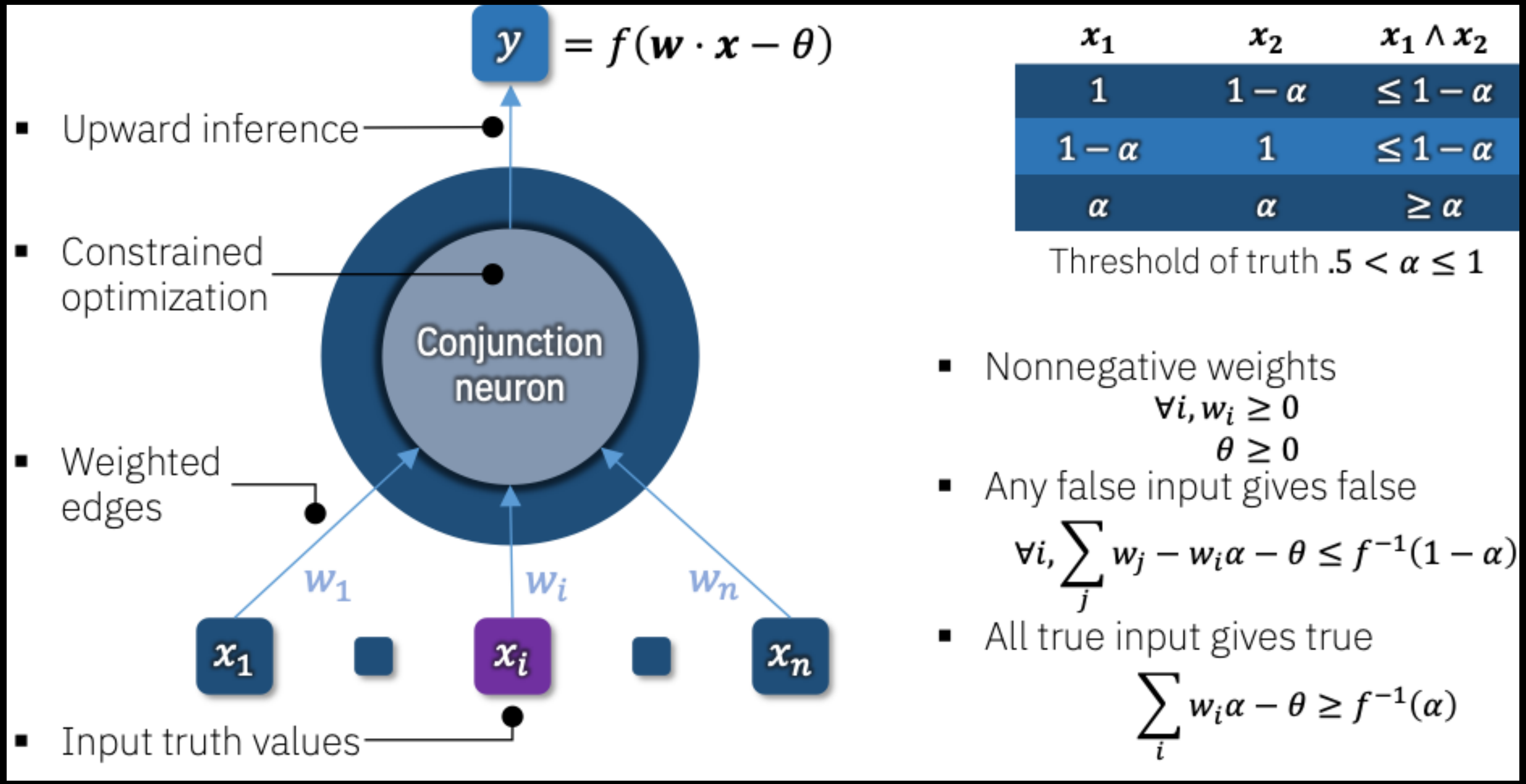
3. Omnidirectional Inference

Main Idea #1: Logical Constraints

- Each neuron should behave like logical gate
- Constrain neural parameters to achieve this behaviour
- Conjunction and negation (1-x) we can implement all other logic
- The choice of *f* affects the logic
- Converges to classical inference behaviour



