Weightless Networks for Verification and Control

IWCV, Stresa
May 27, 2026

Christoph Lampert
Institute of Science and Technology, Austria

- public research institute, opened in 2009
- located in outskirts of Vienna

Focus on curiosity-driven basic research

- avoiding boundaries between disciplines
- current 95 research groups
  - Computer Science, Mathematics, Physics, Astronomy, Chemistry, Biology, Neuroscience, Earth and Climate Sciences
- ELLIS unit since 2019

We're hiring!

- interns, PhD students, postdocs, faculty, ...

More information: `chl@ist.ac.at` or `https://cvml.ist.ac.at`

Topics in Our Research Group

Machine Learning Theory

- Transfer Learning (Lifelong Learning, Meta-Learning, Multi-Task Learning)
- Theory of Deep Learning (Neural Collapse, Attention Sinks)

Models/Algorithms

- Differential Privacy
- Continual Learning
- Federated Learning
- Robustness/Fairness

Applications

- LLM Safety/Security
- Logic Gate Networks

Past Topics

- Video Compression
- Document Image Analysis
- Computer Vision
- Kernel Methods
- Structured Prediction
- Semantic Representations

Topics in Our Research Group

Machine Learning Theory

- Transfer Learning (Lifelong Learning, Meta-Learning, Multi-Task Learning)
- Theory of Deep Learning (Neural Collapse, Attention Sinks)

Models/Algorithms

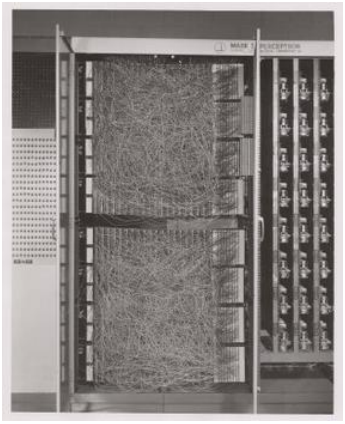
- Differential Privacy
- Continual Learning
- Federated Learning
- Robustness/Fairness

Applications

- LLM Safety/Security
- Logic Gate Networks

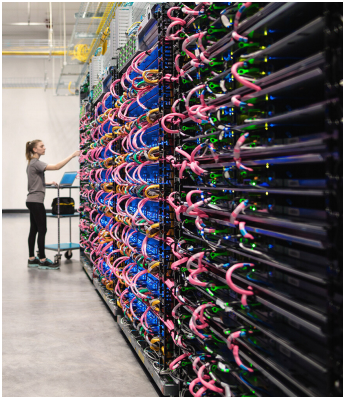
Past Topics

- Video Compression
- Document Image Analysis
- Computer Vision
- Kernel Methods
- Structured Prediction
- Semantic Representations

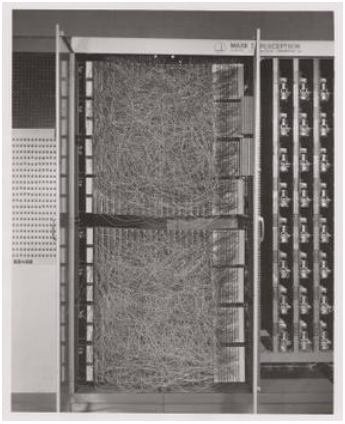


Perceptron:
one inner product

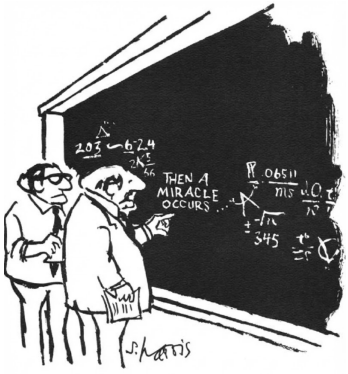
... more inner products ...

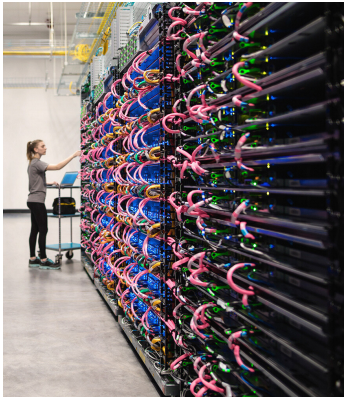


Transformers:
lots and lots of inner products



Perceptron:
one inner product





Transformers:
lots and lots of inner products

Neural Networks without Inner Products: Weightless Neural Networks (aka RAM networks)

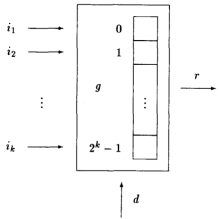


Fig. 1. q-RAM neuron.

