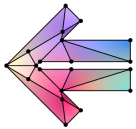
Neuro-Symbolic Learning with Relational Logic via Differentiable Semantics

Gustav Šír et al.

Czech Technical University in Prague

CGL&BL 2026



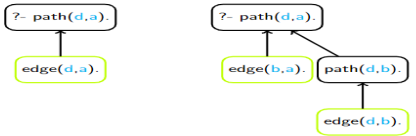
Outline

Outline

Symbolic vs. Neural AI

Symbolic AI

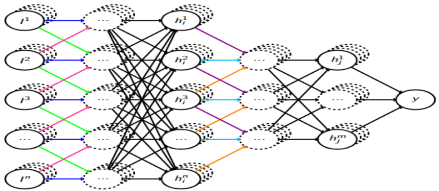
- discrete representations
- \+ data efficient
- \+ interpretable
- \+ systematic generalization
- reasoning, planning, ...




```
path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z), path(Z,Y).
```

Neural AI

- continuous representations
- \+ computation efficient
- \+ captures uncertainty
- \+ robust to noise
- perception, learning, ...



Deep Learning

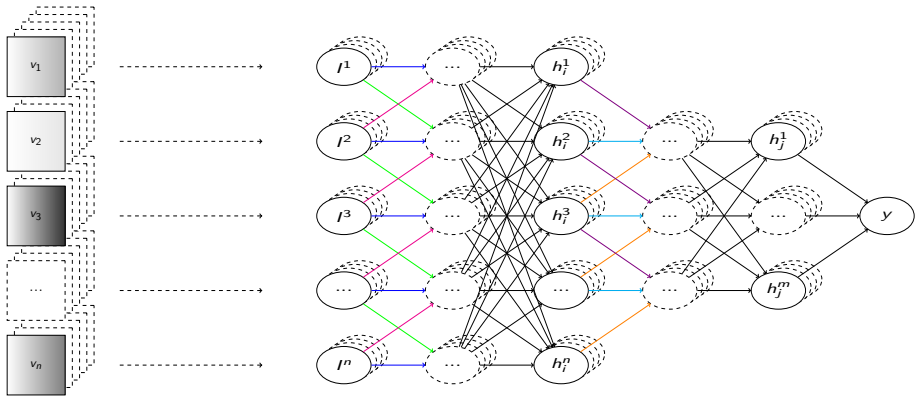


Figure 1: An example deep (convolutional) network ingesting tensor samples.

Symbolic Learning

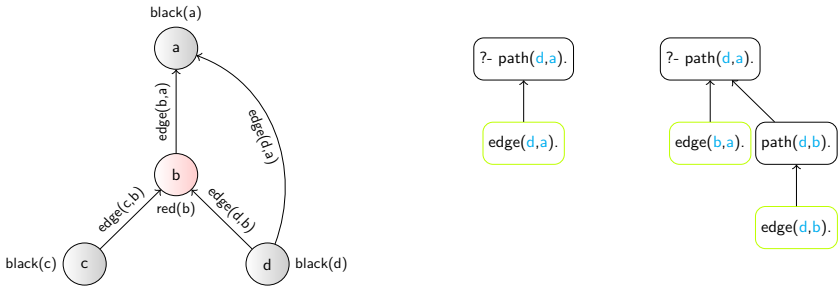
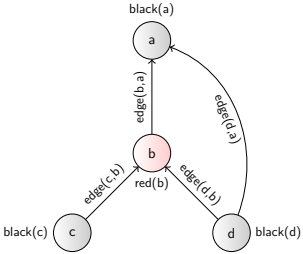
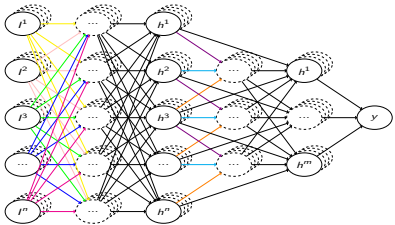


Figure 2: A graph structure encoded in relational logic (left), with two possible proof trees of the query $path(d, a)$ derived from it (right) through the program:

```
1 path(X,Y) :- edge(X,Y).
2 path(X,Y) :- edge(X,Z), path(Z,Y).
```

Problem Statement





$$
\begin{aligned}
&1\ \ \mathsf{h}_1(\mathsf{X},\mathsf{Y}) \ :-\ \mathsf{b}_0(\mathsf{X},\mathsf{Y}).\\
&2\ \ \ldots\\
&3\ \ \mathsf{h}_n(\mathsf{X},\mathsf{Y}) \ :-\ \mathsf{b}_1(\mathsf{X},\mathsf{Z}),\ \ \ldots,\ \ \mathsf{b}_2(\mathsf{Z},\mathsf{Y}).
\end{aligned}
$$

Statistical vs. Relational Learning Representations

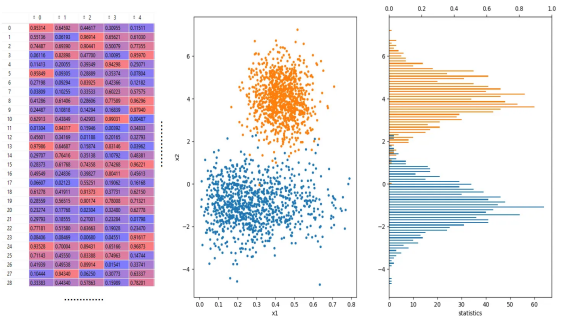


Figure 3: Statistical Learning.

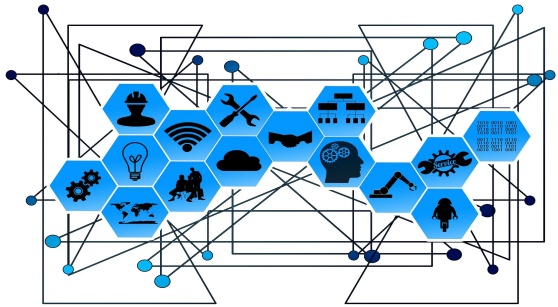


Figure 4: Relational Learning.

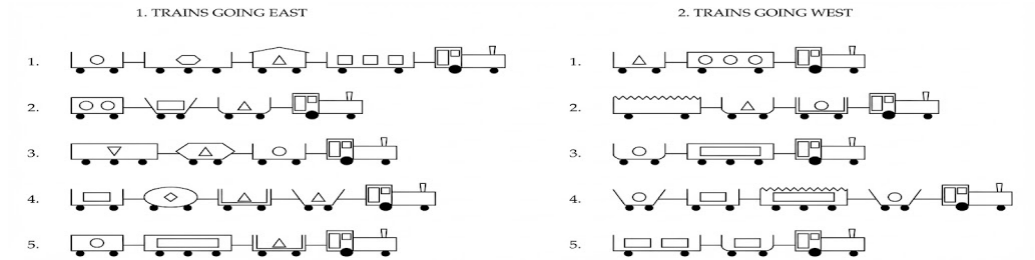
Outline

Why Logic for Learning?

