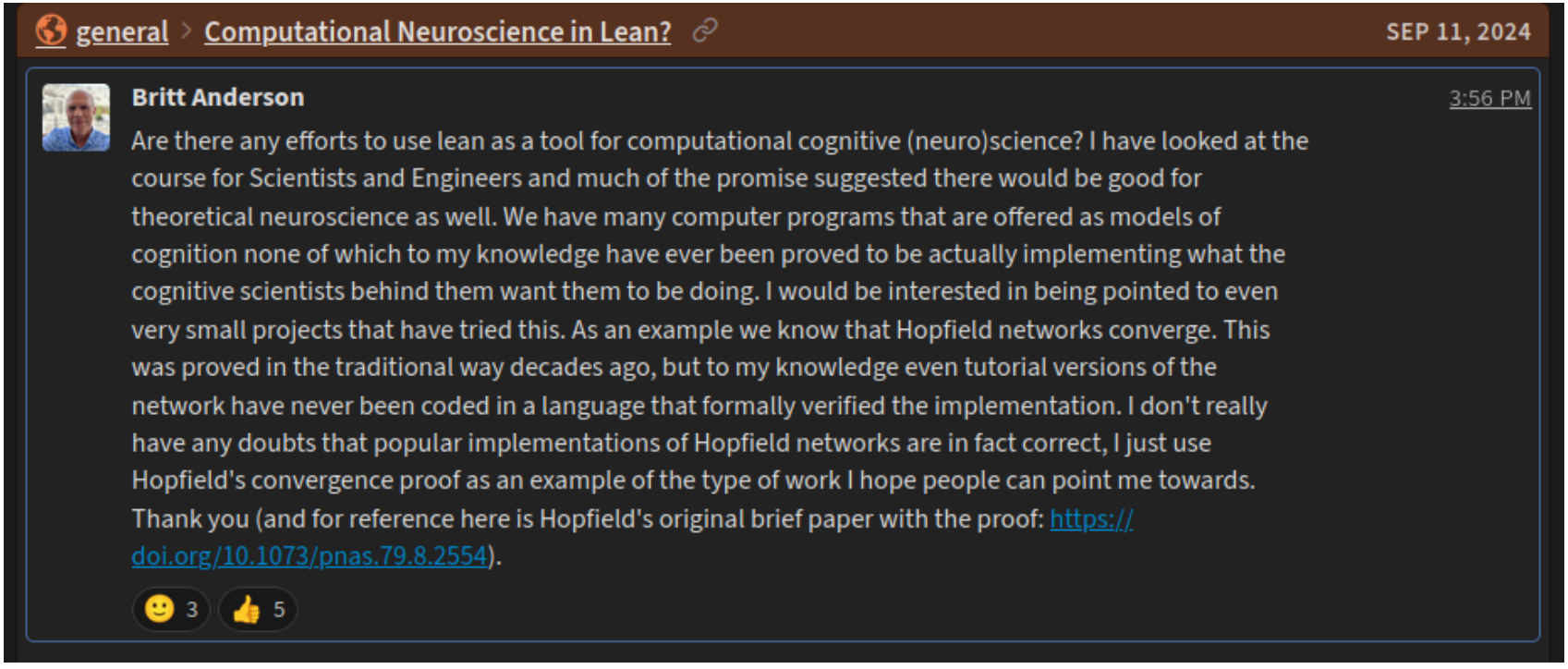
Hopfield Networks and Boltzmann Machines

Leaning In! March 12, 2026, Berlin, Germany

Michail Karatarakis,
Radboud University Nijmegen, The Netherlands
(joint work with Matteo Cipollina and Freek Wiedijk)

March 12, 2026

Project idea



general › Computational Neuroscience in Lean? SEP 11, 2024

Britt Anderson 3:56 PM

Are there any efforts to use lean as a tool for computational cognitive (neuro)science? I have looked at the course for Scientists and Engineers and much of the promise suggested there would be good for theoretical neuroscience as well. We have many computer programs that are offered as models of cognition none of which to my knowledge have ever been proved to be actually implementing what the cognitive scientists behind them want them to be doing. I would be interested in being pointed to even very small projects that have tried this. As an example we know that Hopfield networks converge. This was proved in the traditional way decades ago, but to my knowledge even tutorial versions of the network have never been coded in a language that formally verified the implementation. I don't really have any doubts that popular implementations of Hopfield networks are in fact correct, I just use Hopfield's convergence proof as an example of the type of work I hope people can point me towards. Thank you (and for reference here is Hopfield's original brief paper with the proof: https://doi.org/10.1073/pnas.79.8.2554).