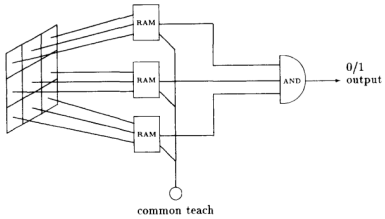


Fig. 2. RAM network with three nodes.

- neurons are lookup tables (LUTs)
  - k binary inputs, 1 binary output $\rightarrow$ table has 2^k entries
  - learning $\equiv$ setting LUT entries $\rightarrow$ no "weights", no inner products
- easy to execute, difficult to train
- going back to at least 1959, made practical in the 1990s, re-popularized in 2020s

---

[W. Bledsoe and I. Browning. "Pattern Recognition and Reading by Machine". Eastern Joint Computer Conference 1959]
Images: [Teresa Ludermir and Wilson R de Oliveira. "Weightless neural models", Computer Standards and Interfaces, 1994]
[Petersen, Borgelt, Kuehne, Deussen. "Deep Differentiable Logic Gate Networks", NeurIPS 2022]
[Bacellar, Susskind, Breternitz Jr, John, John, Lima, Franca. "Differentiable Weightless Neural Networks", ICML 2024]

Special case: $k = 2$: **"Logic Gate Network (LGNs)"**

Each LUT has $2^2 = 4$ entries:

$$f(x_1, x_2) = \begin{cases} \theta_0 & \text{if } (x_1, x_2) = (0, 0) \\ \theta_1 & \text{if } (x_1, x_2) = (0, 1) \\ \theta_2 & \text{if } (x_1, x_2) = (1, 0) \\ \theta_3 & \text{if } (x_1, x_2) = (1, 1) \end{cases}$$

with $\theta = (\theta_0, \theta_1, \theta_2, \theta_3) \in \{0, 1\}^4$.

- each neuron computes a boolean operation
- perfect match to computer hardware
  - 2-LUT: ≤ 12 transistors
  - FP32 multiplication: $\approx 50\,000$ transistors

Table: Boolean Operations

ID	Formula	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	$\neg(B \Rightarrow A)$	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	$B \Rightarrow A$	1	0	1	1
12	$\neg A$	1	1	0	0
13	$A \Rightarrow B$	1	1	0	1
14	$\neg(A \vee B)$	1	1	1	0
15	True	1	1	1	1

Table: adapted from [Petersen *et al.* NeurIPS 2022]

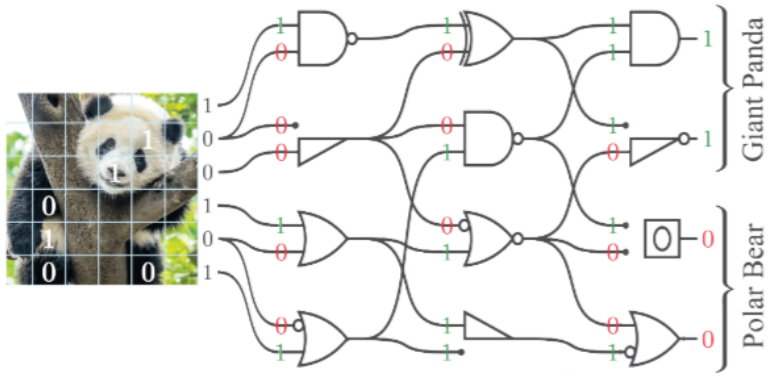


Image: [Petersen *et al*. NeurIPS 2024]

- network consists of stacked layer of gates: $f : \{0,1\}^d \to \{0,1\}^k$
- fully discrete $\to$ how to train?

Observation: learning $\equiv$ identify one of 16 possible configurations per neuron:

$$\underbrace{(0,0,0,0)}_{=:t_1},\ \underbrace{(0,0,0,1)}_{=:t_2},\ \ldots,\ \underbrace{(1,1,1,1)}_{=:t_{16}}$$

Relaxation:

$$\theta = \sum_{j=1}^{16} \alpha_j t_j \quad \text{with} \quad \alpha_j = \frac{e^{a_j}}{\sum_i e^{a_i}} \qquad \text{i.e. } \alpha_j \geq 0 \text{ for all } j, \text{ and } \sum_j \alpha_j = 1)$$

- training time: for each neuron learn $a \in \mathbb{R}^{16}$ "soft assignment"
- deployment time:
  - set $\theta \leftarrow t_j$ with $j = \mathbf{argmax}_i\, a_i$ "hard assignment"
  - compile into boolean representation