Most machine learning operates on attribute-value (*propositional*) representations.

The Problem of Structure

How do you represent a molecule, a social network, or a train structure with a variable number of cars, using a fixed-size feature vector? Use the language of **logic** instead!



Logic Programming: A Relational Formalism

In symbolic ML, we use **Logic Programming** (Prolog/Datalog) language representations,

- a subset of FOL tailored for computation (unique minimal model and efficient inference).

Key Components:

- **Constants:** Objects in the domain (e.g., 'train_1', 'car_a').
- **Variables:** Placeholders for any object (e.g., 'X', 'Y'). *Crucial for generalization!*
- **Predicates:** *Relations* between objects. Form logical atoms/facts on the domain.
  - 'has_car(train_1, car_a)' – a fact about structure
  - 'shape(car_a, rectangle)' – a fact about attribute
- **Horn Clauses (Rules):**

$$p_0(t_1, \ldots, t_n) \leftarrow p_1(t_1, \ldots, t_m) \wedge p_2(t_1, \ldots, t_l) \wedge \ldots$$

Inductive Logic Programming (ILP)

Standard ML View: Learning is approximating a function $f(x) \approx y$.
Logical View: Learning is finding a hypothesis H that explains observations E given background knowledge B.

The Task (Inverse Entailment): Given:

- **Background Knowledge (B):** Facts and rules we already know.
- **Positive Examples (E^+):** Things that should be true.
- **Negative Examples (E^-):** Things that should be false.

Find a **Hypothesis (H)** (a Logic Program) such that:

$$\forall e \in E^+ : B \wedge H \models e \quad \text{and} \quad \forall e \in E^- : B \wedge H \not\models e$$

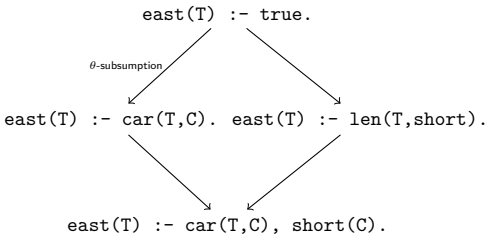
Historical Note: GUHA

Before ILP, there was **GUHA** (1960s, Hájek et al.).

- *"Mechanizing Hypothesis Formation"*
- First logic-based approach to automated data mining.
- Established the tradition of using logic to structure data analysis.

Hypothesis Search: The Lattice

How do we find the rules (program)? We search a **Subsumption Lattice** ordered by generality.



Search Strategies:

1. **Top-Down (Specialization):** Start with general rule, add literals to exclude negative examples.
2. **Bottom-Up (Generalization):** Start with specific example, find Least General Generalization (LGG).

Learned Hypothesis (H)

```
eastbound(T) :- has_car(T, C), length(C, short), closed(C).
```

Summary of ILP

Symbolic Strengths

- **Expressive:** Variables handle varying structure sizes (trains with 2 vs 10 cars).
- **Data Efficient:** Learns from few examples.
- **Interpretable:** H is human-readable.

Symbolic Weaknesses

- **Combinatorial Explosion:** The lattice is huge; discrete search is NP-Hard.
- **Discrete:** Cannot use gradient descent.
- **Brittle:** Fails if "rectangle" is slightly "trapezoid" (noise).

Outline

Propositionalization and Graphicalization

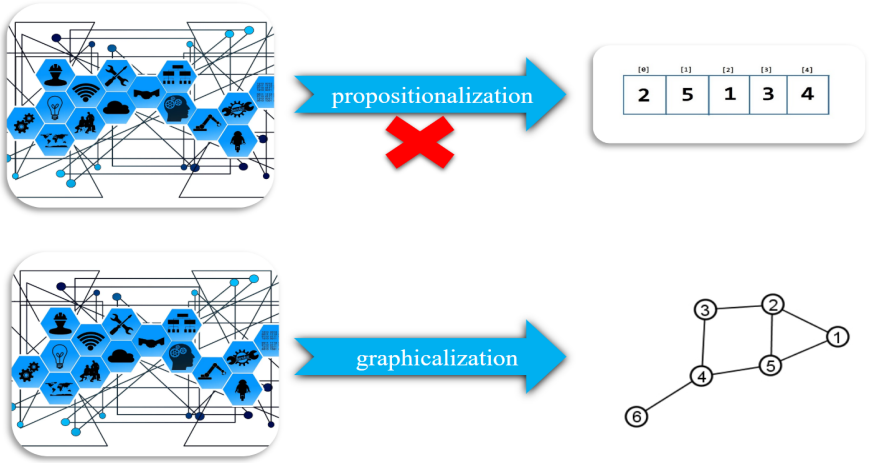
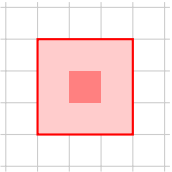


Figure 5: Propositionalization and Graphicalization

From Grids to Graphs: Generalizing Convolutions

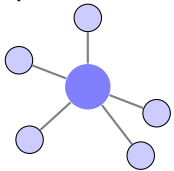
How do we process data structures with neural networks?

CNN (Euclidean Grid)



Fixed Neighbors (Up, Down...)
Translation Invariance

GNN (Arbitrary Graph)



Variable Neighbors
Permutation Invariance

The Generalization

A pixel image is just a graph where every node has exactly 8 neighbors.

- CNNs share weights by **Translation** (Kernel slides over grid).
- GNNs share weights by **Permutation** (Message function applies to all edges).

Graph Neural Networks

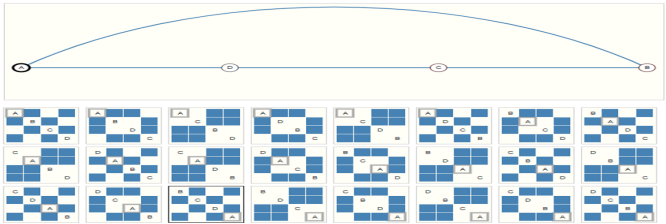
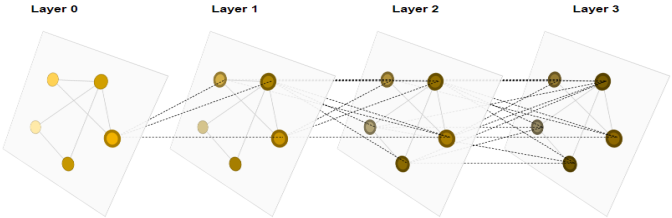
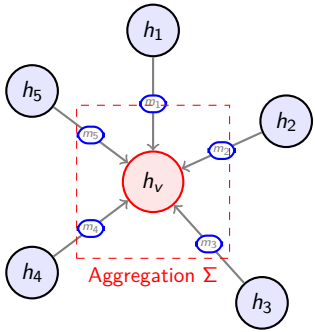


Figure 6: A high-level view of Graph Neural Networks (source: distill.pub)



The Mechanism: Neural Message Passing

GNNs learn by aggregating information from each node's local neighborhoods:



Step 1: Message

$$m_{uv} = \phi_\theta(h_u, h_v, e_{uv})$$

NN ϕ computes messages for each edge.

