Łukasiewicz Logic with Actions for Neural Networks training

Ioana Leuștean[a] Bogdan Macovei[a,b]

[a]Faculty of Mathematics and Computer Science, University of Bucharest
[b]Research Center for Logic, Optimization and Security (LOS)

FROM 2025
September 18, 2025

Overview

1. Motivation
2. Background on Łukasiewicz Logic
3. Hybrid Modal Logic Framework
4. Implementation in Lean
5. Related work & Conclusions

Motivation

- neural networks are powerful tools, but they are black boxes;
- our goal is to represent the **training process** as logical deduction, in order to verify properties:
  - we represent the multi-layer perceptron as a logical formula;
  - we represent the actions of the training process as modal operators;
- we implement this system in **Lean 4**.

Historical Context

Figure: Jan Łukasiewicz

- 1920s: J. Łukasiewicz defines 3-valued logics.
- 1930: extended to n-valued and ∞-valued (with Tarski).
- we denote the ∞-valued Łukasiewicz as Łuk_∞: truth values in $[0, 1]$.

Łukasiewicz Logic

- the logical connectives are *implication* ($\rightarrow_L$) and *negation* ($\neg_L$)
- $\neg_L x := 1 - x$ and $x \rightarrow_L y := \min(1 - x + y, 1)$, for any $x, y \in [0, 1]$;
- the axioms are:

(L1) $\varphi \rightarrow_L (\psi \rightarrow_L \varphi)$
(L2) $(\varphi \rightarrow_L \psi) \rightarrow_L ((\psi \rightarrow \chi) \rightarrow_L (\varphi \rightarrow_L \chi))$
(L3) $(\varphi \rightarrow_L \psi) \rightarrow_L \psi) \rightarrow_L (\psi \rightarrow_L \varphi) \rightarrow_L \varphi)$
(L4) $(\neg\psi \rightarrow_L \neg\varphi) \rightarrow_L (\varphi \rightarrow_L \psi)$

MV-algebra

Definition (MV-algebra)

An MV-algebra is a structure $(A, \oplus, ^*, 0)$ such that:

- $(A, \oplus, 0)$ is an abelian monoid;
- the following properties are satisfied:

$$\begin{array}{ll} (MV_1) & (x^*)^* = x \\ (MV_2) & (0^*) \oplus a = 0^* \\ (MV_3) & (x^* \oplus y)^* \oplus y = (y^* \oplus x)^* \oplus x \end{array}$$

We define in any MV-algebra the auxiliary operations, for any $x, y \in A$:

$$\begin{array}{lll} 1 := 0^* & x \odot y := (x^* \oplus y^*)^* & x \to y := x^* \oplus y \\ x \ominus y = x \odot \neg_L y & x \vee y := (x \odot y^*) \oplus y & x \wedge y := (x \oplus y^*) \odot y \end{array}$$

Riesz MV-algebra

Definition (Riesz MV-algebra)

A *Riesz MV-algebra* is a structure $(R, \oplus, ^*, \{r \mid r \in [0,1]\}, 0)$ such that $(R, \oplus, ^*, 0)$ is an MV-algebra and $\{r \mid r \in [0,1]\}$ is a family of unary operation such that the following properties (RMV1)-(RMV4) hold:

(RMV1) $r(x \odot y^*) = (rx) \odot (ry)^*$

(RMV2) $(r \odot q^*) \cdot x = (rx) \odot (qx)^*$

(RMV3) $r(qx) = (rq)x$

(RMV4) $1x = x$.

Note that if we consider $\{r \mid r \in [0,1] \cap \mathbb{Q}\}$ we obtain DMV-algebras (divisible MV-algebras). We denote, in general, $[0,1]_{\mathbb{Q}} := [0,1] \cap \mathbb{Q}$

Definition (Multi-Layer Perceptron in Łukasiewicz Logic)

A MLP with k hidden layers, n inputs and n outputs can be represented as a function $F : [0,1]^n \to [0,1]^n$, such that

$$y_j = \rho\left(\sum_{l=1}^{n} w_{jl}^{k} \rho\left(\dots \rho\left(\sum_{i=1}^{n} w_{pi}^{0} x_i + b^0\right) \dots\right) + b_k\right)$$

where

- $F(x_1, \dots, x_n) = (y_1, \dots, y_n)$;
- $\rho : \mathbb{R} \to [0,1]$ is the activation function $\rho(x) := ReLU_1(x) := \min(1, \max(0, x))$;
- w_{ij}^{k} are the weights in the k^{th} layer.

Hybrid Modal Logic Framework

- we recall: modal logic, hybrid modal logic and many-sorted hybrid modal logic ($\mathcal{H}_\Sigma(@)$);
- then, we specify the multi-layer perceptron and its training process as a particular theory (Λ_{MLP}) of $\mathcal{H}_\Sigma(@)$.