[F. Petersen, C. Borgelt, H. Kuehne, O. Deussen. "Deep Differentiable Logic Gate Networks", NeurIPS 2022]

Problem: Logic gates are discrete
$\rightarrow$ gradient does not flow backwards

Solution: in backwards pass, relax logical formulas to real-valued expressions

Example:

- original: $o = x_i \oplus x_j$ (XOR)
- relaxation: $o = x_i + x_j - 2x_i x_j$

$$\frac{\partial o}{\partial x_k} = \begin{cases} 1 - 2x_j & \text{for } k = i \\ 1 - 2x_i & \text{for } k = j \\ 0 & \text{otherwise} \end{cases}$$

(other relaxations are also possible)

Table: Continuos Relaxations

ID	Formula	Relaxation
0	False	0
1	$A \wedge B$	AB
2	$\neg(A \Rightarrow B)$	$A - AB$
3	A	A
4	$\neg(B \Rightarrow A)$	$B - AB$
5	B	B
6	$A \oplus B$	$A + B - 2AB$
7	$A \vee B$	$A + B - AB$
8	$\neg(A \vee B)$	$1 - (A + B - AB)$
9	$\neg(A \oplus B)$	$1 - (A + B - 2AB)$
10	$\neg B$	$1 - B$
11	$B \Rightarrow A$	$1 - B + AB$
12	$\neg A$	$1 - A$
13	$A \Rightarrow B$	$1 - A + AB$
14	$\neg(A \vee B)$	$1 - AB$
15	True	1

Table: adapted from [Petersen *et al*. NeurIPS 2022]

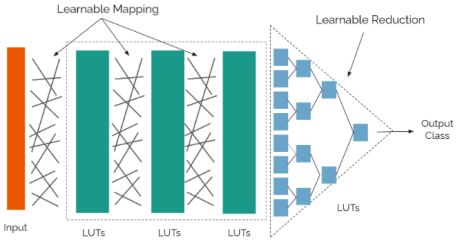
Problem: How to learn network connectivity?

Solution 1: no learning, random connections

[Petersen *et al.*, 2022] $\longrightarrow$ simple and cheap, but suboptimal

Solution 2: parametrize connections as stochastic matrix [Bacellar *et al.*, 2024]

$\longrightarrow$ expensive, especially for wide layers



Solution 3: hybrid [Kresse *et al.*, 2025]

- for each input, keep sparse selection of *candidate* connections
- learn stochastic matrix across candidates
- update candidate set regularly based on gradient signals

[Petersen, Borgelt, Kuehne, Deussen. "Deep Differentiable Logic Gate Networks", NeurIPS 2022]
Image: [Bacellar, Susskind, Breternitz Jr, John, John, Lima, Franca. "Differentiable Weightless Neural Networks", ICML 2024]
[Fabian Kresse, Emily Yu, CHL. "Scalable interconnect learning in boolean networks", arXiv:2507.02585]

Observation: LGNs output are bits. How to get *confidence* in classification?

- use multiple output bits per class:
- for C classes, each class has L output bits:
- outputs: $o_{c,l} \in \{0,1\}$: $c = 1, \ldots, C$, $l = 1, \ldots, L$

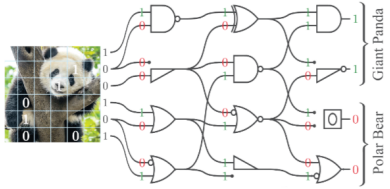


Image: [Petersen *et al*. NeurIPS 2024]

Per-class scores:

$$\text{score}(c) = \sum_{l=1}^{L} o_{c,l} \qquad \text{for } c = 1, \ldots, C$$

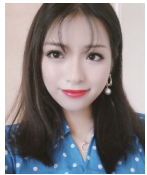
Decision and confidence:

$$f(x) = \underset{c=1,\ldots,C}{\textbf{argmax}}\, \text{score}(c), \qquad \text{conf}(f(x)) = \frac{\text{score}(f(x))}{\sum_j \text{score}(j)}$$

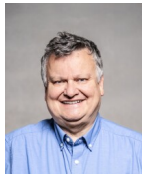
[F. Kresse, E. Yu, T. Henzinger, CHL. *"Logic Gate Neural Networks are Good for Verification"*, NeuS 2025]

"Logic Gate Networks are Good for Verification"

Fabian Kresse
(ISTA)



Emily Yu
(Leiden U)



Tom Henzinger
(ISTA)