Step 2: Aggregate

$$a_v = \sum_{u \in \mathcal{N}(v)} m_{uv}$$

Summing makes it order-independent.

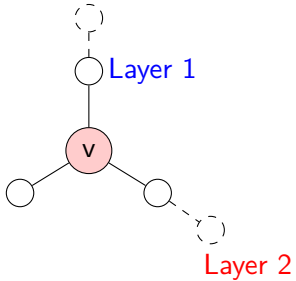
Step 3: Update

$$h'_v = \psi_\omega(h_v, a_v)$$

Neural Network ψ updates node state.

Going Deep: The Receptive Field

A single GNN layer aggregates information from **1-hop** neighbors. Stacking L layers allows a node to see L-hops away.



Key Concept: The Computation Graph

- The neural network structure is **dynamic**.
- It unfolds based on the input graph's topology.
- This is very different from MLPs/CNNs where the computation graph is fixed.

This dynamic structure unfolding is what connects GNNs to Logic (Proof Trees).

Outline

The Logical View: GNNs as C_2 Formulas

GNNs are not arbitrary function approximators. Their expressiveness is strictly bounded by the **2-Variable Fragment of First-Order Logic (C_2)** with counting.

The C_2 Fragment:

- Logic using only two variable names, e.g., x and y.
- Includes counting quantifiers $\exists^{\geq k}$.

The Correspondence (Grohe, 2021):

- **Node Feature:** Predicate $P(x)$.
- **Aggregation ($\sum$):** Corresponds to Counting Quantifiers.

$$\text{sum}(\{h_u \mid u \in \mathcal{N}(v)\}) \approx \exists^{\geq k} y.(E(x,y) \wedge \phi(y))$$

- **Layer Update:** Corresponds to Boolean Connectives / Refinement.

A GNN layer learns to evaluate a C_2 formula.

The Logical View: How Expressive are GNNs?

GNNs are powerful, but they are **not** universal function approximators on graphs.

Theorem (Grohe, 2021; Xu et al., 2019)

The expressive power of standard GNNs is bounded by the **1-Weisfeiler-Lehman (1-WL)** graph isomorphism test.

Logical Correspondence:

- GNNs $\equiv$ **2-Variable Fragment of First-Order Logic (C_2)** with counting quantifiers.
- GNNs $\equiv$ **Graded Modal Logic** ($\Diamond_k \varphi$: "There exist at least k neighbors where φ holds.").

Intuition: GNNs view the graph as a collection of trees rooted at each node.

The Limits of C_2 Logic (The Triangle Problem)

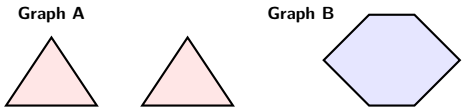
Why is C_2 (2 variables) a limitation? **We cannot detect cycles (e.g., triangles).**

To define a triangle:

$$\exists x, y, z : \text{edge}(x, y) \wedge \text{edge}(y, z) \wedge \text{edge}(z, x)$$

Requires 3 distinct variables to "close the loop".

GNN / 1-WL View:



Indistinguishable! Locally, every node sees "2 neighbors who each have 2 neighbors..."

Conclusion: GNNs are efficient (differentiable) but logically shortsighted.

Outline

Encoding Planning States to Graphs

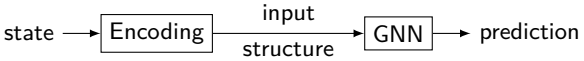


Figure 7: Procedural flow of Encoding Planning States into Graphs

Example: consider a planning state s over the set of objects $O = \{t, l_1, l_2\}$ as the following set of ground atoms: $s = \{N, T(t), L(l_1), L(l_2), A(t, l_1), A_G(t, l_2), R(t, l_1, l_2)\}$.

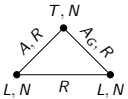


Figure 8: An object edge-typed graph built from the state s.

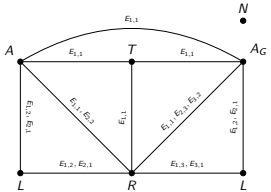


Figure 9: An atom edge-typed graph corresponding to the state s.

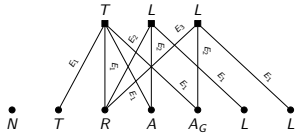


Figure 10: An object-atom edge-typed graph corresponding to the state s.

GNN Expressiveness is limited

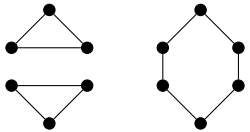


Figure 11: Two example C_2-equivalent graphs

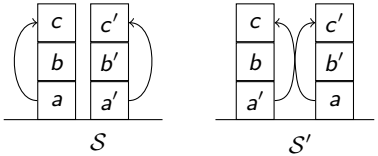


Figure 12: Two C_2-equivalent states in the blocksworld

Expressiveness of GNNs encodings

Table 1: Numbers of undistinguishable state pairs over planning domains and GNN architectures.

Models				O^G	O^{MG}	O-A^{BG}	O-A^{BMG}	A^G	A^{MG}	$\hat{A}^{MG}$	O-P^G	O-P^{MG}
GCN [?]				222,04	30,09							
SAGE [?]				249,78	32,66	54,80	61,94	71,72	4,54	21,80	340,03	31,48
GIN [?]				227,46	24,55			142,20	35,05	42,41	18,49	17,73
GINE [?]				12,22	61,33			40,19	4,55	1,67	26,46	23,31
GATv2 [?]				32,56	22,03	99,19	78,21	50,85	6,38	4,18	7,64	6,27
NN [?]				29,40	38,16			44,26	3,72	1,03	5,34	1,73
Transformer [?]				34,55	30,44	102,30	66,82	49,12	3,64	1,65	12,66	9,89
GEN [?]				29,46	32,06	54,83	65,96	53,74	1,75	2,05	1,30	1,00
Domains	$\overline{\#objects}$	$\overline{\#states}$	$\overline{V^*}$									
barman	23,3	45,5	4,2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
floortile	16,0	43,0	10,9	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
freecell	51,1	21,3	1,9	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
gripper	17,0	89,1	10,0	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
logistics	19,9	100,0	13,5	0,00	0,00	0,15	0,25	0,00	0,00	0,00	0,05	0,30
parcprinter	46,0	61,2	11,0	59,83	60,17	57,83	57,83	61,83	84,50	77,00	7,50	7,33
pegsol	33,0	37,2	9,8	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
rovers	41,4	91,5	5,7	0,00	0,00	0,04	0,04	0,17	0,55	0,55	0,00	0,00
satellite	32,9	80,0	5,8	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,21	0,79
scanalyzer	13,0	66,5	3,6	87,00	153,50	6,75	8,00	27,75	1,25	27,75	6,50	6,00
sokoban	111,6	116,2	27,2	644,55	695,85	1243,15	1508,00	1209,35	8,95	19,26	1,00	0,67
tidybot	27,4	64,1	3,7	0,05	0,05	0,05	0,05	0,00	0,00	0,00	0,10	0,00
tpp	16,9	87,7	4,4	0,00	0,18	0,00	0,00	0,00	0,00	0,00	0,00	0,00
vacuum-sep	21,8	101,0	1,6	1,50	1,10	4,95	4,10	4,95	1,35	0,00	5,20	3,55
vacuum-sh	21,8	101,2	1,5	3,75	0,40	7,40	4,65	10,60	2,65	0,20	13,80	10,10
avg. instance eval. runtime (s)				1,17	1,47	3,34	2,90	6,11	6,50	7,38	11,72	13,74

