Neuro-Symbolic Inductive Logic Programming with Logical Neural Networks

Authors: Prithviraj Sen, Breno W. S. R. de Carvalho, Ryan Riegel, Alexander Gray

The Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI-22)

Presented by:
Pavel Dolin
Matthew Marquez
Lucas Saldyt

Outline

- Motivation
- Proposed architecture
- Architecture performance
- LNN review
- ILP-LNN Specific Details
- Future work and limitations

Motivation

Motivation - Learn Rules from Noisy Data

ILP task description:

> “In inductive logic programming, given a dataset, a set of starting view definitions, and a target predicate, we can infer the view definition of the target predicate.”

The novelty of the paper:

> Perform ILP with Logical Neural Nets (LNN)

Model tldr:

1. Assume a rule template for a given task (data)
2. Learn LNN (predicates in the template)
3. Enjoy first order logical interpretability

Kautz Taxonomy:

> Neuro:Symbolic →Neuro

https://web.stanford.edu/~vinayc/logicprogramming/html/inductive_logic_programming.html

Proposed Architecture

Proposed Architecture - Toy Example

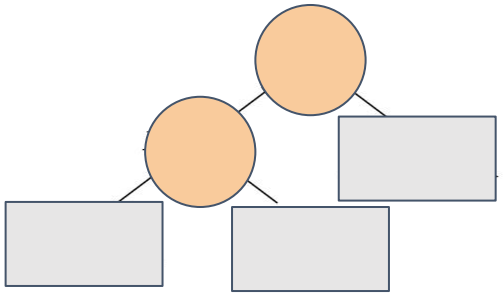
1. Start with a toy DB

A	X	Y
a_1	1	2
a_2	1	5

B	X	Y
b_1	1	2

C	W	Z
c_1	2	5

2. Propose a rule template



- Structure is a tree that connects template placeholder predicates
  - Placeholder predicate at root of tree is target predicate and is labeled
- Forward pass
  - Input: Truth value assignments for every fact in the KB
  - Output: Truth value predictions for facts across all possible target predicates

Proposed Architecture - Overview

- Learning problem statement:
  - Given:
    - A collection of first order logic (FOL) predetermined predicates in KB
    - Labels for possible target predicate facts
    - A logic program template containing placeholders for predicates
  - Find:
    - An assignment of KB predicates to base placeholder predicates

Proposed Architecture - Toy Example

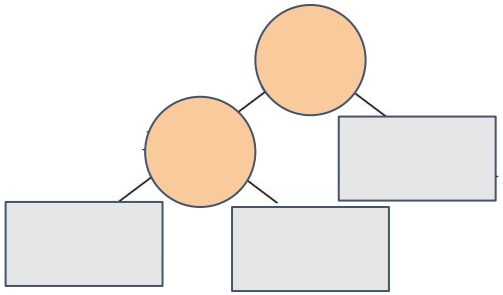
1. Start with a toy DB

A	X	Y
a_1	1	2
a_2	1	5

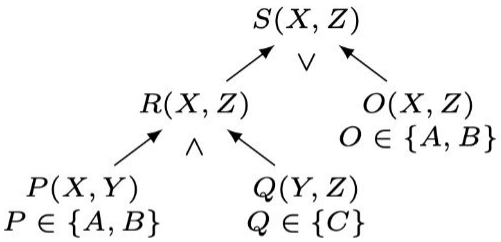
B	X	Y
b_1	1	2

C	W	Z
c_1	2	5

2. Propose a rule template



3. Learn the predicates
&
get generated facts



P	X	Y
p_1	1	2
p_2	1	5

Q	Y	Z
q_1	2	5

O	X	Z
o_1	1	2
o_2	1	5

PQ	X	Y	Z
pq_1	1	2	5

R	X	Z
r_1	1	5

S	X	Z
s_1	1	5
s_2	1	2

Proposed Architecture - Overview

- Assignment reveals that data in KB can be represented by a logic program
- Problem can also be seen as learning structure **among** predicates since we explore various placements of predicates in the tree