Modal Logic

Language. Propositional variables $p \in \mathrm{Prop}$, Booleans $\neg, \wedge, \vee, \rightarrow$, and one modality $\Box$ (dual $\Diamond\varphi := \neg\Box\neg\varphi$).

Kripke semantics. A frame $F = (W, R)$, model $M = (F, V)$ with $V : \mathrm{Prop} \rightarrow \mathcal{P}(W)$. For $w \in W$:

$$
\begin{aligned}
M, w \models p &\iff w \in V(p) \\
M, w \models \neg\varphi &\iff M, w \not\models \varphi, \\
M, w \models \varphi \rightarrow \psi &\iff (M, w \models \varphi \Rightarrow M, w \models \psi), \\
M, w \models \Box\varphi &\iff \text{for all } v\,(wRv \Rightarrow M, v \models \varphi).
\end{aligned}
$$

Hilbert system (K).

- All propositional tautologies.
- *Modal axiom* (K): $\Box(\varphi \rightarrow \psi) \rightarrow (\Box\varphi \rightarrow \Box\psi)$.
- Rules: *Modus Ponens (MP)* and *Necessitation (Nec)*: from $\vdash \varphi$ infer $\vdash \Box\varphi$.

This system is sound and complete for the class of *all* Kripke frames (W, R); no frame conditions are imposed on R.

Hybrid Modal Logic H(@)

Language. Extend K with a denumerable set of *nominals* $i, j, \dots$ (names of single worlds) and the *satisfaction* operator $@_i\varphi$.

Kripke semantics (with names). A model $M = (W, R, V)$ with $V(i) \in W$ for every nominal i (single designated world). For $w \in W$:

$$M, w \models i \iff w = V(i), \qquad M, w \models @_i\varphi \iff M, V(i) \models \varphi,$$

Booleans and $\Box$ as in K.

Axioms/rules on top of K.

- (K@) $@_i(\varphi \to \psi) \to (@_i\varphi \to @_i\psi)$.
- (Ref@) $@_i\, i$.
- Rules: MP, Nec (for $\Box$), and *Hybrid generalization* (Gen@): from $\vdash \varphi$ infer $\vdash @_i\varphi$ (with i fresh).

Nominals name states; $@_i\varphi$ says "φ holds at the state named i". This enables direct reference to states and local reasoning while retaining K's frame-general completeness.

Many-sorted Hybrid Modal Logic $\mathcal{H}_\Sigma(@)$

Signature. $\Sigma = (S, \Sigma, N)$ where S is a set of sorts; Σ gives poly-ary modal operators $\sigma : s_1 \times \cdots \times s_n \to s$; $N = (N_s)_{s \in S}$ are constant nominals of sort s.

Language (by sort s). $\varphi_s \ ::= \ p \mid j \mid \neg\varphi_s \mid \varphi_s \vee \varphi_s \mid \sigma(\varphi_{s_1}, \ldots, \varphi_{s_n})_s \mid @_k^t \varphi_t$.

Frames and models. A Σ-frame $\mathcal{F} = ((W_s)_{s\in S}, (R_\sigma)_{\sigma\in\Sigma}, (N_s^{\mathcal{F}})_{s\in S})$, with $R_\sigma \subseteq W_s \times W_{s_1} \times \cdots \times W_{s_n}$, $N_s^{\mathcal{F}} = \{w^c \mid c \in N_s\} \subseteq W_s$ (singletons). A model $\mathcal{M} = (\mathcal{F}, V)$ where $V : \mathrm{PROP} \to \mathcal{P}(W)$ is S-sorted.

Satisfaction (only non-Boolean clauses).

- $\mathcal{M}, w \models^s j$, if and only if $V_s(j) = \{w\}$ for any $j \in \mathrm{NOM_s} \cup N_s$,
- if $\sigma \in \Sigma_{s_1 \ldots s_n, S}$ then $\mathcal{M}, w \models^s \sigma(\phi_1, \ldots, \phi_n)$, if and only if there is $(w_1, \ldots, w_n) \in W_{s_1} \times \cdots \times W_{s_n}$ such that $R_\sigma w w_1 \ldots w_n$ and $\mathcal{M}, w_i \models^{s_i} \phi_i$ for any $i \in [n]$,
- $\mathcal{M}, w \models^s @_k^s \psi$ if and only if $\mathcal{M}, u \models^t \psi$ where $k \in \mathrm{NOM}_t \cup N_t$, ψ has the sort t and $V_t^N(k) = \{u\}$.