*R. Horcik, G. Sir: *Expressiveness of GNNs in Planning domains*; ICAPS, 2024 (best paper runner-up)

GNN encoding performance

In practice, small and natural encodings (Gaifmann) seem to work best.

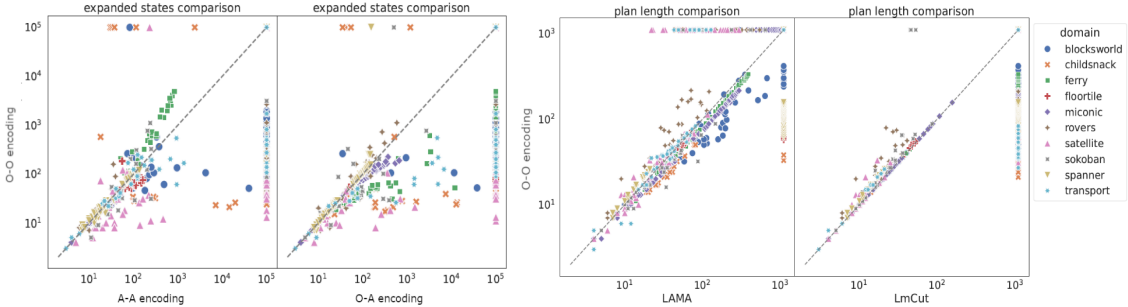
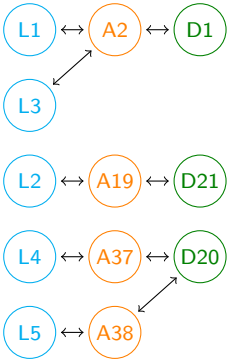


Figure 13: Informativeness of the GNN encodings (left) and comparison with LAMA/LMCut (right).

*R. Horcik, G. Sir, V. Simek, T. Pevny: *State Encodings for GNN-based Lifted Planners*; AAAI, 2025 (oral)

Outline

Relational Deep Learning: Graphicalization of Databases...



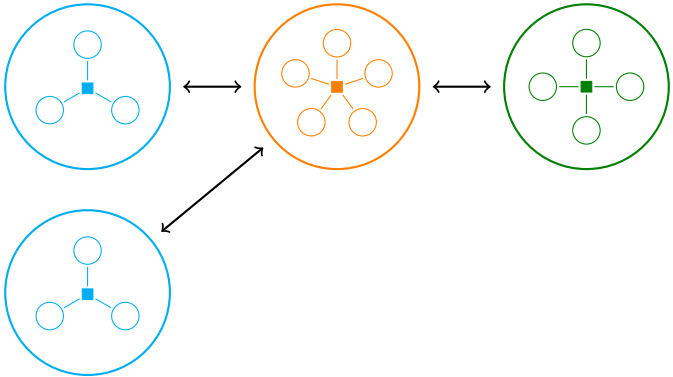
Account Id	Date	Amount	Status
2	1994-01-05	80952	A
19	1996-04-29	30276	B
2	1997-12-08	30276	A
37	1998-10-14	318480	D
38	1998-04-19	110736	C
...	...	...	...

Account Id	District Id	Frequency	Date Created	...
2	1	Monthly	1993-02-26	...
19	21	Monthly	1995-04-07	...
37	20	Monthly	1997-08-18	...
38	20	Weekly	1997-08-08	...
...	...	...	...	...

District Id	District	Location	...
1	Prague	Prague	...
20	Strakonice	S Bohemia	...
21	Tabor	S Bohemia	...
...	...	...	...

Relational Deep Learning

Our DB Representation: Two-level Multi-relational **Hypergraph**.



*L. Zahradnik, L. Neumann, G. Sir: *A Deep Learning Blueprint for Relational Databases*; TRL@NeurIPS, 202

Deep Relational Learning workflow

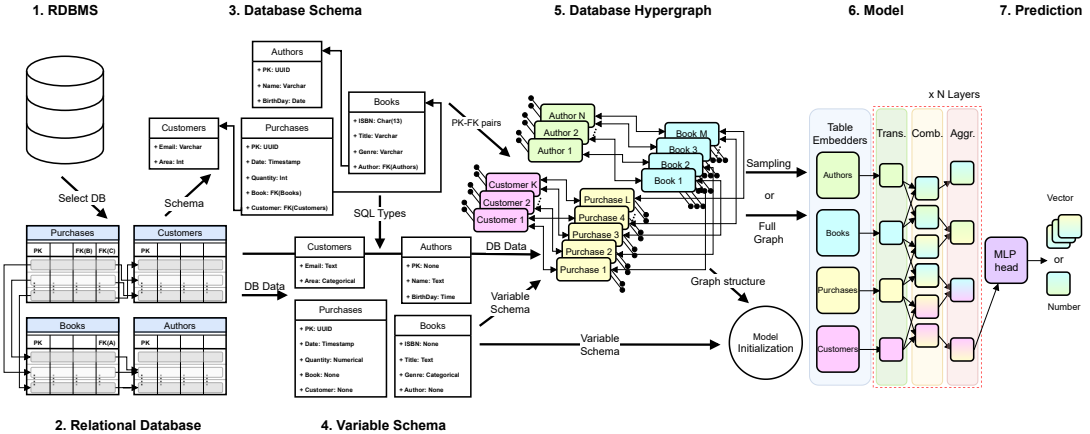


Figure 14: Deep Relational Learning workflow

*J. Peleska, G. Sir: *Tabular Transformers meet Relational Databases*; ACM TIST, 2025

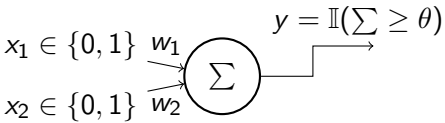
Outline

Outline

Origins: From Logic Gates to Geometry (1943–1957)

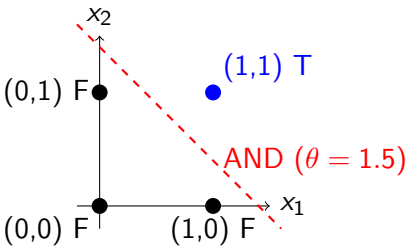
1. McCulloch & Pitts (1943): *"A Logical Calculus of the Ideas Immanent in Nervous Activity"*

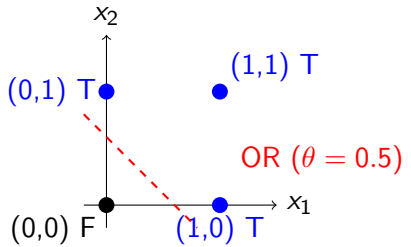
- Neurons are **Logic Gates**.
- $y = \mathbb{I}(\sum w_i x_i \geq \theta)$.