Architecture Performance

Experiments

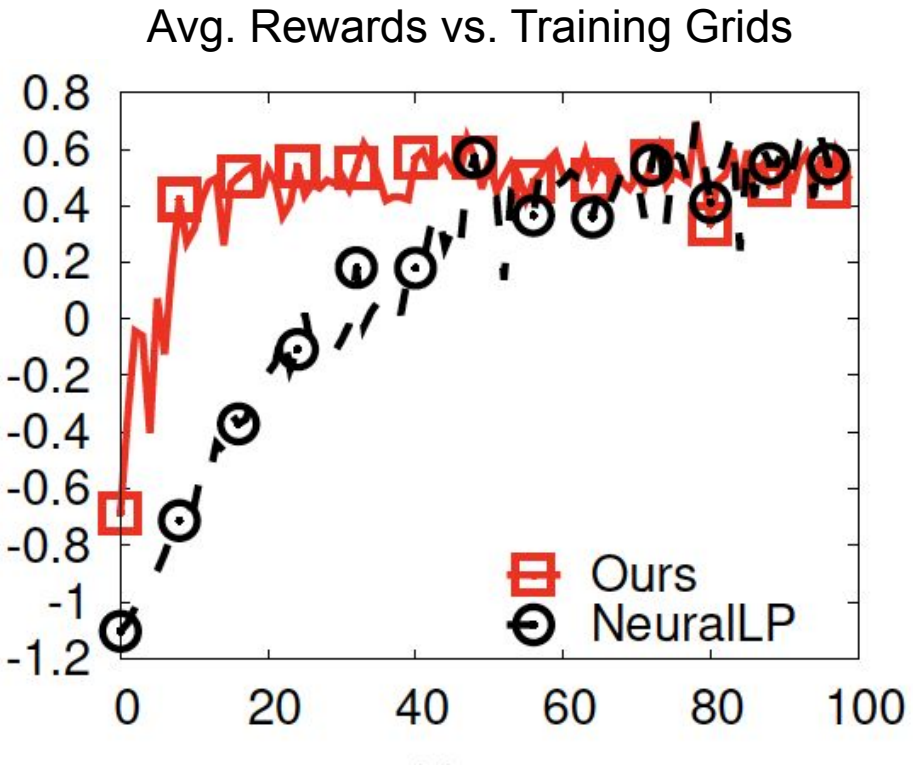
Knowledge Base Completion

		NeuralLP	DRUM	CTP	RNNLogic	Ours
Kinship	Hits@10	89.1	86.1	93.9	91.1	**98.4**
	Hits@3	63.0	48.2	79.7	72.9	**89.3**
	MRR	48.8	40.0	70.3	64.5	**81.9**
UMLS	Hits@10	93.0	97.9	97.0	91.1	**99.4**
	Hits@3	75.4	91.2	91.0	82.1	**98.3**
	MRR	55.3	61.8	80.1	71.0	**90.0**
WN18RR	Hits@10	50.2	52.1	–	53.1*	**55.5**
	Hits@3	43.3	44.6	–	47.5*	**49.7**
	MRR	33.7	34.8	–	45.5*	**47.3**
FB15K-237	Hits@10	32.8	33.1	–	44.5*	**47.0**
	Hits@3	19.8	20.0	–	31.5*	**34.2**
	MRR	17.4	17.5	–	28.8*	**30.7**

Name	Vertices	Predicates	Facts	Queries
UMLS	135	49	5216	661
Kinship	104	26	10686	1074
WN18RR	40945	11	86835	3134
FB15K-237	14505	237	272115	20466

Experiments

Grid World



LNN Review

LNN Review - Concepts

- Designed to provide interpretable logic in neural nets
  - Mapping between neurons and logical atoms and operators
  - Activation functions of non-primary nodes help determine which operator is used
- Uses a bounds-based weighted probabilistic logic
  - Weights are learned during backward pass
- Bounds are improved during inference
  - Inference itself uses upward and downward passes **within** the forward pass

LNN Review - Concepts

Table 1: Primary truth value bound states

Bounds	Unknown	True	False	Contradiction
Upper	$[\alpha, 1]$	$[\alpha, 1]$	$[0, 1-\alpha]$	Lower > Upper
Lower	$[0, 1-\alpha]$	$[\alpha, 1]$	$[0, 1-\alpha]$	

Riegel, Ryan, et al. "Logical neural networks." *arXiv preprint arXiv:2006.*13155 (2020).