Many-sorted Hybrid Modal Logic $\mathcal{H}_\Sigma(@)$

- The axioms and the deduction rules of $\mathscr{K}_\Sigma$:
  - For any $s \in S$, if α is a formula of sort s which is a theorem in propositional logic, then α is an axiom.
  - Axiom schemes: for any $\sigma \in \Sigma_{s_1 \cdots s_n, s}$ and for any formulas $\phi_1, \ldots, \phi_n, \phi, \chi$ of appropriate sorts, the following formulas are axioms:

$$\begin{array}{rl}
(K_\sigma) & \sigma^\Box(\ldots, \phi_{i-1}, \phi \to \chi, \phi_{i+1}, \ldots) \to \\
 & \quad (\sigma^\Box(\ldots, \phi_{i-1}, \phi, \phi_{i+1}, \ldots) \to \sigma^\Box(\ldots, \phi_{i-1}, \chi, \phi_{i+1}, \ldots)) \\
(Dual_\sigma) & \sigma(\psi_1, \ldots, \psi_n) \leftrightarrow \neg\sigma^\Box(\neg\psi_1, \ldots, \neg\psi_n)
\end{array}$$

  - Deduction rules: *Modus Ponens* and *Universal Generalization*

$$\begin{array}{rl}
(MP) & \text{if } \vdash^s \phi \text{ and } \vdash^s \phi \to \psi \text{ then } \vdash^s \psi \\
(UG) & \text{if } \vdash^{s_i} \phi \text{ then } \vdash^s \sigma^\Box(\phi_1, .., \phi, .. \phi_n)
\end{array}$$

- Axiom schemes: any formula of the following form is an axiom, where s, s', t are sorts, $\sigma \in \Sigma_{s_1 \cdots s_n, s}$, $\phi, \psi, \phi_1, \ldots, \phi_n$ are formulas (when necessary, their sort is marked as a subscript), j, k are nominals or constant nominals:

$$\begin{array}{rlrl}
(K@) & @_j^s(\phi_t \to \psi_t) \to (@_j^s \phi \to @_j^s \psi) & (Agree) & @_k^t @_j^{t'} \phi_s \leftrightarrow @_j^t \phi_s \\
(SelfDual) & @_j^s \phi_t \leftrightarrow \neg @_j^s \neg \phi_t & (Intro) & j \to (\phi_s \leftrightarrow @_j^s \phi_s) \\
(Back) & \sigma(\ldots, \phi_{i-1}, @_j^{s_i} \psi_t, \phi_{i+1}, \ldots)_s \to @_j^s \psi_t & (Ref) & @_j^s j_t \\
(Nomx) & @_k x \wedge @_j x \to @_k j & &
\end{array}$$

- Deduction rules:

$$\begin{array}{rl}
(BroadcastS) & \text{if } \vdash^s @_j^s \phi_t \text{ then } \vdash^{s'} @_j^{s'} \phi_t \\
(Gen@) & \text{if } \vdash^{s'} \phi \text{ then } \vdash^s @_j \phi\text{, where } j \text{ and } \phi \text{ have the same sort } s' \\
(Name@) & \text{if } \vdash^s @_l \phi \text{ then } \vdash^{s'} \phi\text{, where } l \text{ is a nominal} \\
(Paste) & \text{if } \vdash^s @_j \sigma(\ldots, l, \ldots) \wedge @_l \phi \to \psi \text{ then } \vdash^s @_j \sigma(\ldots, \phi, \ldots) \to \psi \\
 & \text{where } l \text{ is a nominal}
\end{array}$$

Here, j and k are nominals or constant nominals having the appropriate sort.

Figure 1: **The system $\mathscr{H}_\Sigma(@)$** [19]

Multi-layer perceptron theory

- we consider $\mathbf{\Sigma} := (S, \Sigma, N)$ with $S = \{rmv, act, ln\}$ with the following definition of the particular sets of operators and constant nominals:
- $\Sigma_{rmv} = \ \{\neg_L : rmv \to rmv, \oplus_L : rmv \times rmv \to rmv\} \cup \{\Diamond_r : rmv \to rmv \mid r \in [0,1]_{\mathbb{Q}}\}$;
- $N_{rmv} = \{\gamma_r \mid r \in [0,1]_{\mathbb{Q}}\}$ a set of nominal constants
- $\Sigma_{act} = \{init, train, stop : rmv^n \to act \mid n \in \mathbb{N}\}$;
- $\Sigma_{ln} = \{[\_]\langle\_\rangle : act \times rmv^n \to ln \mid n \in \mathbb{N}\}$

Axioms for *rmv*-formulas

- Axioms for nominal constants:

  (Nom1) $\gamma_{\neg_L r} \leftrightarrow \neg_L \gamma_r$ (Nom2) $\gamma_{r \oplus_L q} \leftrightarrow \gamma_r \oplus_L \gamma_q$ (Nom3) $\gamma_{r \cdot q} \leftrightarrow \Diamond_r \gamma_q$

- Axioms for the MV-algebraic operations:

  (M1) $(\varphi \oplus_L (\psi \oplus_L \chi)) \leftrightarrow (\varphi \oplus_L \psi) \oplus_L \chi$
  (M2) $((\neg_L \gamma_0) \oplus_L \varphi) \leftrightarrow (\neg_L \gamma_0)$
  (M3) $((\varphi \odot_L \neg_L \psi) \oplus_L \psi) \leftrightarrow ((\psi \odot_L \neg_L \varphi) \oplus_L \varphi)$
  (M4) $(\neg_l(\neg_L \varphi)) \leftrightarrow (\varphi)$
  (M5) $(\varphi \oplus_L \psi) \leftrightarrow (\psi \oplus_L \varphi)$
  (M6) $\gamma_0 \leftrightarrow (\varphi \oplus_L \gamma_0)$