2. Rosenblatt (1957): The perceptron.

- Learning Logic = Finding a hyperplane.
- Weights determine the connective.





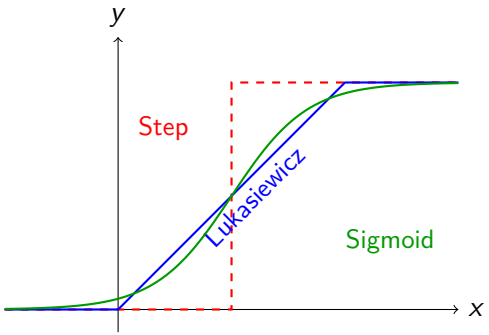
Linearly Separable Functions

Relaxing Logic: Fuzzy Semantics

Linear Separability:

- AND, OR are linearly separable, **XOR** is not (famous Minsky & Papert, 1969).
- *We need multi-layer networks to represent complex (propositional) logic formulae.*

Problem: Standard logic is discrete/boolean $\{0, 1\}$, and step functions have no gradient. Neural Networks are continuous $[0, 1]$ and require gradient. **Solution:** Fuzzy Logic.



e.g., Lukasiewicz Logic ($[0, 1]$):

- **Conjunction (T_L):** $\max(0, x + y - 1)$
- **Disjunction (S_L):** $\min(1, x + y)$
- **Negation:** $1 - x$

This can be seen as a clamped ReLU. Also, the Sigmoid function is a smooth approximation of the Lukasiewicz truth function.

Neural Networks as Fuzzy Logic Machines

Analogy: Loosely speaking, neural networks can be seen as functional approximators of
Petr Hájek's Basic Logic (BL) (Metamathematics of Fuzzy Logic, 1998).

- BL is the fundamental logic of all continuous t-norms.
- The form of t-norm then reflects the network's *inductive bias*.

1. The Process Parallel:

- **Forward Pass ≈ Deduction**
  Computing the consequences (truth values) of the current model/theory.
- **Backward Pass ≈ Model Finding**
  Gradient Descent searches a model that satisfies the constraints (data).

2. The Structural Parallel:

Logic (BL-Extension)	**Neural Architecture**
Lukasiewicz Logic $T(x,y) = \max(0, x+y-1)$	**ReLU** activations standard interactions
Product Logic $T(x,y) = x \cdot y$	**Gating / Attention** multiplicative interactions
Gödel Logic $T(x,y) = \min(x,y)$	**Max/Min-Pooling** aggregative interactions

Outline

Knowledge Based Artificial Neural Networks

Towell & Shavlik (1994): The first major system to exploit this connection.
Idea: Use a *Propositional* Logic Theory to initialize the Neural Network's parameters.

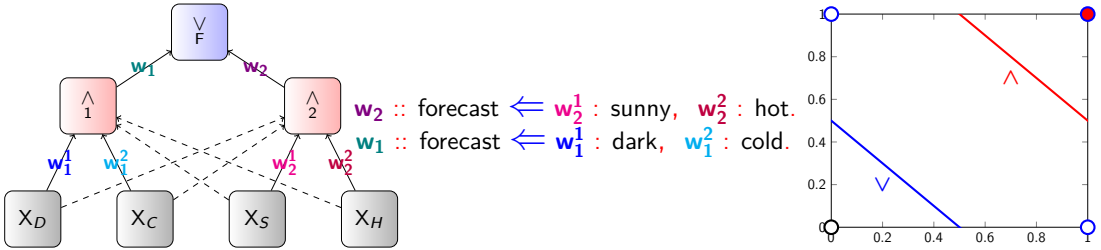


Figure 15: An example correspondence between a neural network (left) and a propositional logic program (middle), based on conjunctive and disjunctive activations emulated by the respectively weighted neurons (right), in the spirit of KBANN.

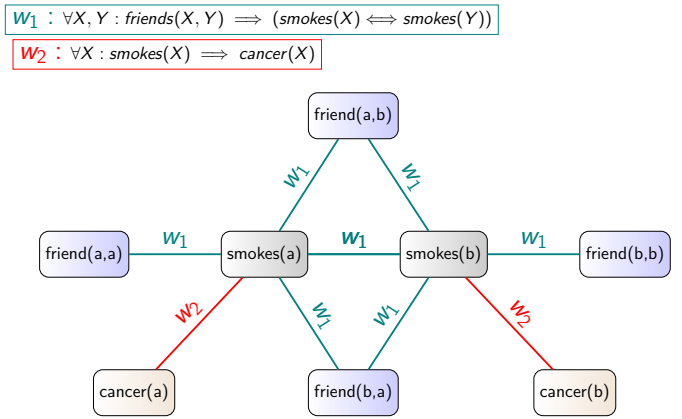


Figure 16: A ground Markov Logic Network unfolded from the given template for two constants $\{a, b\}$.