3 5

A coincidence

- In October 2024, John Hopfield and Geoffrey Hinton were awarded the Nobel Prize in Physics for their foundational work in artificial neural networks.
- **Hopfield Networks (1982):** Introduced recurrent connections to model associative memory. They store and recall patterns from partial or noisy inputs by converging to stable states.
- **Boltzmann Machines (1985):** Introduced by Ackley, Hinton, and Sejnowski as a stochastic generalization of Hopfield networks.
- In Boltzmann machines, neuron updates are probabilistic, allowing the network to sample from probability distributions over binary states rather than converging to a single stable state.

Why do we care about Hopfield networks?

- **Associative Memory:** Unlike standard computer memory (which uses exact addresses), Hopfield networks address memory by content. They can recall a complete, stored pattern even when presented with partial, noisy, or corrupted inputs.
- **Error Correction & Pattern Recognition:** A key feature is their convergence property, where the network reaches a stable state corresponding to a stored pattern. This ability to converge to the "closest" stable state makes them highly effective for error correction and recognizing patterns in distorted data.
- **Cross-Disciplinary Bridge:** They are valuable across fields such as physics, neuroscience, and machine learning, due to their connections to statistical mechanics, recurrent neural networks, and cognitive psychology.
- **Unsupervised Learning:** Boltzmann machines (the stochastic generalization) are powerful tools for probabilistic modeling and unsupervised learning, capable of learning the underlying probability distributions of complex datasets.

Related Work

- **Verification vs. Learning:** Extensive work exists on using automated theorem provers to verify neural network *outputs* [4, 7, 9, 10, 15, 11, 12], but formalization of *learning algorithms* themselves is relatively unexplored.
- **Convergence Formalizations:** Murphy et al. [14] formalized the one-layer perceptron in Rocq.
- **Sequential vs. General Architectures:** Formalizations typically rely on sequential layer paradigms or computational graphs, such as in Isabelle/HOL [6, 5] or Rocq [1]. These are limited in natively capturing arbitrary topologies, especially recurrent cycles.

Beginning of the project

- Initially we only formalized Hopfield networks independently and then joined forces.
- One version used rational-valued matrices and a graph-based neural network framework, allowing direct computation (e.g. evaluating the Hebbian algorithm).
- The other version, followed a physics-inspired approach and extended Hopfield networks to asymmetric or stochastic variants, but employed real numbers, making certain parts non-computable.

Why formalize neural network theory

Neural networks are widely used, their formal verification remains challenging.

- Verification must cover not just the outputs of these networks, but the mathematical foundations underlying the models themselves.
- Choice between formalizing theory, algorithms or implementations.
- Neural networks have combinatorial, probabilistic, and thermodynamic aspects and require a verified bridge between these fields.

Our contribution

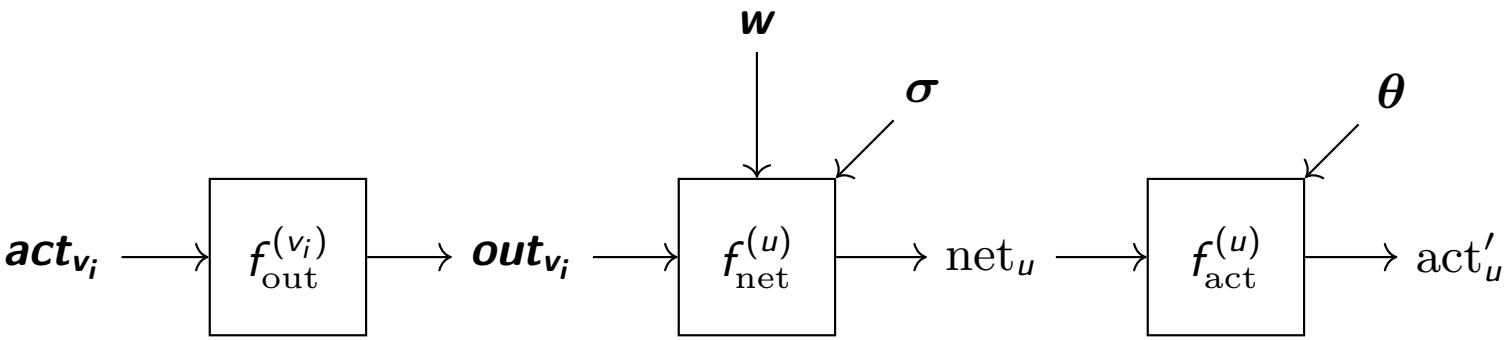
- We provide a comprehensive Lean 4 formalization of neural networks, covering both deterministic and stochastic models.
- **Key Contributions:**
  - A unified framework encompassing feedforward and recurrent architectures.
  - Formalization of Hopfield networks, proofs of their convergence, and implementation of the Hebbian learning algorithm.
  - Formalization of probabilistic concepts like reversibility, Markov kernels, and Gibbs sampling.
  - The first formalization of the Perron-Frobenius theorem for irreducible and stochastic matrices in Lean 4.
  - Formalization of Boltzmann machines and proofs of their convergence (ergodicity) to a unique stationary distribution.
- Over 15,300 lines of code, bridging machine learning, statistical mechanics, and mathematics.