- Axioms for the scalar multiplication:

  (R1) $(\Diamond_r(\varphi \odot_L \neg_L \psi)) \leftrightarrow ((\Diamond_r \varphi) \odot_L \neg_L (\Diamond_r \psi))$
  (R2) $(\Diamond_{r \odot \neg q} \varphi) \leftrightarrow ((\Diamond_r \varphi) \odot_L \neg_L (\Diamond_q \varphi))$
  (R4) $(\Diamond_1 \varphi) \leftrightarrow \varphi$
  (R3) $(\Diamond_r(\Diamond q \varphi)) \leftrightarrow (\Diamond_{r \cdot q} \varphi)$

where $r, q \in [0,1]_{\mathbb{Q}}$, $r \cdot q$ is the real product on $[0,1]$, $\leftrightarrow$ is the modal equivalence from $\mathscr{H}_\Sigma(@)$ and φ, ψ, χ are arbitrary *rmv*-formulas.

Figure 2: Axioms for *rmv*-formulas

Definitions for neural networks

- n (the number of inputs), k (the number of hidden layers) $\in \mathbb{N}$;
- if $h = (h_1, \dots, h_n) \in [0,1]^n_{\mathbb{Q}}$, then we denote by h^n_1 the vector $(\mathrm{h}_1, \dots, \mathrm{h}_n)$ of corresponding *rmv*-nominal constants;
- *if* $w = (w_{ij})^n_{i,j=1} \in M_n([0,1]_{\mathbb{Q}})$ is a square matrix then we denote by $\mathrm{w} := (\mathrm{w}_{ij})^n_{i,j=1}$ the corresponding matrix of *rmv*-nominal constants

Definitions for neural networks

- n (the number of inputs), k (the number of hidden layers) $\in \mathbb{N}$;
- if $h = (h_1, \ldots, h_n) \in [0,1]^n_{\mathbb{Q}}$, then we denote by h_1^n the vector $(\mathrm{h}_1, \ldots, \mathrm{h}_n)$ of corresponding *rmv*-nominal constants;
- *if* $w = (w_{ij})^n_{i,j=1} \in M_n([0,1]_{\mathbb{Q}})$ is a square matrix then we denote by $\mathrm{w} := (\mathrm{w}_{ij})^n_{i,j=1}$ the corresponding matrix of *rmv*-nominal constants

- The atomic *act*-formulas are:
  - $init(\mathrm{h}_1^n)$ starts the forward training for the n inputs h_1^n;
  - $train(\mathrm{h}_1^n)$ performs a forward step for the n inputs h_1^n;
  - $stop(\mathrm{h}_1^n)$ stops the training process with the outputs h_1^n.

Definitions for neural networks

- the training process of a neural network is an inference on the sort *ln*;
- the particular operator is $[\alpha_{act}]\langle st_{rmv}\rangle$ where
  - α_{act} is an action;
  - st_{rmv} is a sequence of formulas of sort *rmv* representing a configuration;
- the entire formula means that in the state st_{rmv} we perform the action α_{act};
- note that we use $[\alpha_{act}]\langle\rangle$ which means that we reached the empty state.

Neural networks axioms

- before defining the axioms, we consider the following notations where λ_1^n, b_0^k are vectors and w_0^k is a matrix of nominal terms of sort *rmv*:

  (n1) $next_{\mathrm{w},\mathrm{b}}(\lambda_1^n) := (\mathrm{b} \oplus \bigoplus_{i=1}^n \Diamond_{w_{\mathbf{1}i}} \lambda_i, \ldots, \mathrm{b} \oplus \bigoplus_{i=1}^n \Diamond_{w_{ni}} \lambda_i)$

  (n2) $end(\mathrm{y}, \lambda_1^n, \varepsilon) := d_L(\mathrm{y}, \bigvee_1^n \lambda_i) \rightarrow_L \varepsilon$

  (n3) $updated_{\lambda_{\mathbf{1}}^n} \langle \mathrm{w}_0^k, \mathrm{b}_0^k \rangle := \langle \mathrm{uw}_0^k, \mathrm{ub}_0^k \rangle$.