[F. Kresse, E. Yu, T. Henzinger, CHL. *"Logic Gate Neural Networks are Good for Verification"*, NeuS 2025]
[F. Kresse, CHL. "Differentiable Weightless Controllers: Learning Logic Circuits for Continuous Control", ICML 2026]

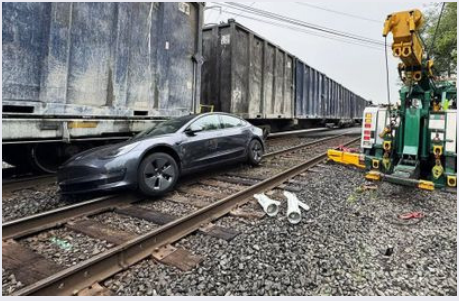
Low Risk Applications

errors are inconvenient but not harmful

entertainment, communication, content creation, productivity tools, ...

High Risk Applications

errors can have catastrophic consequences



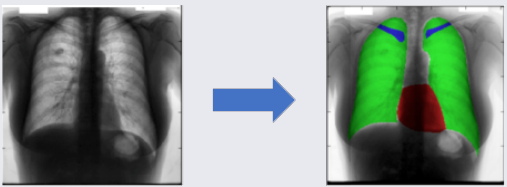
infrastructure (energy, transport), finance, law enforcement, medical decisions, ...

(At least) for high-risk applications, we should *verify* their properties before deploying them.

Images: stable diffusion, futurisms.com

What properties would we like to verify for a deep network?

Decision Making Systems

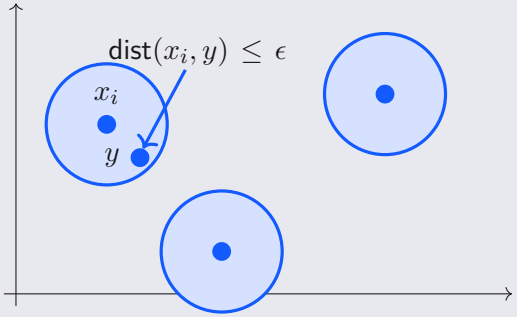


- robustness
- fairness
- compliance with regulations

Control Systems

- robustness
- safety
- reachability

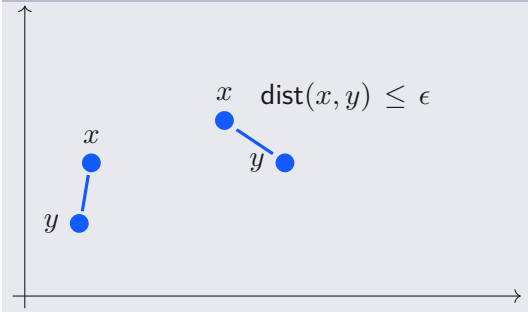
Local Robustness



For $x_1, \ldots, x_n \in \mathcal{D}$:

$\forall y \in \mathcal{X}: \ \text{dist}(x_i, y) \leq \epsilon \ \Rightarrow \ f(y) = f(x_i)$

Global Robustness

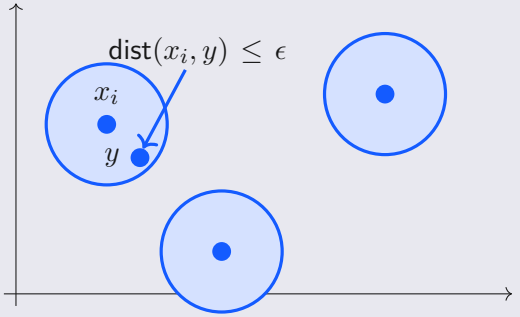


$\forall x, y \in \mathcal{X} : \text{dist}(x, y) \leq \epsilon \ \Rightarrow \ f(y) = f(x)$

Examples:

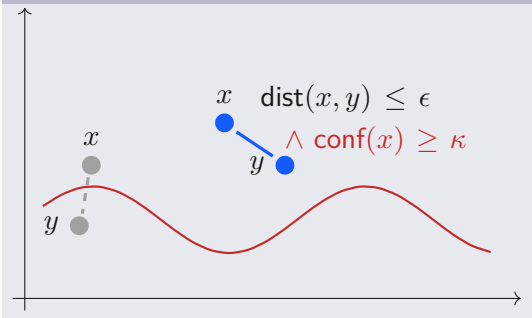
- adversarial examples: $\text{dist}(x, y) = \|x - y\|_{L^p}$
- counterfactual fairness: dist = "same, except protected group membership"

Local Robustness



For $x_1, \ldots, x_n \in \mathcal{D}$:
$\forall y \in \mathcal{X}$: dist$(x_i, y) \leq \epsilon \Rightarrow f(y) = f(x_i)$

Global Robustness with Confidence



$\forall x, y \in \mathcal{X}$: dist$(x, y) \leq \epsilon \ \wedge$ conf$(x) \geq \kappa$
$\Rightarrow f(y) = f(x)$

Examples:

- adversarial examples: dist$(x, y) = \|x - y\|_{L^p}$
- counterfactual fairness: dist = "same, except protected group membership"

How to verify if networks have such properties or not?