General Neural Networks and Sequential Architectures

Feature	General Neural Network	Sequential Architecture
Graph Structure	Directed graph $G = (U, C)$	Layered structure $\mathcal{L} = (L_0, \ldots, L_k)$ $(k \geq 1)$
Neuron Partition	$U_{\text{in}} \cup U_{\text{out}} \cup U_{\text{hidden}}$	$U_{\text{in}} = L_0$, $U_{\text{out}} = L_k$, $U_{\text{hidden}} = \bigcup_{i=1}^{k-1} L_i$
Connections (C)	Arbitrary directed edges $c \in C$	Restricted to adjacent layers: $u \in L_i \wedge v \in L_{i-1}$
Parameter Count	Variable $\kappa_1(u)$ and $\kappa_2(u)$	Fixed $\kappa_1(u) = 1$ (bias β_u), variable $\kappa_2(u)$
Activations act_u	$\mathbb{R}$	Binary states (e.g., 1 or -1)
Network input function $f_{\text{net}}^{(u)}$	$\mathbb{R}^{2\lvert\text{pred}(u)\rvert+\kappa_1(u)} \to \mathbb{R}$	$\sum_{v \in \text{pred}(u)} w_{uv}\text{out}_v + \beta_u$
Activation function $f_{\text{act}}^{(u)}$	$\mathbb{R}^{1+\kappa_2(u)} \to \mathbb{R}$	e.g. ReLU
Output function $f_{\text{out}}^{(u)}$	$\mathbb{R} \to \mathbb{R}$	$f_{\text{out}}^{(u)} = \text{act}_u$

Parameter vectors $\boldsymbol{\sigma}^{(u)} = (\sigma_1^{(u)}, \ldots, \sigma_{\kappa_1(u)}^{(u)})$ and $\boldsymbol{\theta}^{(u)} = (\theta_1^{(u)}, \ldots, \theta_{\kappa_2(u)}^{(u)})$ are the input parameter vectors and each connection $(v, u) \in C$ has a weight w_{uv}.

Structure of a generalized neuron (abstractly)

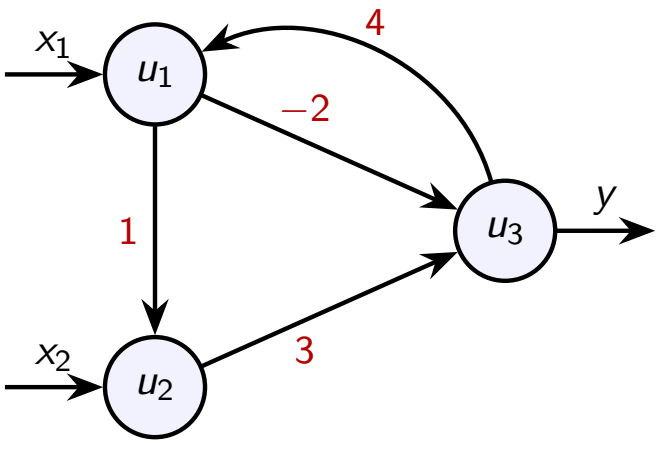


Here, $\mathbf{act}_{v_i} = (\mathrm{act}_{v_1}, \ldots, \mathrm{act}_{v_{|U|}})$ and $\mathbf{out}_{v_i} = (\mathrm{out}_{v_1}, \ldots, \mathrm{out}_{v_{|U|}})$.