Neural network axioms

For a neural network with one input $(h_1, \ldots, h_n) \in [0,1]^n_{\mathbb{Q}}$ and the expected output $y \in [0,1]_{\mathbb{Q}}$ the axioms are:

(N0) $[\mathit{init}(\mathrm{h}_1^n)]\langle \mathrm{w}_0^k, \mathrm{b}_0^k \rangle \to [\mathit{train}(\mathrm{h}_1^n)]\langle \mathrm{w}_0^k, \mathrm{b}_0^k \rangle$

(N1) $[\mathit{train}(\mathrm{h}_1^n)]\langle \mathrm{w}_i^k, \mathrm{b}_i^k \rangle \to [\mathit{train}(\mathit{next}_{\mathrm{w}_i,\mathrm{b}_i}(\mathrm{h}_1^n))]\langle \mathrm{w}_{i+1}^k, \mathrm{b}_{i+1}^k \rangle$

(N2) $[\mathit{init}(\mathrm{h}_1^n)]\langle \mathrm{w}_0^k, \mathrm{b}_0^k \rangle \to ([\mathit{train}(\lambda_1^n)]\langle\rangle \wedge \neg @_{1_L}^{ln} \mathit{end}(\mathrm{y}, \lambda_1^n, \varepsilon) \to$
$[\mathit{init}(\mathrm{h}_1^n)]\mathit{updated}_{\lambda_1^n} \langle \mathrm{w}_0^k, \mathrm{b}_0^k \rangle)$

(N3) $[\mathit{init}(\mathrm{h}_1^n)]\langle \mathrm{w}_0^k, \mathrm{b}_0^k \rangle \to ([\mathit{train}(\lambda_1^n)]\langle\rangle \wedge @_{1_L}^{ln} \mathit{end}(\mathrm{y}, \lambda_1^n, \varepsilon) \to [\mathit{stop}(\lambda_1^n)]\langle \mathrm{w}_0^k, \mathrm{b}_0^k \rangle)$

Neural network axioms

For a neural network with one input $(h_1, \ldots, h_n) \in [0,1]^n_{\mathbb{Q}}$ and the expected output $y \in [0,1]_{\mathbb{Q}}$ the axioms are:

(N0) $[init(\mathrm{h}_1^n)]\langle \mathrm{w}_0^k, \mathrm{b}_0^k\rangle \to [train(\mathrm{h}_1^n)]\langle \mathrm{w}_0^k, \mathrm{b}_0^k\rangle$

(N1) $[train(\mathrm{h}_1^n)]\langle \mathrm{w}_i^k, \mathrm{b}_i^k\rangle \to [train(next_{\mathrm{w}_i,\mathrm{b}_i}(\mathrm{h}_1^n))]\langle \mathrm{w}_{i+1}^k, \mathrm{b}_{i+1}^k\rangle$

(N2) $[init(\mathrm{h}_1^n)]\langle \mathrm{w}_0^k, \mathrm{b}_0^k\rangle \to ([train(\lambda_1^n)]\langle\rangle \wedge \neg @_{1_L}^{ln} end(\mathrm{y}, \lambda_1^n, \varepsilon) \to [init(\mathrm{h}_1^n)]updated_{\lambda_1^n}\langle \mathrm{w}_0^k, \mathrm{b}_0^k\rangle)$

(N3) $[init(\mathrm{h}_1^n)]\langle \mathrm{w}_0^k, \mathrm{b}_0^k\rangle \to ([train(\lambda_1^n)]\langle\rangle \wedge @_{1_L}^{ln} end(\mathrm{y}, \lambda_1^n, \varepsilon) \to [stop(\lambda_1^n)]\langle \mathrm{w}_0^k, \mathrm{b}_0^k\rangle)$

Our logic is $\mathcal{H}_{\mathbf{\Sigma}}(@) + \Lambda_{MLP}$, where

$$\Lambda_{MLP} = \{(Nom1) - (Nom3), (M1) - (M6), (R1) - (R4), (N(0) - (N3)\}$$

The (weak) completeness results hold: our logic is complete with respect to the class of models defined by Λ_{MLP}.

Example

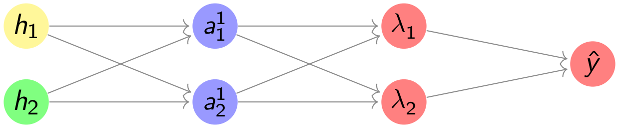

Figure: Example

- we have: $n = 2$, $k = 1$;
- we consider: the inputs $\mathrm{h} = (0.2, 0.3)$, the expected output $\mathrm{y} = 0.8$, the admitted error $\varepsilon = 10^{-1}$, the learning rate $\eta = 0.1$ and the initial weights and biases:

$$\mathrm{w}_0 = \begin{pmatrix} 0.4 & 0.3 \\ 0.6 & 0.1 \end{pmatrix}, \mathrm{w}_1 = \begin{pmatrix} 0.9 & 0.8 \\ 0 & 1 \end{pmatrix}, \mathrm{b}_0 = 0.1, \mathrm{b}_1 = 0.15.$$