Testing

For any input $x \in \mathcal{X}$, try to find $y \in \mathcal{X}$ nearby ($\text{dist}(x, y) \leq \epsilon$) such that $f(x) \neq f(y)$.

In practice: run optimization, e.g. gradient-based (like for adversarial examples)

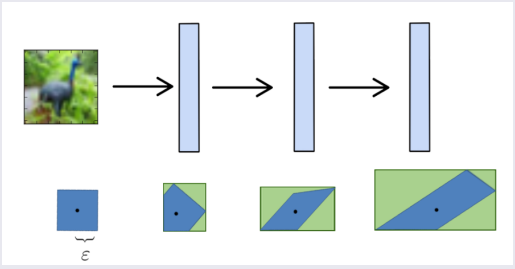
Problem:

- only for local properties
- prone to false negatives, can miss violations

Bound Propagation for ReLU Networks

For any input x,

- propagate *safety region around* x through network
- when region shape changes due to nonlinearities, form outer bounds



Problems:

- prone to false positives
- can verify only local properties

Active research area, many recent extensions, e.g.

- branch-and-bound to tighten relaxation: β-CROWN (Wang *et al.*, 2021)
- non-linearities beyond ReLU: (Shi *et al.*, 2024)

Image: [P. Morawiecki, P. Spurek, M. Smieja, J. Tabor. "Fast and Stable Interval Bounds Propagation for Training Verifiably Robust Models". ESANN 2020]

- $x, y \in \{0,1\}^d$, $u \in \{1, \ldots, C\}$, $\kappa, \epsilon \in \mathbb{N}$, $\quad f : \mathcal{X} \to \mathcal{Y}$ is logic-gate network

SAT-based verification of LGNs [Kresse *et al.* NeUS 2025]

We express robustness property as a boolean formula: $\texttt{formula}(x, y, u, v) :=$

$$\forall x, y, u, v : \quad \texttt{dist}(x, y, \epsilon) \wedge \texttt{conf}(x, \kappa) \wedge \texttt{output}(x, u) \wedge \texttt{output}(y, v) \Rightarrow (u = v)$$

$$f(x) = u \quad \Leftrightarrow \quad \texttt{output(x,u)}$$

$$\text{conf}(x) \geq \kappa \quad \Leftrightarrow \quad \texttt{conf(x,}\kappa\texttt{)}$$

$$\text{dist}(x, y) \geq \epsilon \quad \Leftrightarrow \quad \texttt{dist(x,y,}\epsilon\texttt{)}$$

Verification: check if any example of negation exists (=counterexample)

$$\exists x, y, u, v : \quad \neg\texttt{formula}(x, y, u, v) \qquad \leftarrow \text{SAT(isfiability) problem}$$

- $x, y \in \{0,1\}^d$, $u \in \{1, \ldots, C\}$, $\kappa, \epsilon \in \mathbb{N}$, $\quad f : \mathcal{X} \to \mathcal{Y}$ is logic-gate network

SAT-based verification of LGNs [Kresse *et al.* NeUS 2025]

We express robustness property as a boolean formula: $\texttt{formula}(x, y, u, v) :=$

$$\forall x, y, u, v : \quad \texttt{dist}(x, y, \epsilon) \wedge \texttt{conf}(x, \kappa) \wedge \texttt{output}(x, u) \wedge \texttt{output}(y, v) \Rightarrow (u = v)$$

$$f(x) = u \quad \Leftrightarrow \quad \texttt{output(x,u)}$$

$$\text{conf}(x) \geq \kappa \quad \Leftrightarrow \quad \texttt{conf(x,}\kappa\texttt{)}$$

$$\text{dist}(x, y) \geq \epsilon \quad \Leftrightarrow \quad \texttt{dist(x,y,}\epsilon\texttt{)}$$

Verification: check if any example of negation exists (=counterexample)

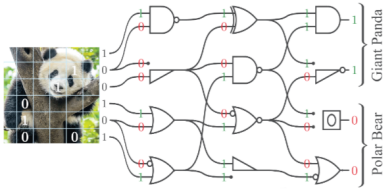
$$\exists x, y, u, v : \quad \neg\texttt{formula}(x, y, u, v) \qquad \leftarrow \text{SAT(isfiability) problem}$$