A logistic activation function and a linearly interpolated activation function

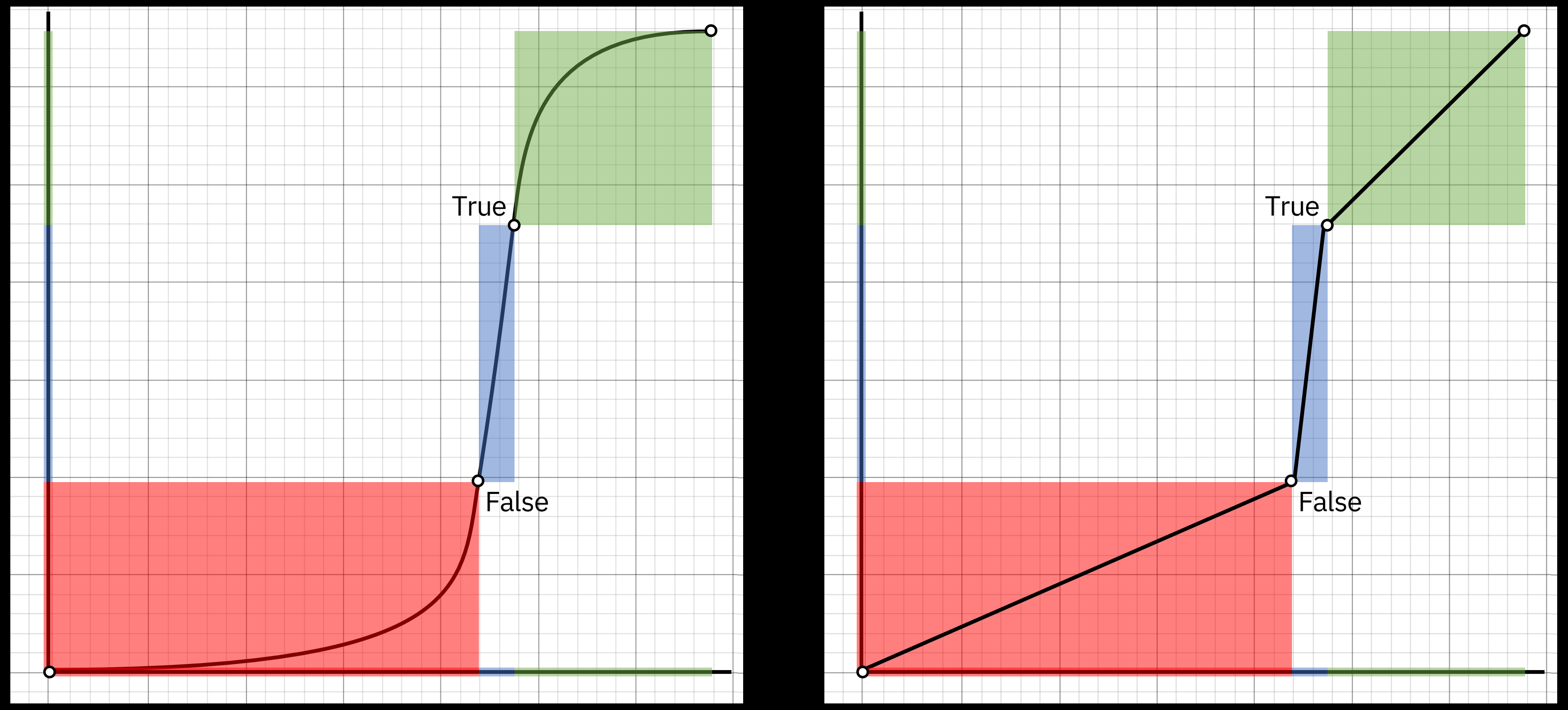


Most common real-valued logics

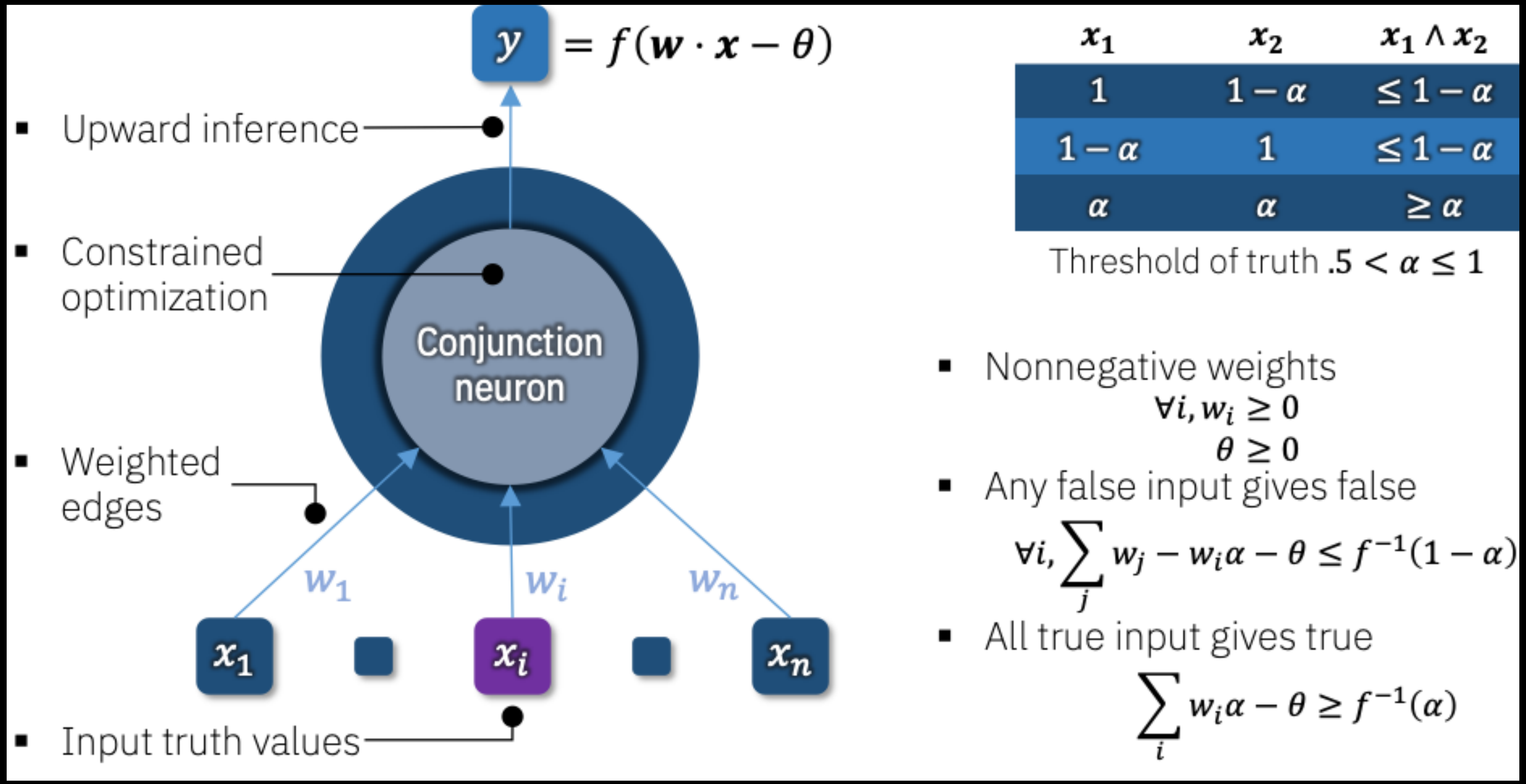
Logic	T-Norm (AND) $a \otimes b$	T-conorm (OR) $a \oplus b$	Residuum (IMPLIES) $a \to b$
Gödel	$\min\{a, b\}$	$\max\{a, b\}$	b if $a < b$ else 1
Product	$a \cdot b$	$a + b - a \cdot b$	$\frac{b}{a}$ if $a < b$ else 1
Łukasiewicz	$\max\{0,\ a + b - 1\}$	$\min\{1,\ a + b\}$	$\min\{1,\ 1 - a + b\}$

Main Idea #1: Logical Constraints

- Each neuron should behave like logical gate
- Constrain neural parameters to achieve this behaviour
- Conjunction and negation (1-x) we can implement all other logic
- The choice of *f* affects the logic
- Converges to classical inference behaviour