Example

The training process performs as follows:

$$
\begin{array}{lll}
(1) & [\mathit{init}(\mathrm{h})]\langle(\mathrm{w}_0,\mathrm{w}_1),(\mathrm{b}_0,\mathrm{b}_1)\rangle \to [\mathit{train}(\mathrm{h})]\langle(\mathrm{w}_0,\mathrm{w}_1),(\mathrm{b}_0,\mathrm{b}_1)\rangle & (\mathrm{N0}) \\
(2) & [\mathit{train}(\mathrm{h})]\langle(\mathrm{w}_0,\mathrm{w}_1),(\mathrm{b}_0,\mathrm{b}_1)\rangle \to [\mathit{train}(\mathit{next}_{\mathrm{w}_0,\mathrm{b}_0}(\mathrm{h}))]\langle\mathrm{w}_1,\mathrm{b}_1\rangle & (\mathrm{N1})
\end{array}
$$

If $\mathrm{a}^1 = (a^1_1, a^1_2) = \mathit{next}_{\mathrm{w}_0,\mathrm{b}_0}(\mathrm{h})$, then $\mathrm{a}^1 = (0.27, 0.25)$.

$$
(3) \quad [\mathit{train}(\mathrm{a})]\langle\mathrm{w}_1,\mathrm{b}_1\rangle \to [\mathit{train}(\mathit{next}_{\mathrm{w}_1,\mathrm{b}_1}(\mathrm{a}))]\langle\rangle \quad (\mathrm{N1})
$$

We note that $\lambda = (\lambda_1, \lambda_2) = \mathit{next}_{\mathrm{w}_1,\mathrm{b}_1}(\mathrm{a}) = (0.393, 0.626)$.

$$
(4) \quad [\mathit{init}(\mathrm{h})]\langle(\mathrm{w}_0,\mathrm{w}_1),(\mathrm{b}_0,\mathrm{b}_1)\rangle \to [\mathit{train}(\lambda)]\langle\rangle \quad (1,2,3)
$$

We note that $\hat{y} = 0.626$, so $\mathit{end}(\mathrm{y}, \lambda, \varepsilon) = d_L(\mathrm{y}, \hat{y}) \to_L \varepsilon$ is equivalent with $0.174 \to_{\mathrm{L}} 0.1$, which means that $@^{ln}_{1_L}\mathit{end}(\mathrm{y}, \lambda, \varepsilon)$ is *false*. Consequently, we apply (N2):

$$
\begin{array}{lll}
(5) & [\mathit{init}(\mathrm{h})]\langle(\mathrm{w}_0,\mathrm{w}_1),(\mathrm{b}_0,\mathrm{b}_1)\rangle \to ([\mathit{train}(\lambda)]\langle\rangle \wedge \neg @^{ln}_{1_L}\mathit{end}(\mathrm{y},\lambda,\varepsilon) \to & \\
 & \quad \to [\mathit{init}(\mathrm{h})]\mathit{updated}_\lambda\langle(\mathrm{w}_0,\mathrm{w}_1),(\mathrm{b}_0,\mathrm{b}_1)\rangle) & (\mathrm{N2})
\end{array}
$$

Verifying network properties

- the system $\mathcal{H}_\Sigma(@) + \Lambda_{MLP}$ can be adapted for verifying network properties;
- we show that we can track the number of eochs of the training process;
- we consider E our limit, and if $1_E = 1/E$, then $1_E \oplus \cdots \oplus 1_E = 1$ if the sum has E terms;
- we keep this formula as the first argument of the configuration operator $\langle \_ \rangle : rmv^n \to ln$.

Verifying network properties - axioms

$$(\mathrm{N0}_{\neg E})\quad [\mathit{init}(\mathrm{h}_1^n)]\langle \mathrm{r}, \mathrm{w}_0^k, \mathrm{b}_0^k\rangle \wedge \neg @_{1_L}^{ln}\mathrm{r} \rightarrow [\mathit{train}(\mathrm{h}_1^n)]\langle \mathrm{r}, \mathrm{w}_0^k, \mathrm{b}_0^k\rangle$$

$$(\mathrm{N0}_{E})\quad [\mathit{init}(\mathrm{h}_1^n)]\langle \mathrm{r}, \mathrm{w}_0^k, \mathrm{b}_0^k\rangle \wedge @_{1_L}^{ln}\mathrm{r} \rightarrow [\mathit{stop}()]\langle 1_L, \mathrm{w}_0^k, \mathrm{b}_0^k\rangle$$

$$(\mathrm{N1})\quad [\mathit{train}(\mathrm{h}_1^n)]\langle \mathrm{r}, \mathrm{w}_i^k, \mathrm{b}_i^k\rangle \rightarrow [\mathit{train}(\mathit{next}_{\mathrm{w}_i,\mathrm{b}_i}(\mathrm{h}_1^n))]\langle \mathrm{r}, \mathrm{w}_{i+1}^k, \mathrm{b}_{i+1}^k\rangle$$

$$(\mathrm{N2})\quad [\mathit{init}(\mathrm{h}_1^n)]\langle \mathrm{r}, \mathrm{w}_0^k, \mathrm{b}_0^k\rangle \rightarrow ([\mathit{train}(\lambda_1^n)]\langle\rangle \wedge \neg @_{1_L}^{ln}\mathit{end}(\mathrm{y}, \lambda_1^n, \varepsilon) \rightarrow [\mathit{init}(\mathrm{h}_1^n)]\mathit{updated}_{\lambda_1^n}\langle \mathrm{r} \oplus 1_E, \mathrm{w}_0^k, \mathrm{b}_0^k\rangle)$$