LNN Review - Constrained Activations

- Activation functions are constrained to match “characteristic shapes” of traditional logic functions
  - Weighted t-norms with constraints on weights typically are used to implement this
  - Tailored activation function learns weights automatically such that characteristic shapes are met

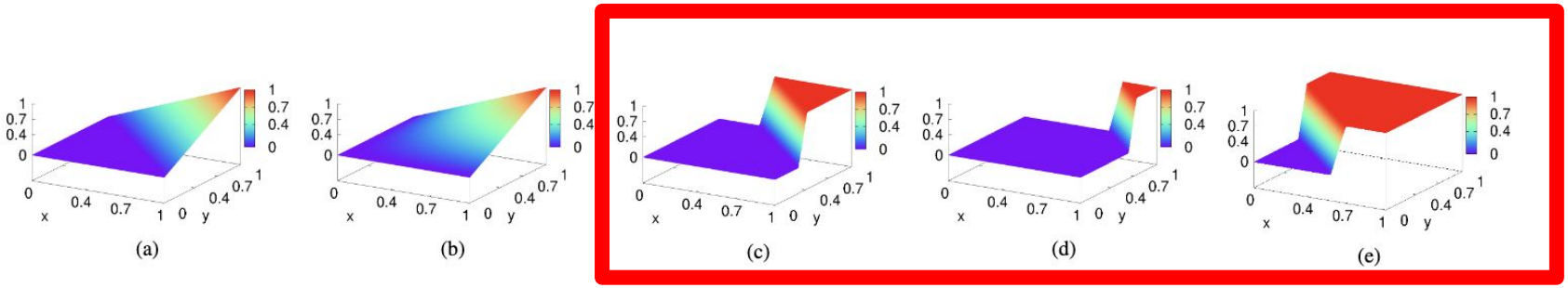


Figure 2: (a) Lukasiewicz t-norm vs. (b) Product t-norm vs. LNN-$\wedge$ with (c) $\alpha = 0.7$, (d) $\alpha = 0.9$. (e) LNN-$\vee$ ($\alpha = 0.7$).

Base LNN Definitions

Conjunction:

$$\begin{aligned} \text{LNN-}\wedge(\mathbf{x};\beta,\mathbf{w}) \equiv \quad & \texttt{relu1}(\beta - \mathbf{w}^\top(1-\mathbf{x})) \\ \text{subject to:} \quad & \mathbf{w} \geq 0,\ \beta - \alpha\mathbf{w} \leq (1-\alpha)\mathbf{1} \\ & \beta - (1-\alpha)\mathbf{1}^\top\mathbf{w} \geq \alpha \end{aligned}$$

Disjunction:

$$\text{LNN-}\vee(\mathbf{x};\beta,\mathbf{w}) = 1 - \text{LNN-}\wedge(1-\mathbf{x};\beta,\mathbf{w})$$

Negation:

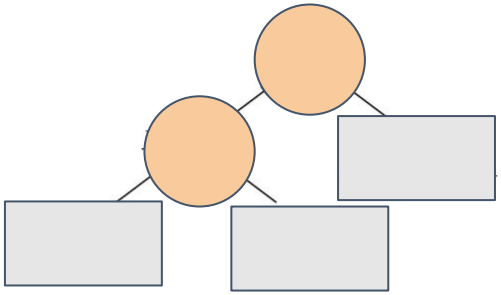
$$\text{LNN-}\neg(\mathbf{x};\beta,\mathbf{w}) = 1 - x$$

LNN Logic will be extended by gating over predicates

ILP-LNN Specific Details

Logical Template Graphs

Templates are graphs, defined by vertices ($\mathcal{V}$), edges ($\mathcal{E}$), and a mapping function ($\mathcal{L}$) :



$$\mathcal{T} = (\mathcal{V}, \mathcal{E}, \mathcal{L})$$

$\mathcal{L}$ determines which logical operators are used at each node, as well as which predicates are available.