A logistic activation function and a linearly interpolated activation function



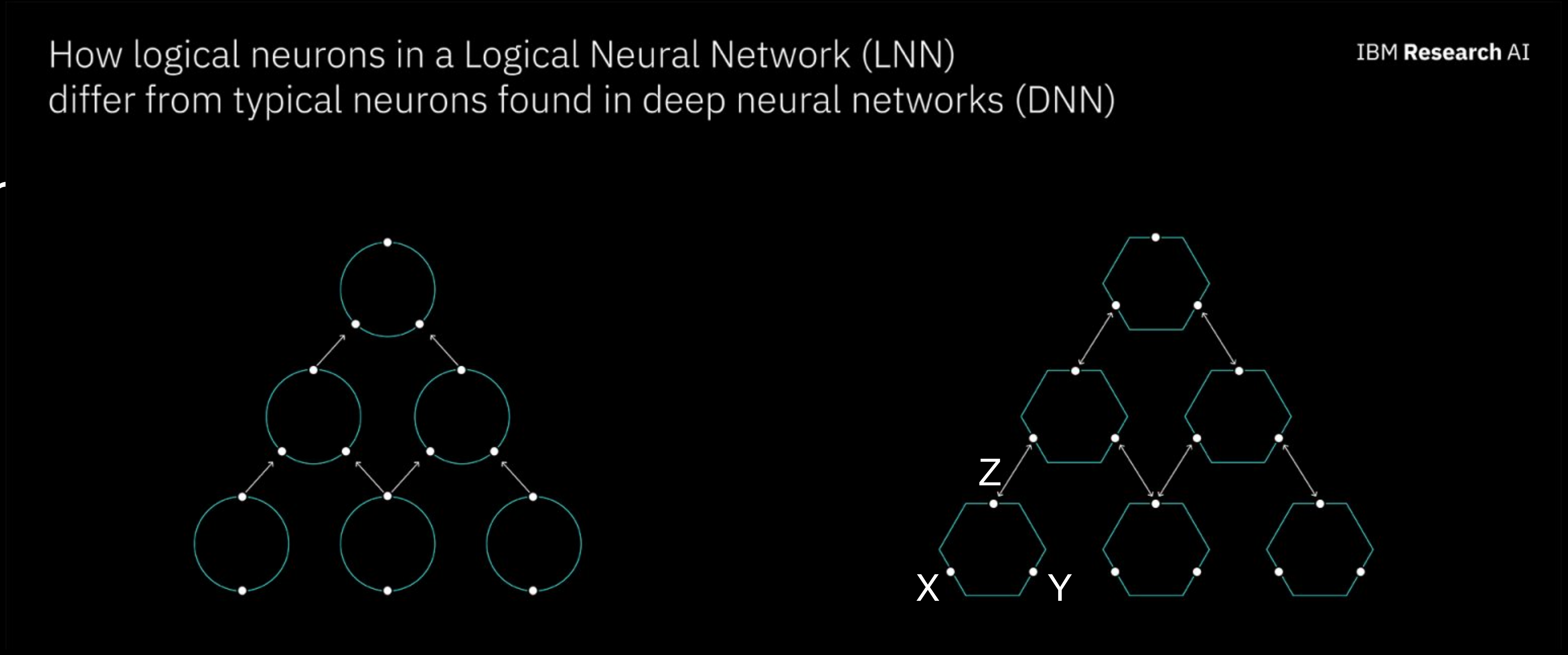
Main Idea #2

Bounds on Truth Values

- Represent truth value with a lower and upper bound
- Allows open-world assumption
- Other neuro-symbolic methods assume that the truth value can be known
- Unknown $(L = 0, U = 1)$, Contradiction $(L > U)$, Ambiguity $(L = U = 0.5)$

Main Idea #3
Omnidirectional Inference

- Use f^{-1} to allow inference in any directior
- Upward and downward algorithm = "feed-forward"
    - Upward ALG does normal evaluation (AND, OR, etc)
    - Downward ALG enables inference rules such as *modus ponens (x, x->y |- y)*