$$(\mathrm{N3})\quad [\mathit{init}(\mathrm{h}_1^n)]\langle \mathrm{r}, \mathrm{w}_0^k, \mathrm{b}_0^k\rangle \rightarrow ([\mathit{train}(\lambda_1^n)]\langle \mathrm{r}\rangle \wedge @_{1_L}^{ln}\mathit{end}(\mathrm{y}, \lambda_1^n, \varepsilon) \rightarrow [\mathit{stop}(\lambda_1^n)]\langle \mathrm{r}, \mathrm{w}_0^k, \mathrm{b}_0^k\rangle)$$

Backpropagation in Łukasiewicz logic

- Backpropagation is formulated entirely within Łukasiewicz logic: every stage is computed in $[0,1]$ and uses only MV–algebraic operations.
- For each layer $t \in \{1,\ldots,k\}$, the forward pass is $a_t := \mathrm{ReLU}_1(z_t)$, where $z_t = W_t a_{t-1} + b_t$
- The derivative of the activation is represented as a diagonal matrix

$$D^t := \mathrm{diag}\big(1_{(0,1)}(z_t^1), \ldots, 1_{(0,1)}(z_t^{n_t})\big),$$

  where n_t is the number of neurons of layer t and $1_{(0,1)}(z) = 1$ if $0 < z < 1$ and 0 otherwise.
- At the output layer the initial gradient is $g := \mathrm{sign}(a_k - y) \in \{-1,0,1\}^{n_k}$.

Chain rule and parameter gradients

- The loss is measured with the Łukasiewicz distance d_L and backpropagation proceeds by the chain rule.
- For any hidden layer t,

$$\nabla_{z_t} d_L \;=\; \Pi_t g, \qquad \Pi_t := D_t (W_{t+1})^\top D_{t+1} (W_{t+2})^\top \cdots D_k .$$

- Parameter gradients:

$$G_{W_t} \;=\; \nabla_{W_t} d_L \;=\; (\nabla_{z_t} d_L)\,(a_{t-1})^\top, \qquad G_{b_t} \;=\; \nabla_{b_t} d_L \;=\; \nabla_{z_t} d_L .$$

Normalization and Łukasiewicz updates

- Since raw gradients may lie outside $[0,1]$, normalize by the ℓ_∞–norm:

$$\hat{g} \;=\; \frac{|g|}{\|G\|_\infty + \varepsilon} \;\in [0,1], \qquad \varepsilon > 0.$$

- With learning rate $\eta \in [0,1]$, combine via the Łukasiewicz product:

$$\Delta \;=\; \eta \otimes \hat{g}.$$

- Update is expressed exclusively with Łukasiewicz operations. For each weight:

$$\mathsf{uw} \;=\; (\mathsf{w} \ominus \Delta^-) \;\oplus\; \Delta^+$$

$$\Delta^+ = \begin{cases} \eta \otimes \hat{g}, & g < 0 \\ 0, & \text{otherwise} \end{cases}, \qquad \Delta^- = \begin{cases} \eta \otimes \hat{g}, & g > 0 \\ 0, & \text{otherwise} \end{cases}$$

Implementation in Lean

- implementation of the many-sorted hybrid modal logic + the multi-layer perceptron theory;
- algorithm that generates a model;
- real-world experiments.

Many-sorted hybrid modal logic in Lean 4

```
N0 {Γ : Ctx σ} {φ ψ : FormNN σ} : ProofNN Γ $ [[ActionNN.init]] φ ⊃ [[ActionNN.train]] ψ
N1 {Γ} {n m : Nat} {W : Matrix Float n m} {b : Float} {φ : List (FormNN σ)} :
  ProofNN Γ $ [[ActionNN.train]] (FormNN.list φ) ⊃ FormNN.list (layer_activation_form b W φ)
N2 {Γ : Ctx σ} {n k : Nat}
   {W : Vector (Matrix Float n n) k} {b : Vector Float k}
   {input : Vector Float n} {L : List (FormNN σ)}
   {target : FormNN σ} {ε : FormNN σ} :
  let ψ := FormNN.list (encode_pair W b)
  let trainPart := [[ActionNN.train]] (FormNN.list L)
  let condition := ¬L (FormNN.hybrid (#n 1) (sort.atom 0) (target L ε))
  ProofNN Γ $ ([[ActionNN.init]] ψ ⊃ (trainPart & condition)) ⊃ [[ActionNN.update]] ψ
N3 {Γ : Ctx σ} {n k : Nat}
   {W : Vector (Matrix Float n n) k} {b : Vector Float k}
   {input : Vector Float n} {L : List (FormNN σ)}
   {target : FormNN σ} {ε : FormNN σ} :
  let ψ := FormNN.list (encode_pair W b)
  let trainPart := [[ActionNN.train]] (FormNN.list L)
  let condition := FormNN.hybrid (#n 1) (sort.atom 0) (target L ε)
  ProofNN Γ $ ([[ActionNN.init]] ψ ⊃ (trainPart & condition)) ⊃ [[ActionNN.Stop]] ψ
```