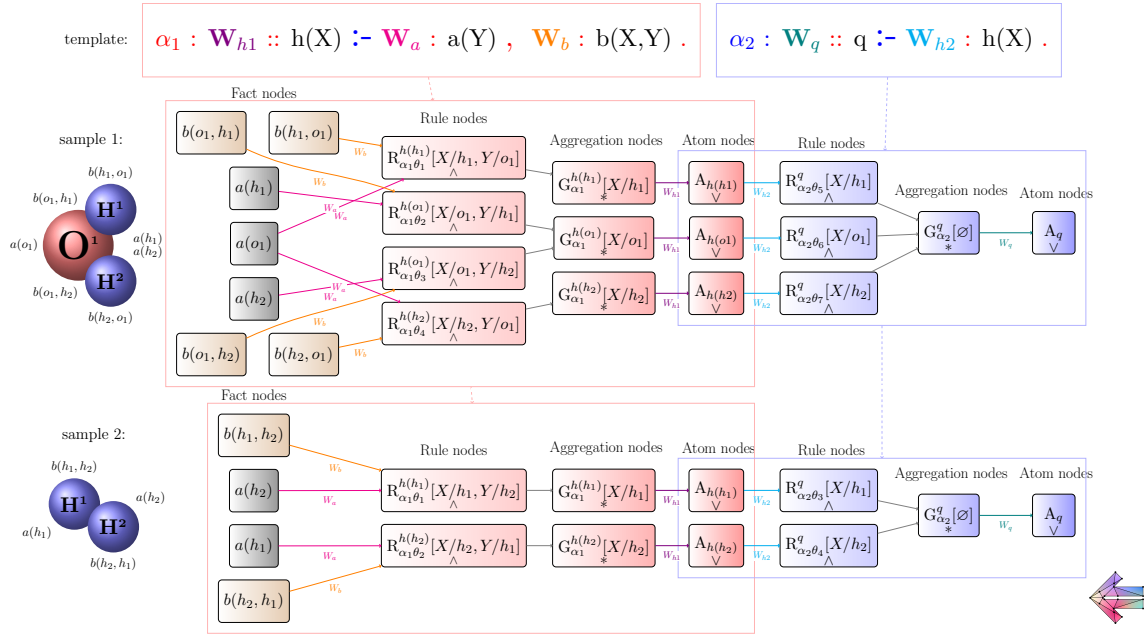
Lifted Relational Neural Networks

Syntax

- Template: set of *weighted definite clauses* $\mathcal{T} = \{C_1, \ldots, C_l\}$
  - $C_i = \mathbf{W} :: \mathrm{h}_1(\ldots) \Leftarrow \mathbf{W_1} : \mathrm{b}_1(\ldots)\ ,\ \ \ldots\ ,\ \mathbf{W_m} : \mathrm{b}_m(\ldots).$
- Examples: are also sets $\mathcal{E}$ of weighted (unit) clauses (facts)
  - $\mathcal{E}_j = \mathbf{V_1} :: \mathrm{f}_1(\ldots)\ ,\ldots\ ,\mathbf{V_n} :: \mathrm{f}_n(\ldots)$
- Instead of "inputing" an example into model, we create $\mathcal{N}_j = \mathcal{T} \cup \mathcal{E}_j$

Semantics

1. Grounding of $\mathcal{N}_j$ as $\overline{\mathcal{N}_j} = \{C\theta \mid C \in \mathcal{N}_j, C\theta \in G(\mathcal{N}_j)\}$
   - Limited to the *least Herbrand model* $G(\mathcal{N}_j)$ of $\mathcal{N}_j$
2. Transforming the ground logical model $\overline{\mathcal{N}_j}$ into a neural model
   - mapping to **Fact**, **Rule**, **Aggregation**, **Atom** computation nodes (will be demonstrated)
3. Updating the weights in the neural model via gradient descent
   - corresponding to the clause weights in the template


template:
$\alpha_1 : \mathbf{W}_{h1} :: \mathrm{h(X)} \text{ :- } \mathbf{W}_a : \mathrm{a(Y)}\ ,\ \mathbf{W}_b : \mathrm{b(X,Y)}\ .$
$\alpha_2 : \mathbf{W}_q :: \mathrm{q} \text{ :- } \mathbf{W}_{h2} : \mathrm{h(X)}\ .$
sample 1:
Fact nodes
Rule nodes
Aggregation nodes
Atom nodes
Rule nodes
Aggregation nodes
Atom nodes
sample 2:
Fact nodes
Rule nodes
Aggregation nodes
Atom nodes
Rule nodes
Aggregation nodes
Atom nodes

Outline

Declarative ANN Encoding

A *propositional* logic template has a *single* Herbrand model and thus corresponds to a single *static* neural model.

$$\mathbf{w_1} :: \text{forecast} \Leftarrow \mathbf{w_1^1} : \text{dark}, \quad \mathbf{w_1^2} : \text{cold}.$$
$$\mathbf{w_2} :: \text{forecast} \Leftarrow \mathbf{w_2^1} : \text{sunny}, \quad \mathbf{w_2^2} : \text{hot}.$$

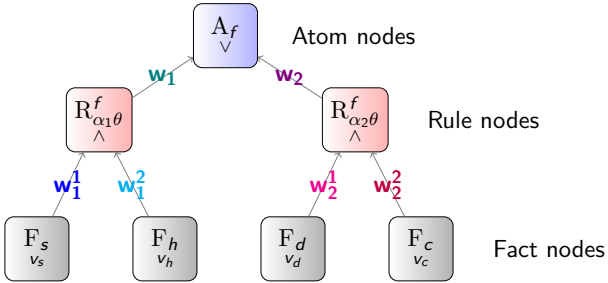


Figure 18: An example propositional model in LRNNs in the style of KBANN.

Convolutional Networks

Relational logic templating in LRNNs provides more expressiveness:

```
1 h ⇐ w_l: f(A), w_m: f(B), w_r: f(C), next(A,B), next(B,C).
```

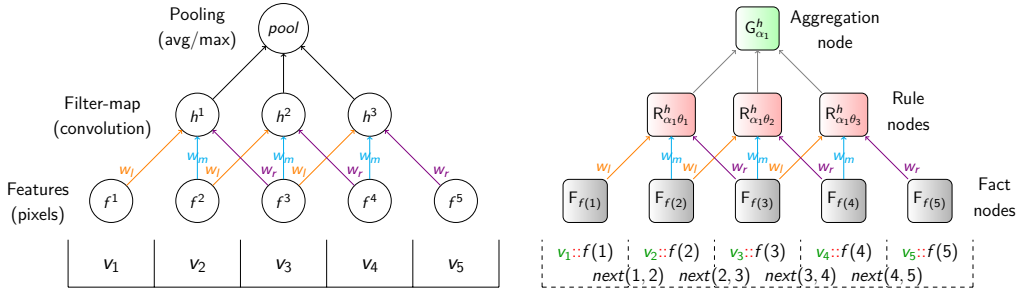


Figure 19: Left: core part of a standard CNN architecture. Right: the same with 1 rule in LRNNs.

GDL Intermezzo: the perspective of Symmetries

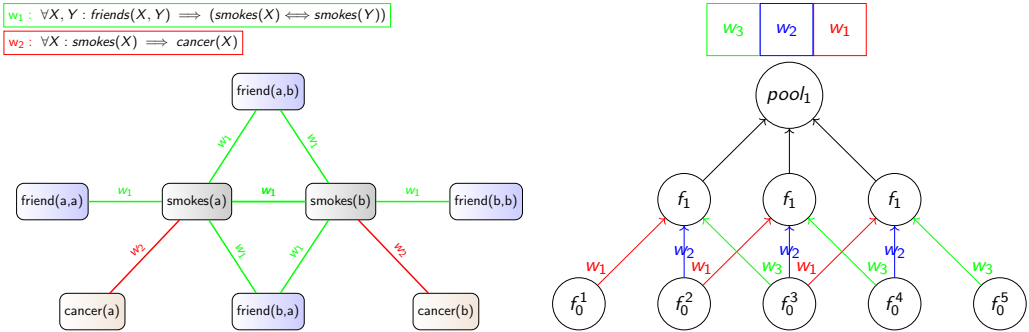


Figure 20: Illustration of two forms of relational expressiveness, with CNNs (right) capturing a spatial pattern of a 3-neighborhood, and an MLN (left) capturing a social pattern of smoking habits amongst friends, both reflected by the symmetries in the respective ground model computations.