Example: $x \wedge y = z$

Upward : $x = y = 1$, then $z = 1$

Downward: $z = 0$ and $y = 1$, then $x = 0$

LNN: Logical Neural Network

Learning

$$\min_{B,W} \quad E(B,W) + \sum_{k\in N} \max\{0, L_{B,W,k} - U_{B,W,k}\}$$

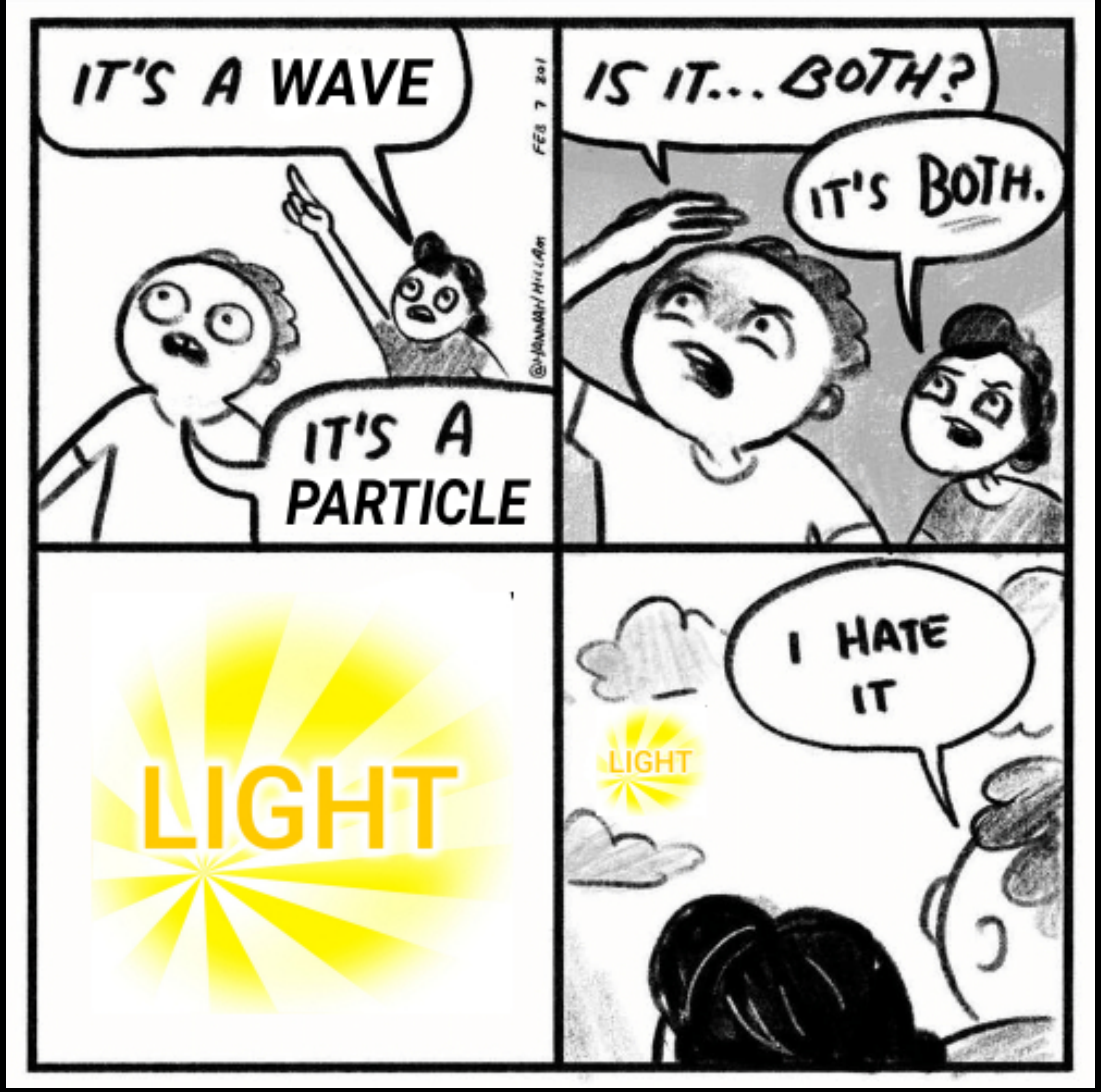
$$\text{s.t.} \quad \forall k \in N,\ i \in I_k, \qquad \alpha \cdot w_{ik} - \beta_k + 1 \geq \alpha, \qquad w_{ik} \geq 0 \tag{6}$$