Verifying the number of epochs

```
theorem inductive_step_termination
  {n m k : Nat} {y η ε E : Float} {Γ : Ctx σ}
  {W : Vector (Matrix Float n m) k} {b : Vector Float k}
  {lφ : List $ FormNN σ} {ln : sort σ}
  {mem ν : FormNN σ}
  [Inhabited $ FormNN σ] [Inhabited $ Nominal σ] [OfNat (Fin σ) 0] :
  Γ ⊢ ([[ActionNN.train]] ≪mem, ν≫ ⊃ FormNN.list lφ) →
  Γ ⊢ [[ActionNN.update]] ≪mem ⊕ nomToForm (#γ (1/E)), ν≫ →
  Γ ⊢ ~@@(#n 1), ln : ((dL (nomToForm (#γ y)) (foldr (fun φ ψ => φ ∨ ψ) zL lφ)) →L nomToForm (#γ ε)) →
  Γ ⊢ (@@(#n 1), ln : dL mem (nomToForm (#γ ((E − 1)/E)))) &
    [[ActionNN.train]] ≪mem ⊕ nomToForm (#γ (1/E)), ν≫
```

Automatically Generated Model Algorithm

1. Start from the initial state s_0, with initial weights and biases
2. apply `Action.train` to compute a new state via forward propagation
3. Evaluate the output of the network.
4. Compute the loss with respect to the given target vector.
5. **If** the loss is below the given threshold: (5.1) apply `Action.stop` to finalize the training and (5.2) terminate the algorithm and return the list of all transitions and the final state, with the computed weights and biases.
6. **Else**: (6.1) apply `Action.update` to adjust the biases and (6.2) repeat from step 2 for the next epoch, up to the maximum allowed number of epochs.

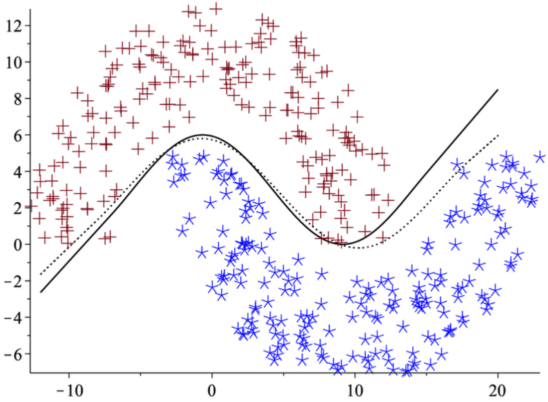

Figure: Two moons dataset for classification

Experiment - training & results

- 6000 training examples & 2000 test examples;
- these classes are balanced;
- each input vector is scaled to the unit interval $[0, 1]$;
- we use a fully-connected architecture with two hidden layers, of 32 neurons each, followed by a single output unit;
- $\eta = 1$;
- we use in the training process mini-batches of size 128 for 250 epochs;
- we compare with a similar Python architecture, but with ReLU in the hidden layers, a *sigmoid* output, binary cross-entropy and SGD as the optimization part.

Model	Train Accuracy	Test Accuracy
Lean Łukasiewicz MLP	0.9	0.89
Python Classic MLP	0.96	0.96

Table: Comparative results

Related Work

- the idea of representing neural networks as formulas of an extension of Łukasiewicz logic goes back to earlier work; recent *Logical Neural Networks* further systematize t-norm–based approaches;
- our setting builds on the general many-sorted hybrid modal logic from prior work where it was used to specify a (toy) programming language and its operational semantics;
- formal verification has emerged as a tool for certifying NN behaviour; the Hoare-like framework *NeSAL* is highlighted. In related results, the system $H_{\Sigma}(@, \forall)$ can model a programming language and an adequate Hoare logic, suggesting future alignment with NeSAL within our logic;
- Lean 4 is chosen for its dual nature as an extensible theorem prover and an efficient programming language.

Conclusions

- we propose **many-sorted hybrid modal logic** as a general, expressive system in which a multilayer perceptron (with ReLU_1) is specified as a *particular theory*; training actions become modal operators and the training *process* is a sequence of logical deductions;
- using Lean 4, the algorithmic implementation of training is backed by logical proofs, integrating specification, verification, and execution;
- on *two-moons* experiment, the Łukasiewicz MLP achieves $\approx$0.90 train / 0.89 test accuracy (compared to 0.96 / 0.96), indicating stable learning under strict Łukasiewicz arithmetic and pointing to refinements (e.g., smoother/fuzzy losses).
- this work contributes to defining and analyzing neural networks within a logical framework, supporting more transparent and reliable AI.

The End