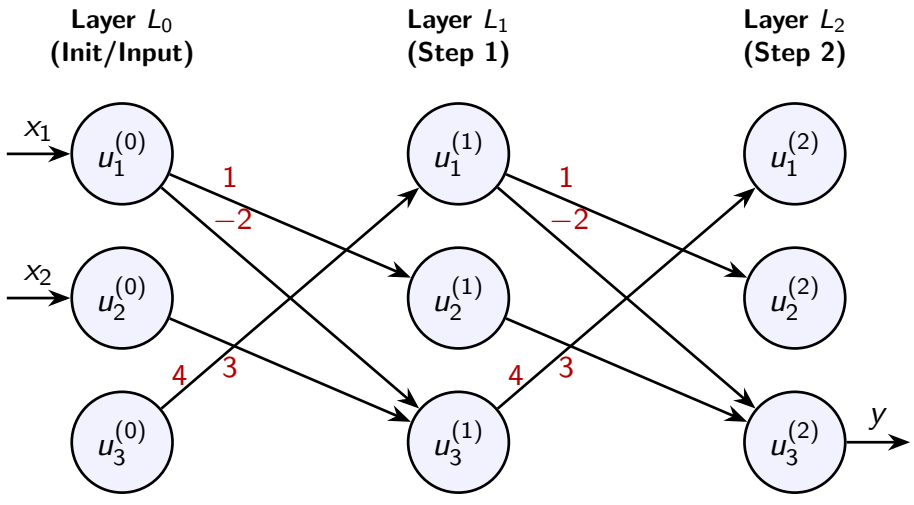
- Neurons may update simultaneously (synchronous update) using previous outputs, or one at a time (asynchronous update), incorporating recent outputs.
- The network's state $s = (\mathrm{act}_{u_1}, \ldots, \mathrm{act}_{u_n})^\top \in \{-1, 1\}^n$, consists of the whole network's neuron activations.

Examples of neural networks

General neural network



Sequential architecture



Formalization

The solution is to divide a neural network into its architecture, defined by its graph structure `NeuralNetwork`, and parameters `Params`, controlling dynamics like input processing. This separation allows flexibility, enabling different functions, weights, or learning rules without redefining the architecture.

Architectural Representation

The Standard Approach (e.g., PyTorch, TensorFlow)

- Most AI frameworks and existing formal verifications (like Certigrad or Rocq models) represent networks sequentially: as lists of layers or directed acyclic graphs (DAGs).
- **The limitation:** This "feedforward" paradigm fundamentally struggles to natively capture recurrent networks (where information loops), like Hopfield and Boltzmann machines.

The Pivot: From `Digraph` to `Quiver`

A Graph-Based Foundation

- To prove global dynamic properties (like energy convergence and ergodicity), we needed to model arbitrary connectivity.
- We started by building our `NeuralNetwork` as a `structure` by extending `mathlib`'s `Digraph` `structure`, which treats adjacency simply as a proposition i.e `Adj : V → V → Prop`
- `Digraph` only tells us *if* an edge exists.
- To prove the Perron-Frobenius theorem (necessary for Boltzmann ergodicity), we needed to reason combinatorially about matrix powers (A^k). This requires counting and manipulating distinct *paths* of length k.

The `Quiver` solution

- `Hom : V → V → Type`.
- We pivoted our entire architecture to `mathlib`'s `Quiver`.
- Unlike `Digraph`, `Quiver` models connections as a *Type*.
- This assigns a distinct identity to every edge, natively equipping the network topology with an algebra of paths required for proofs.

Sequential Architectures as a special case

By adopting the highly general `Quiver` structure, we created a framework that generalizes standard architectures.

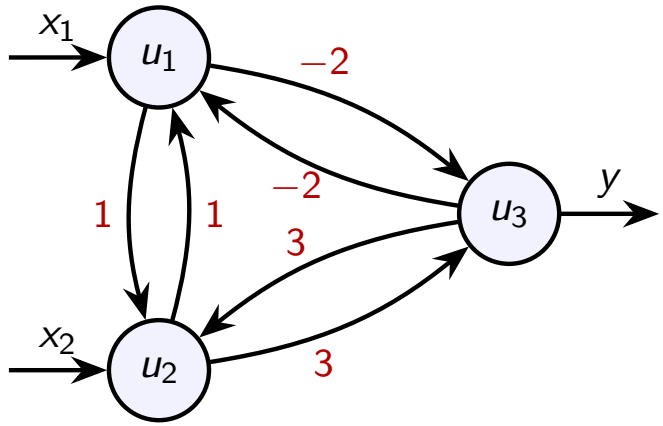
Embedding Feedforward Networks

- We proved that standard "list-of-layers" sequential models are simply a strict sub-category of our framework.
- We achieved this by defining a topological embedding: a `Quiver` where the set of morphisms (edges) is only inhabited if the source neuron lies exactly one layer behind the target neuron.
- **One Result:**The thermodynamic theorems also apply directly to standard feedforward architectures.