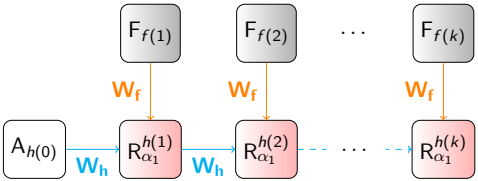
Recurrent and Recursive Networks

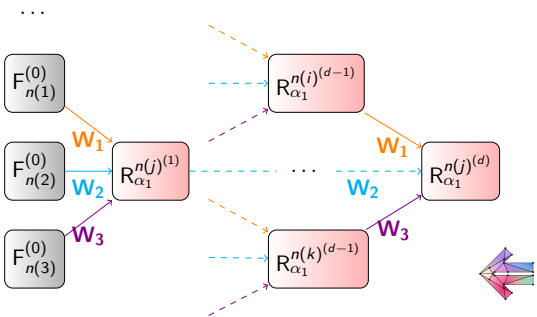
Recurrent net:

```
1 h(Y) ⇐ Wf : f(Y), Wh : h(X), next(X,Y).
```

Recursive net:

```
1 n(P) ⇐ W1 : n(C1), W2 : n(C3), W3 : n(C3), parent(P,C1,C2,C3).
```





Outline

Graph Neural Networks

$$1\ \ \mathrm{h}^{(i)}(\mathrm{V}) \Leftarrow \mathbf{W}_1^{(i)} : \mathrm{h}^{(i-1)}(\mathrm{U}),\ \mathtt{edge}(\mathrm{V},\mathrm{U}).$$
$$2\ \ \mathrm{h}^{(i)}(\mathrm{V}) \Leftarrow \mathbf{W}_2^{(i)} : \mathrm{h}^{(i-1)}(\mathrm{V}).$$

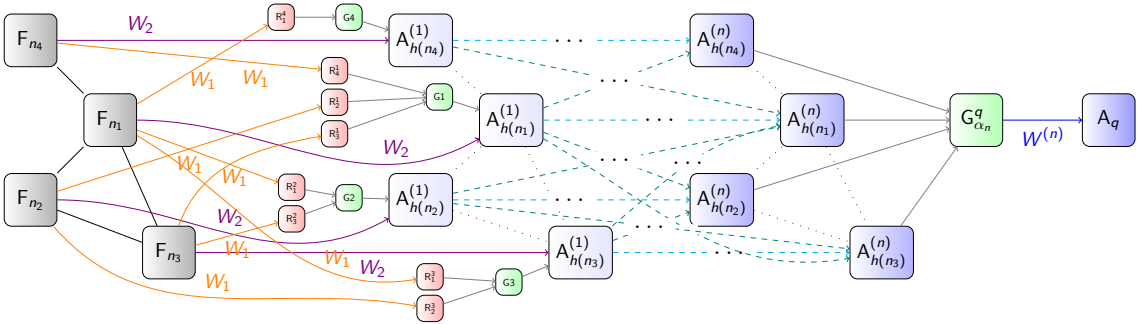


Figure 22: A sample (graph-SAGE) GNN applied to a 4-node input graph ($\mathrm{F}_{n_1} \ldots \mathrm{F}_{n_4}$) in LRNNs.

Beyond GNNs

Of course, LRNNs are not designed to just emulate GNNs. One can use:

- heterogeneous/multi-relational graphs
- hypergraphs, relational databases
- various sub-graph, meta-path and meta-graph GNNs
- completely new message passing schemes (beyond WL/C2)
- (soft) pattern matching
- symbolic deduction
- and more...

Please find the framework at: https://github.com/LukasZahradnik/PyNeuraLogic

G. Sir, O. Kuzelka, F. Zelezny: *Beyond Graph Neural Networks with Lifted Relational Neural Networks*; Machine Learning, 2020

Outline

Outline

Beyond GNNs with Atom Rings

Declaring a ring is simple:

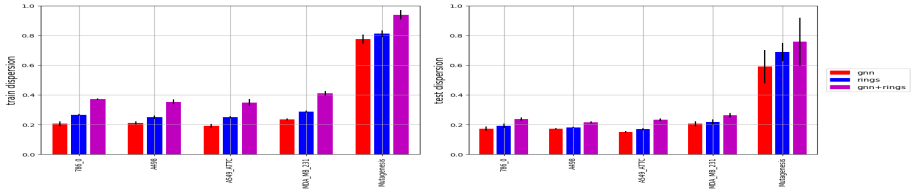
$$\_\mathrm{ring}_6(A,\ldots,F) \Leftarrow \mathrm{bond}(A,B),\ \ldots,\ \mathrm{bond}(E,F),\ \mathrm{bond}(F,A).$$

its distributed representation can then be easily aggregated:

$$\mathbf{W_r} :: \mathrm{ring}_6^{(n)}(A,\ldots,F) \Leftarrow \_\mathrm{ring}_6(A,\ldots,F),\ \mathbf{W_a}:\mathrm{atom}^{(n)}(A),\ \ldots,\ \mathbf{W_f}:\mathrm{atom}^{(n)}(F).$$

and propagated:

$$\mathbf{W_{a'}} :: \mathrm{atom}^{(n)}(A) \Leftarrow \mathbf{W_{r'}}:\ \mathrm{ring}_6^{(n-1)}(A,\ldots,F).$$



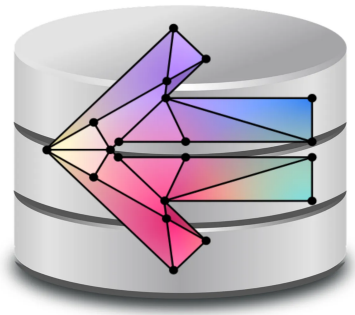
G. Sir, O. Kuzelka, F. Zelezny: *Learning with molecules beyond GNNs*; ML4Molecules@NeurIPS, 2020

Deep Database learning with LRNNs

Table: atom

atom_id	molecule_id	element	type	charge
d100_1	d100	c	22	-0.128
d100_10	d100	h	3	0.132
d100_11	d100	c	29	0.002
d100_12	d100	c	22	-0.128
...	...	...	...	...

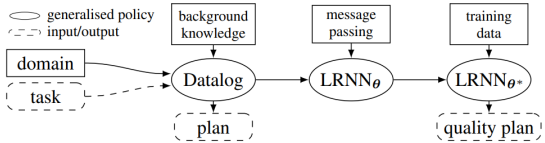
```
R.atom("d100_1", "d100")[-0.128]
R.atom("d100_10", "d100")[0.132]
R.atom("d100_11", "d100")[0.002]
R.atom("d100_12", "d100")[-0.128]
...
```



R.message_2(X) <= R.message_1(Y) & R.edge(X, Y)

= *Join tables 'message1' and 'edge' on the column Y, group by X, and aggregate the resulting values into a new table/view 'message2'*

Figure 23: Workflow of deep learning for generalized planning with LRNNs.