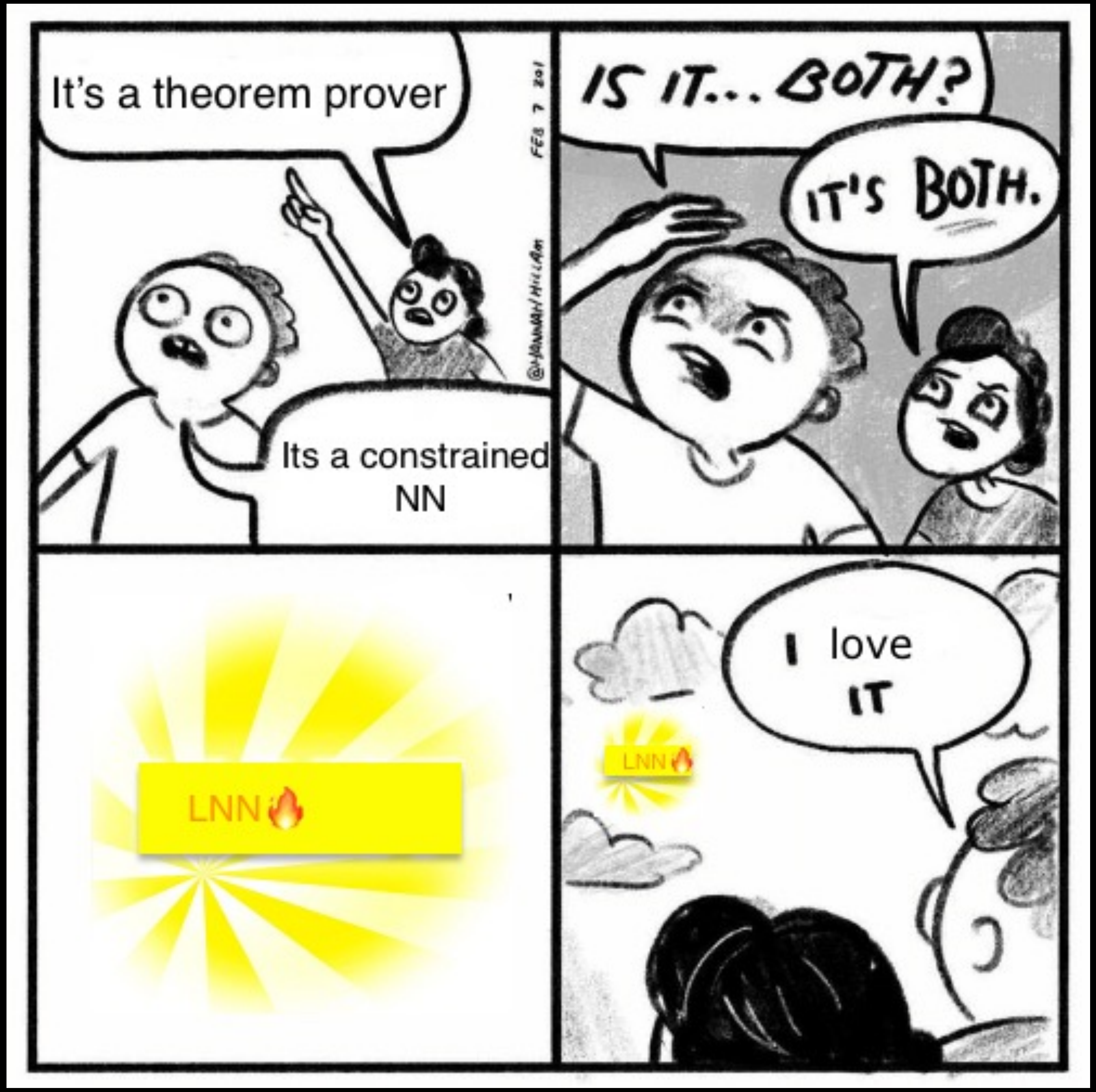
$$\forall k \in N, \qquad \sum_{i\in I_k} (1-\alpha) \cdot w_{ik} - \beta_k + 1 \leq 1 - \alpha, \qquad \beta_k \geq 0 \tag{7}$$

- Incorporate Contradiction Term to the Loss Function

- Standard Backpropagation updates weights (importance of input)

LNN are transparent, interpretable and decomposable!





References

- Trask et al. 2018. “Neural Arithmetic Logic Units”. arXiv:1808.00508.
- Madsen and Johansen. 2020. “Neural Arithmetic Units”. arXiv:2001.05016.
- Heim, Pevny and Smidl. 2020. “Neural Power Units”. arXiv:2006.01681.
- Faber and Wattenhofer. 2020. “Neural Status Registers”. arXiv:2004.07085.
- Riegel et al. 2020. “Logical Neural Networks”. arXiv:2006.13155