Hopfield Network

Definition

Feature	Hopfield Network
Neuron Partition	$U_{\text{hidden}} = \emptyset$, and $U_{\text{in}} = U_{\text{out}} = U$
Connections	No self-connections
Weights	Symmetric ($w_{uv} = w_{vu}$ for $u \neq v$), $w_{uu} = 0$
act_u	Binary states (e.g., 1 or -1)
$f_{\text{net}}^{(u)}$	$\sum_{v \in U \setminus \{u\}} w_{uv} \text{out}_v$
$f_{\text{act}}^{(u)}$	1 if $\text{net}_u \geq \theta_u$, else -1
$f_{\text{out}}^{(u)}$	$f_{\text{out}}^{(u)}(\text{act}_u) = \text{act}_u$



One of the main theorems

Theorem (Convergence Theorem for Hopfield networks)

If the activations of the neurons of a Hopfield network are updated asynchronously, a stable state is reached in a finite number of steps. If the neurons are traversed in an arbitrary, but fixed cyclic fashion, at most $n \cdot 2^n$ steps (updates of individual neurons) are needed, where n is the number of neurons of the network.

The energy function of a Hopfield network with n neurons $u_1, \ldots, u_n$ is

$$E = -\frac{1}{2} \sum_{\substack{u,v \in U \\ u \neq v}} w_{uv}\, act_u\, act_v + \sum_{u \in U} \theta_u\, act_u.$$

Boltzmann machines

- In stochastic networks, "stabilization" no longer means convergence to a single state but convergence to a stationary distribution over states.
- Boltzmann machines are a canonical example of stochastic networks.
- Gibbs algorithm 'resamples' only one coordinate of the system while keeping the others fixed with some probability.
  The energy function is identical to that of Hopfield networks and ensures that Gibbs updates satisfy detailed balance with respect to the Boltzmann distribution.

$$\pi(\mathbf{act}) = \frac{1}{c} \exp\left(-\frac{E(\mathbf{act})}{kT}\right) \tag{1}$$

where $c = \sum_{\mathbf{act}} \exp(-E(\mathbf{act})/T)$ ensures the probabilities of all possible states equal one, a real number $T > 0$ denoting the temperature of the system, and k is Boltzmann's constant[1].

Unification with PhysLean

- **The State Polymorphism Design:** We parameterized our networks over abstract activation types (e.g., bipolar $\{-1, 1\}$ for Hopfield vs. binary $\{0, 1\}$ for Boltzmann).
- **Bridging Disciplines:** The network's energy can be interpreted as physical **Hamiltonian** - the total energy of a system in Physics.
- **Integration:** We embedded our neural networks into PhysLean's Statistical Mechanics codebase.

Currently exploring

- The "Transformer" Bridge": Investigating for a link between the energy function of Continuous Modern Hopfield Networks and the attention mechanism. The goal is to formally link stat-mech to the architecture of LLMs.

Future Work

- **Probability**
  - Markov Chain Monte Carlo (MCMC) methods: a way to sample from a distribution by constructing a Markov chain whose long-run behavior reflects it. Extend the formalization of Markov chain theory to develop a comprehensive MCMC library.
  - Complete the Perron-Frobenius theorem for general non-negative matrices to serve as a foundation for verifying advanced sampling algorithms like Metropolis-Hastings.
- **Computation**
  - Use the current formalization as a target specification to verify a realistic, hardware-oriented implementation via refinement.
  - Transition the development to be fully computable by integrating a constructive library for real numbers (Coquelicot [2], Flocq [3], or CoRN [8] in Rocq, or Meiburg's computable reals library [13]) (in Lean) or our current WIP version.
- Connect with more Lean libraries (e.g. Optlib, CSLib).