Concern: SAT solving is NP-hard

- but: extremely effective solvers exist in practice

Example: Representing a LGN as a boolean formula

Inputs: $x_1, x_2, \ldots, x_6 \in \{0, 1\}$



First layer:

$$a_1 = \neg(x_1 \wedge x_2)$$
$$a_2 = x_3$$
$$a_3 = (x_4 \vee x_5)$$
$$a_4 = (\neg x_5 \vee x_6)$$

Second layer:

$$b_1 = (a_1 \oplus a_2)$$
$$b_2 = \neg(a_2 \wedge a_4)$$
$$b_3 = \neg(\neg a_2 \vee a_3)$$
$$b_4 = a_3$$

Output layer:

$$y_1 = (b_1 \wedge b_2)$$
$$y_2 = \neg b_3$$
$$y_3 = 0$$
$$y_4 = (b_3 \vee \neg b_4)$$

Complete expression:

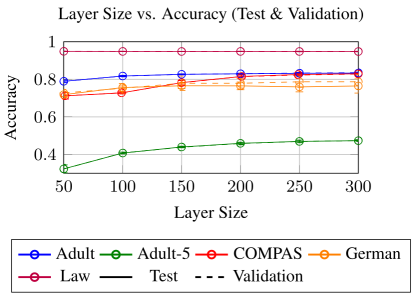
$$y_1 = (\neg(x_1 \wedge x_2) \oplus x_3) \wedge \neg(x_3 \wedge (\neg x_5 \vee x_6))$$
$$y_2 = \neg x_3 \vee x_4 \vee x_5$$
$$y_3 = 0$$
$$y_4 = \neg x_4 \wedge \neg x_5$$

$$\texttt{output}(x, u) = \neg\Big((y_1 \oplus u_1) \vee (y_2 \oplus u_2) \vee (y_3 \oplus u_3) \vee (y_4 \oplus u_4)\Big)$$

Dataset	**#Samples**	**#Features** (Categorical/Numeric)	**#Classes**
German Credit	1,000	16 (12/4)	2
Adult	46,033	7 (4/3)	2
Law School	21,982	9 (5/4)	2
COMPAS	60,798	8 (7/1)	3
Adult-5 Class	195,665	7 (4/3)	5

Models:

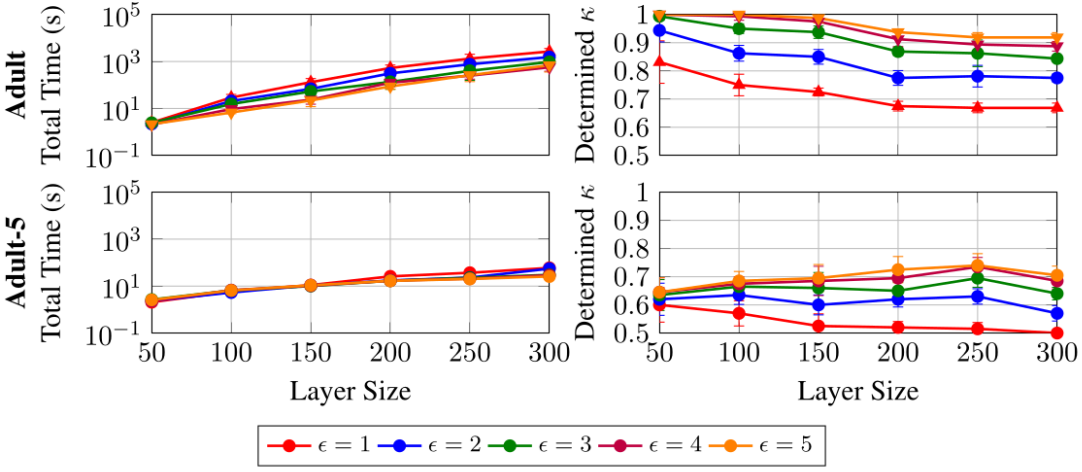
- LGN with 3 layers
- layer size: $\{50, 100, 150, 200, 250, 300\}$
- Adam optimizer, lr 0.01, 200 epochs
- decent accuracy (on simple benchmarks)



Verification:

- encode properties as SAT formulas
- decide SAT/UNSAT using *Kissat* solver (Biere *et al.*, 2024)

[A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt. SAT Competition, 2024]