An example neuro-symbolic learning policy for the Blocksworld domain:

$$\mathrm{m}(A) \leftarrow \mathrm{clear}(A), \mathrm{arm\_empty}$$

$$\mathrm{well\_placed}(A) \leftarrow \mathrm{on}_{ag}(A, B), \mathrm{well\_placed}(B)$$
$$\mathrm{well\_placed}(A) \leftarrow \mathrm{on\_table}_{ag}(A)$$
$$\mathrm{constructive}(A) \leftarrow \mathrm{m}(A), \mathrm{on}_{ug}(A, B), \mathrm{well\_placed}(B), \mathrm{clear}(B)$$
$$\mathrm{constructive}(A) \leftarrow \mathrm{m}(A), \mathrm{on\_table}_{ug}(A)$$

$$\mathrm{unstack}^*(A, B) \leftarrow \mathrm{constructive}(A) \quad \text{(B1)}$$
$$\mathrm{unstack}^*(A, B) \leftarrow \neg\mathrm{well\_placed}(A), \neg\mathrm{constructive}^{\exists} \quad \text{(B2)}$$
$$\mathrm{stack}^*(A, B) \leftarrow \mathrm{on}_{ug}(A, B), \mathrm{well\_placed}(B) \quad \text{(B3)}$$
$$\mathrm{pickup}^*(A) \leftarrow \mathrm{constructive}(A) \quad \text{(B4)}$$
$$\mathrm{putdown}^*(A) \leftarrow \neg\mathrm{stack}^{\exists} \quad \text{(B5)}$$

LRNN planning

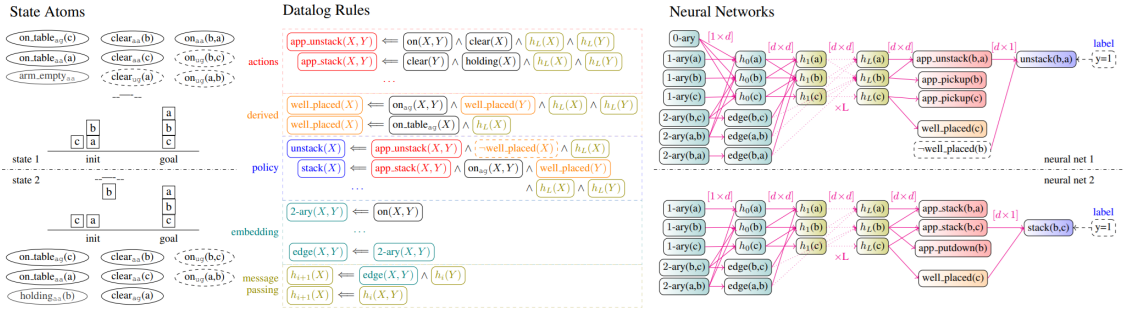


Figure 24: Visualisation of the LRNN architecture. Logical representations of two states (left) form inputs into the Datalog program (middle) that induces differentiable computation graphs (right, partially displayed) for predicting action scores.

D. Chen, R. Horcik, G. Sir: *Deep Learning for Generalised Planning with Background Knowledge*; arxiv, 2024

Outline

Compressing GNN symmetries

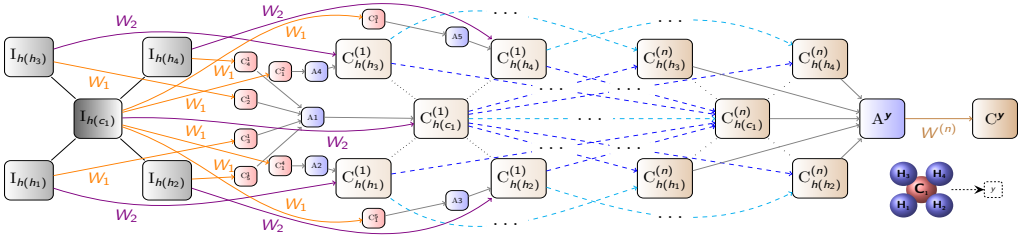
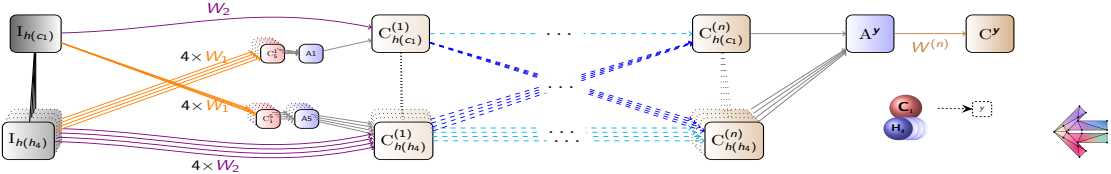


Figure 25: Compressing symmetries in GNN-like models for speedup.

G. Sir, O. Kuzelka, F. Zelezny:*Lossless Compression of Structured Convolutional Models via Lifting*; ICLR,2020



Computing Performance

Using symbolic (lifting) techniques can speed up even standard GNNs!

- but there is a HW & SW lottery in play

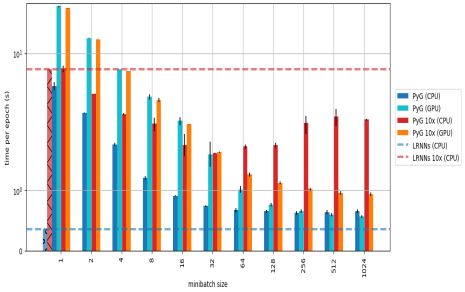


Figure 26: Increasing tensor parameter and batch sizes over CPU/GPU.

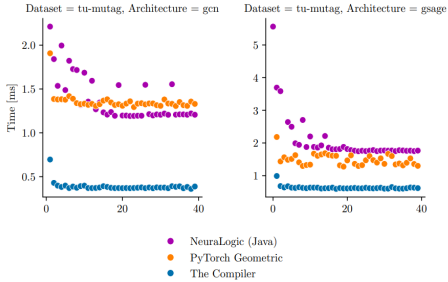


Figure 27: A full batch variant using a more recent, custom vectorizer/compiler.

J. Neumann: *Scaling Up Deep Relational Learning*; MSc thesis, Czech Technical University, 2023

G. Sir: *A Computational Perspective on Neural-Symbolic Integration*; NSAI, 2024

Conclusion

You can find some
more explanations at https://medium.com/@sir.gustav

Please find the framework on Github:
github.com/GustikS/NeuraLogic
with a Python frontend
at github.com/LukasZahradnik/PyNeuraLogic

We hope you will find it useful for designing
your own *relational neurosymbolic learning ideas!*