References I

[1] Alexander Bagnall and Gordon Stewart. "Certifying the True Error: Machine Learning in Coq with Verified Generalization Guarantees". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (July 2019), pp. 2662–2669. DOI: 10.1609/aaai.v33i01.33012662. URL: https://ojs.aaai.org/index.php/AAAI/article/view/4115.

[2] Sylvie Boldo, Catherine Lelay, and Guillaume Melquiond. "Coquelicot: A User-Friendly Library of Real Analysis for Coq". In: *Mathematics in Computer Science* 9.1 (Mar. 2015), pp. 41–62. DOI: 10.1007/s11786-014-0181-1. URL: https://inria.hal.science/hal-00860648.

[3] Sylvie Boldo and Guillaume Melquiond. "Flocq: A Unified Library for Proving Floating-Point Algorithms in Coq". In: *2011 IEEE 20th Symposium on Computer Arithmetic*. 2011, pp. 243–252. DOI: 10.1109/ARITH.2011.40.

References II

[4] Elena Botoeva et al. "Efficient Verification of ReLU-Based Neural Networks via Dependency Analysis". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.04 (Apr. 2020), pp. 3291–3299. DOI: 10.1609/aaai.v34i04.5729. URL: https://ojs.aaai.org/index.php/AAAI/article/view/5729.

[5] Achim D Brucker and Amy Stell. "Formalizing Neural Networks". In: (Nov. 2025). URL: https://ore.exeter.ac.uk/articles/journal_contribution/Formalizing_Neural_Networks/30638984.

[6] Achim D. Brucker and Amy Stell. "Verifying Feedforward Neural Networks for Classification in Isabelle/HOL". In: *Formal Methods: 25th International Symposium, FM 2023, Lübeck, Germany, March 6–10, 2023, Proceedings*. Lübeck, Germany: Springer-Verlag, 2023, pp. 427–444. ISBN: 978-3-031-27480-0. DOI: 10.1007/978-3-031-27481-7_24. URL: https://doi.org/10.1007/978-3-031-27481-7_24.

References III

[7] Rudy R Bunel et al. "A Unified View of Piecewise Linear Neural Network Verification". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/be53d253d6bc3258a8160556dda3e9b2-Paper.pdf.

[8] Luís Cruz-Filipe, Herman Geuvers, and Freek Wiedijk. "C-CoRN, the Constructive Coq Repository at Nijmegen". In: *Mathematical Knowledge Management*. Ed. by Andrea Asperti, Grzegorz Bancerek, and Andrzej Trybulec. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 88–103. ISBN: 978-3-540-27818-4.

References IV

[9] Remi Desmartin et al. "A Certified Proof Checker for Deep Neural Network Verification in Imandra". In: *16th International Conference on Interactive Theorem Proving (ITP 2025)*. Ed. by Yannick Forster and Chantal Keller. Vol. 352. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025, 1:1–1:21. ISBN: 978-3-95977-396-6. DOI: 10.4230/LIPIcs.ITP.2025.1. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITP.2025.1.

[10] Rüdiger Ehlers. "Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks". In: *Automated Technology for Verification and Analysis*. Ed. by Deepak D'Souza and K. Narayan Kumar. Cham: Springer International Publishing, 2017, pp. 269–286. ISBN: 978-3-319-68167-2.

[11] Wang Lin et al. "Robustness Verification of Classification Deep Neural Networks via Linear Programming". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 11410–11419. DOI: 10.1109/CVPR.2019.01168.

References V

[12] Changliu Liu et al. "Algorithms for Verifying Deep Neural Networks". In: *Foundations and Trends in Optimization* 4.3-4 (2021), pp. 244–404. ISSN: 2167-3888. DOI: 10.1561/2400000035. URL: http://dx.doi.org/10.1561/2400000035.

[13] Alex Meiburg. *ComputableReal: Computable implementation of real numbers in Lean 4*. https://github.com/Timeroot/ComputableReal. 2025.

[14] Charlie Murphy, Patrick Gray, and Gordon Stewart. "Verified Perceptron Convergence Theorem". In: *Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*. 2017, pp. 43–50.

[15] Hoang-Dung Tran et al. "Verification of Deep Convolutional Neural Networks Using ImageStars". In: *Computer Aided Verification*. Ed. by Shuvendu K. Lahiri and Chao Wang. Cham: Springer International Publishing, 2020, pp. 18–42. ISBN: 978-3-030-53288-8.

The End