Adult
Total Time (s)
10^5
10^3
10^1
10^{-1}
Determined κ
1
0.9
0.8
0.7
0.6
0.5
Adult-5
Total Time (s)
10^5
10^3
10^1
10^{-1}
Determined κ
1
0.9
0.8
0.7
0.6
0.5
50
100
150
200
250
300
Layer Size
50
100
150
200
250
300
Layer Size
$\epsilon = 1$
$\epsilon = 2$
$\epsilon = 3$
$\epsilon = 4$
$\epsilon = 5$

Summary

- We can learn networks with boolean tables instead of inner product units.
  - at least for $k = 2$ (LGNs) and if they are reasonably small
- many properties of Logic Gate Networks can be written as (big) boolean formulas.
- we can use SAT solvers to verify properties
  - at least if the LGNs are reasonably small

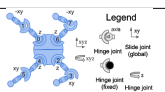
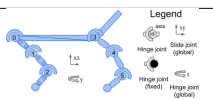
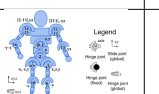
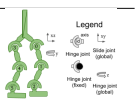
Differentiable Weightless Controllers: Learning Logic Circuits for Continuous Control

Fabian Kresse

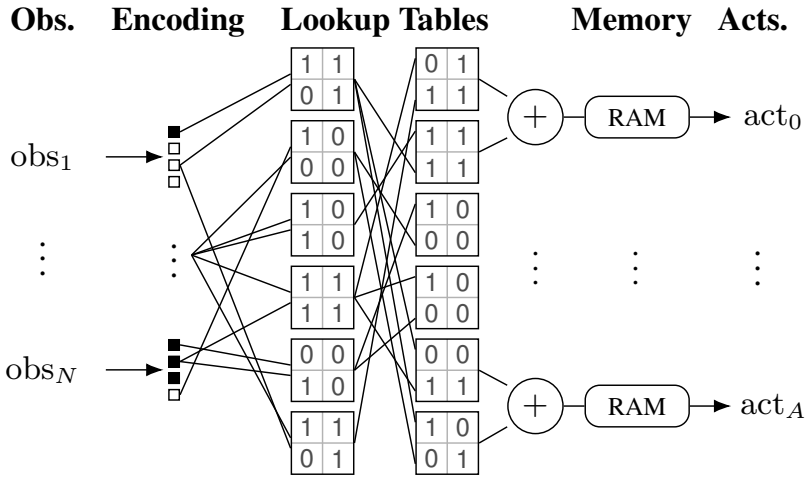
[F. Kresse, CHL. "Differentiable Weightless Controllers: Learning Logic Circuits for Continuous Control", ICML 2026]

Task: learning continuous controllers, i.e. Reinforcement Learning

Table: Five MuJoCo Locomotion Tasks

Task	Ant	HalfCheetah	Hopper	Humanoid	Walker2D
					
state dim	27	17	11	348	17
action dim	8	6	3	17	6
rewards	forward movement, staying alive, penalties for extreme actions				

[E. Todorov, T. Erez, Y. Tassa. "MuJoCo: A physics engine for model-based control". IROS 2012]
[T. Haarnoja, A. Zhou, P. Abbeel, S. Levine. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor", ICML 2018]


Obs.
Encoding
Lookup Tables
Memory
Acts.
obs$_1$
obs$_N$
RAM
RAM
act$_0$
act$_A$

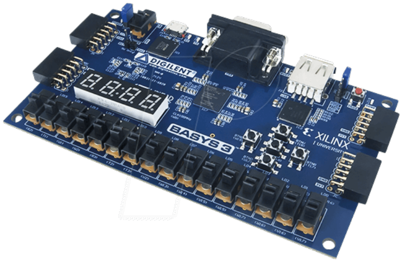
Baselines: standard **FP32** networks, **quantized** networks (2-8 bits)

Table: Policy returns (median and the 25% and 75% quantiles over 10 trained models)

Environment	FP	Quant	DWC
Ant	$5.6\text{k}_{[4.3k, 5.8k]}$	$4.7\text{k}_{[3.9k, 4.9k]}$	$\mathbf{5.7k}_{[5.5k, 5.9k]}$
HalfCheetah	$\mathbf{11.5k}_{[10.1k, 11.9k]}$	$10.5\text{k}_{[9.6k, 11.0k]}$	$7.5\text{k}_{[7.1k, 7.9k]}$
Hopper	$2.8\text{k}_{[2.1k, 3.3k]}$	$1.9\text{k}_{[1.1k, 3.3k]}$	$\mathbf{3.1k}_{[2.8k, 3.4k]}$
Humanoid	$\mathbf{6.2k}_{[6.0k, 6.7k]}$	$6.0\text{k}_{[5.8k, 6.1k]}$	$6.1\text{k}_{[5.8k, 6.6k]}$
Walker2D	$\mathbf{5.0k}_{[4.7k, 5.2k]}$	$4.7\text{k}_{[4.4k, 5.0k]}$	$5.0\text{k}_{[4.5k, 5.2k]}$

- DWC returns are comparable to reference and Quant baseline, except HalfCheetah

[F. Kresse, CHL. "Learning quantized continuous controllers for integer hardware", L4DC 2026]



Xilinx Artix XC7A15T-FGG484-1

Table: FPGA Specifications

Resource	Quantity
LUTs	10,400
Flip-flops	20,800
DSPs	45
Block RAM (36 Kb)	25
Clock Speed	100 MHz

DWCs can run on extremely small hardware.

Table: FPGA resource utilization on 100 MHz Xilinx Artix-7 (power estimated by Xilinx Toolchain)

	Environment	LUTs	FFs	BRAM	DSP	Latency	Power	Throughput	Energy per action
$l = 256$	Ant	0.8k	0.5k	0	0	0.01 µs	0.105 W	1.0×10^8	1.1×10^{-9} J
	HalfCheetah	0.8k	0.4k	0	0	0.01 µs	0.105 W	1.0×10^8	1.1×10^{-9} J
	Hopper	0.9k	0.3k	0	0	0.01 µs	0.116 W	1.0×10^8	1.2×10^{-9} J
	Humanoid	0.9k	1.1k	0	0	0.01 µs	0.102 W	1.0×10^8	1.0×10^{-9} J
	Walker2D	0.8k	0.4k	0	0	0.01 µs	0.107 W	1.0×10^8	1.1×10^{-9} J
$l = 1024$	Ant	3.2k	1.7k	0	0	0.02 µs	0.225 W	1.0×10^8	2.3×10^{-9} J
	HalfCheetah	3.0k	2.2k	0	0	0.03 µs	0.208 W	1.0×10^8	2.1×10^{-9} J
	Hopper	3.2k	2.0k	0	0	0.03 µs	0.228 W	1.0×10^8	2.3×10^{-9} J
	Humanoid	3.2k	3.7k	0	0	0.02 µs	0.219 W	1.0×10^8	2.2×10^{-9} J
	Walker2D	2.8k	2.1k	0	0	0.03 µs	0.206 W	1.0×10^8	2.1×10^{-9} J
Quant	Ant	2.7k	4.5k	3	45	2.29 µs	0.39 W	4.4×10^5	8.9×10^{-7} J
	HalfCheetah	4.3k	4.6k	15	11	243.23 µs	0.33 W	4.1×10^3	8.0×10^{-5} J
	Hopper	2.4k	2.0k	0	45	0.21 µs	0.31 W	4.8×10^6	6.5×10^{-8} J
	Humanoid	2.3k	3.1k	1.5	45	15.36 µs	0.33 W	6.5×10^4	5.1×10^{-6} J
	Walker2D	1.9k	1.6k	2	4	162.23 µs	0.17 W	6.2×10^3	2.8×10^{-5} J

Table: FPGA resource utilization on 100 MHz Xilinx Artix-7 (power estimated by Xilinx Toolchain)

HalfCheetah	3.0k	2.2k	0	0	0.03 µs	0.208 W	1.0×10^8	2.1×10^{-9} J
Hopper	3.2k	2.0k	0	0	0.03 µs	0.228 W	1.0×10^8	2.3×10^{-9} J
Humanoid	2.3k	3.1k	1.5	45	15.36 µs	0.33 W	6.5×10^4	5.1×10^{-6} J
Walker2D	1.9k	1.6k	2	4	162.23 µs	0.17 W	6.2×10^3	2.8×10^{-5} J

Latency

10-30 ns

(∼ single clock-cycle)

Throughput

10^8

actions / sec

Energy Usage

∼1 nJ

per action

DWCs can be orders of magnitude more efficient even than standard (even aggressively quantized) networks

Summary

- We can learn controllers with lookup table neurons instead of inner product units.
  - at least for $k \leq 6$ inputs and if the networks are reasonably small
- Controller quality on par with floating point architectures
  - at least on four out of five MuJoCo tasks (HalfCheetah seems capacity-limited)
- DWCs can run on very small hardware (specifically FPGAs).
  - ultra-low latency (here: 1–3 clock cycles, 10–30ns from observation to action)
  - ultra-high throughput (here: 10^8 actions per second)
  - ultra-low energy usage (here: nanojoule-level per action)
- DWCs allow for elementary interpretability (e.g. feature thresholds used)

Next steps:

- putting things together: learn controllers with provable guarantees