Gradient descent then assigns weights to predicates and children to *fill in* the structured template.

Differentiable Gating: Predicates

How does ILP-LNN fill in a given template?

Weights can be learned over available predicates:

$$\sum_{P \in \mathcal{L}(V)} w_P \psi(P(x))$$

Where psi is the truth value of a predicate

Differentiable Gating: Sparsity

But ideally, this gating is *sparse*, so the authors propose:

$$1 - \texttt{relu}\left(\beta - \sum_{P \in \mathcal{L}(V)} w_P \psi(P(x))\right)$$

Which clips when weighted sum is smaller than bias

Sparsity would improve interpretability by encouraging zero-weights for irrelevant predicates..

Differentiable Gating: Disjunction

Applied ~verbatim to learn disjunction arguments:

$$1 - \texttt{relu}\left(\beta - \sum_{P \in \mathcal{N}(V)} w_P \psi(P(x))\right)$$

Where $\mathcal{N}(V)$ are the children of a disjunction node

Differentiable Gating: Conjunction

A similar formula is used for conjunctions

$$\texttt{relu}\left(\beta - \sum_{P \in \mathcal{N}(V)} w_P(1 - \psi(P(x)))\right)$$

Where $\mathcal{N}(V)$ are the children of a conjunction node

Differentiable Gating: Negation

For negation, consider a universe of assignments, $\mathcal{U}$

$$1 - \psi(P(x)), \forall x \in \mathcal{U}$$

Operator Constraints

Recall that LNN operators define inequality constraints:

$$\text{LNN-}\wedge(\mathbf{x};\beta,\mathbf{w}) \equiv \quad \texttt{relu1}(\beta - \mathbf{w}^\top(1-\mathbf{x}))$$

$$\text{subject to:} \quad \mathbf{w} \geq 0,\ \beta - \alpha\mathbf{w} \leq (1-\alpha)\mathbf{1}$$

$$\beta - (1-\alpha)\mathbf{1}^\top\mathbf{w} \geq \alpha$$

The authors propose to address these with the *double-description* method & Minkowski-Weyl theorem.

Essentially, a differentiable network layer can calculate weights and biases that follow these constraints.

Operator Constraints

Given inequality constraints of the form: $\boldsymbol{Az} \leq \mathbf{b}$ ere
A is a matrix of coefficients, and **b** constants..

Minkowski-Weyl states that there is a translation rays and Υ vertices : Γ

Theorem 1. *A set* $\mathcal{C} = \{\mathbf{z} \mid \boldsymbol{Az} \leq \mathbf{b}\}$ *is a convex polyhedron if and only if:*

$$\mathcal{C} = \left\{ \Upsilon\mu + \Gamma\lambda \,\middle|\, \mu, \lambda \geq \mathbf{0}, \mathbf{1}^\top \lambda = 1 \right\}$$

where Υ *and* Γ *contain a finite number of rays and vertices, respectively.*

Operator Constraints

Double-description can calculate rays and vertices: $\Upsilon\ \Gamma$

And the conditions can be satisfied by standard neural network operations like ReLU and softmax:

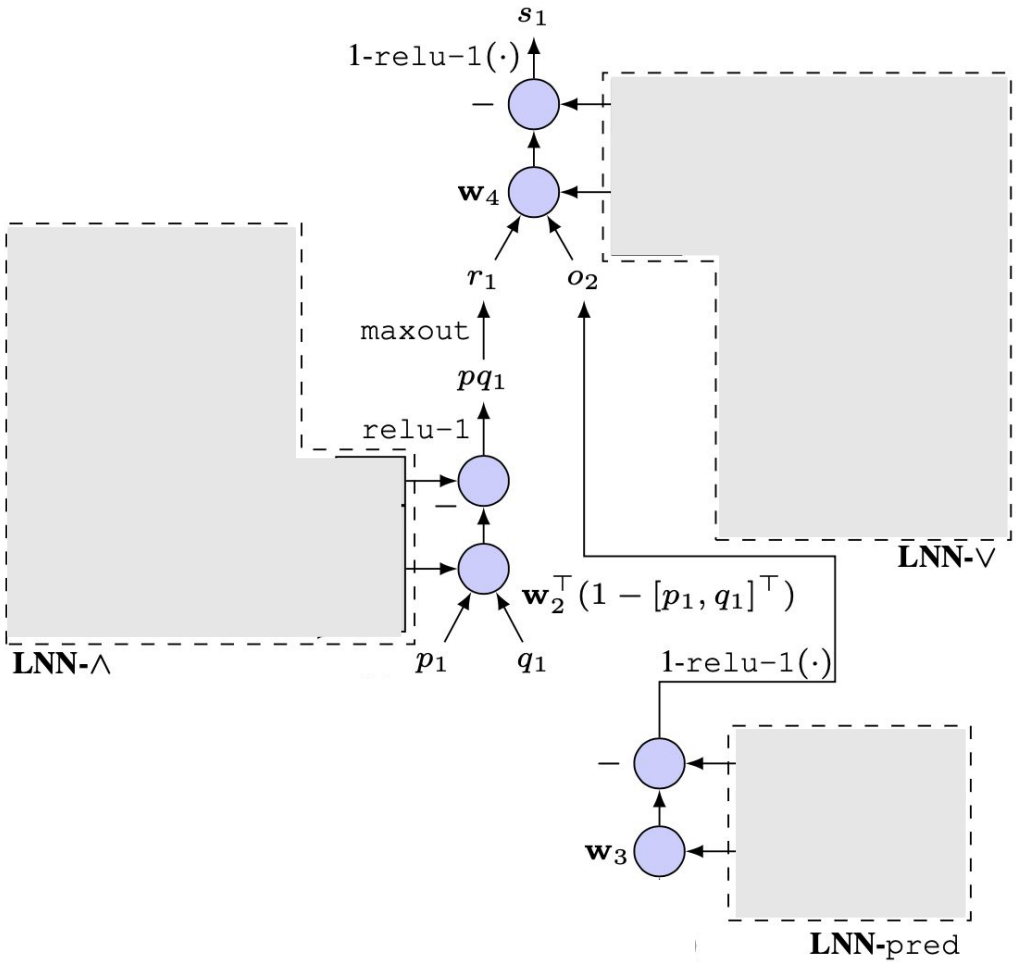
$$\mu = \max(0, \widehat{\mu}) \qquad \lambda = \frac{\exp(\widehat{\lambda})}{Z},\ Z = \mathbf{1}^\top \exp(\widehat{\lambda})$$

And finally, weights and biases are updated with:

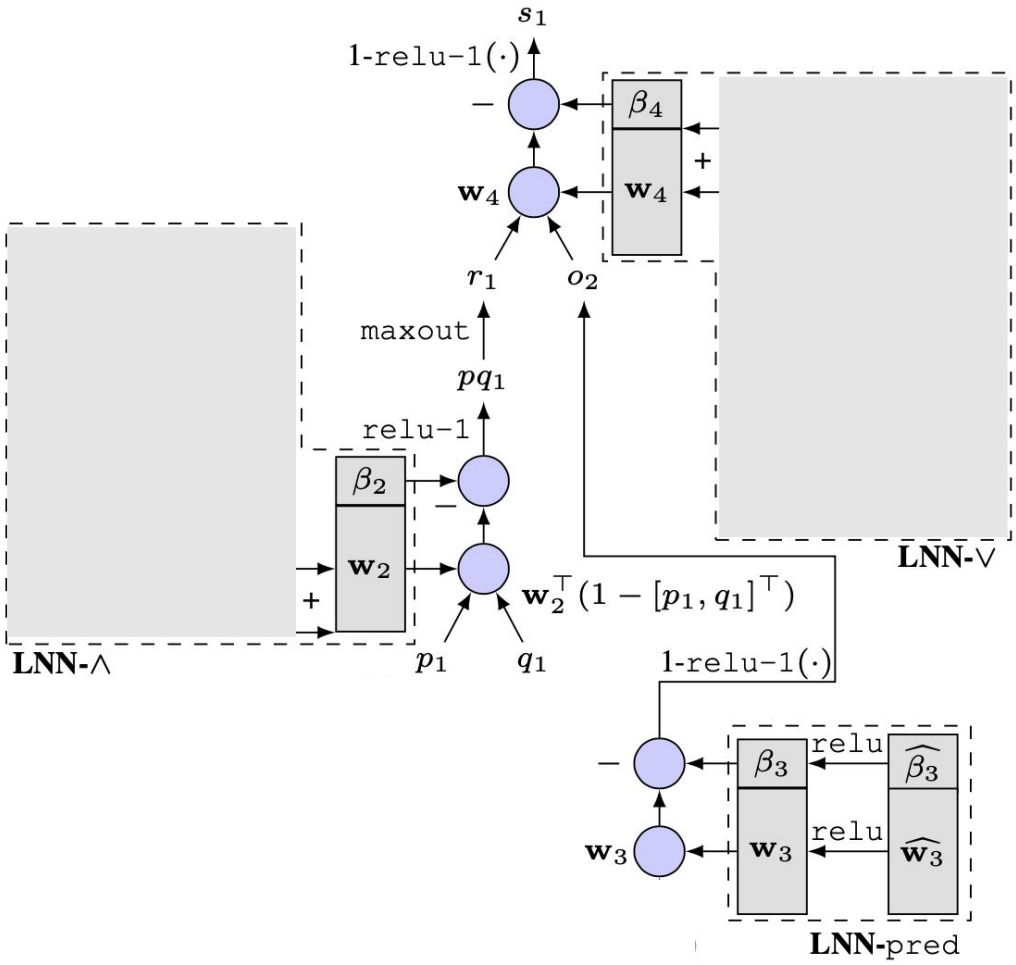
$$[\beta, \mathbf{w}^\top]^\top = \Upsilon\mu + \Gamma\lambda$$

So that LNN no longer implodes :)

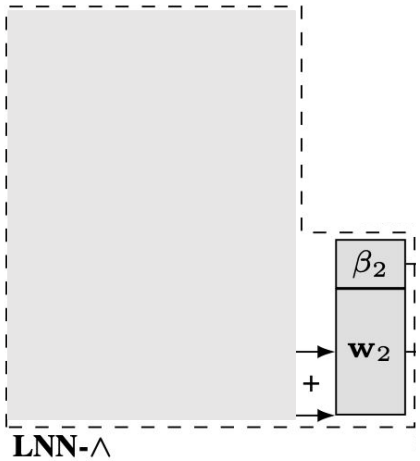
Final Architecture



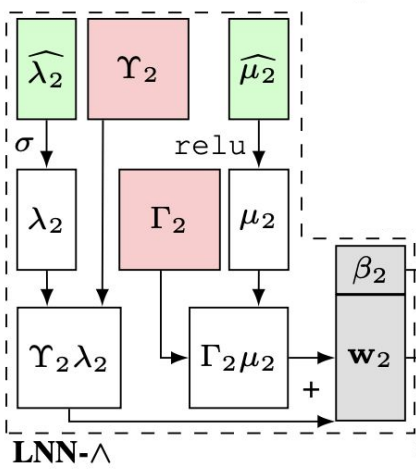
Final Architecture



Final Architecture



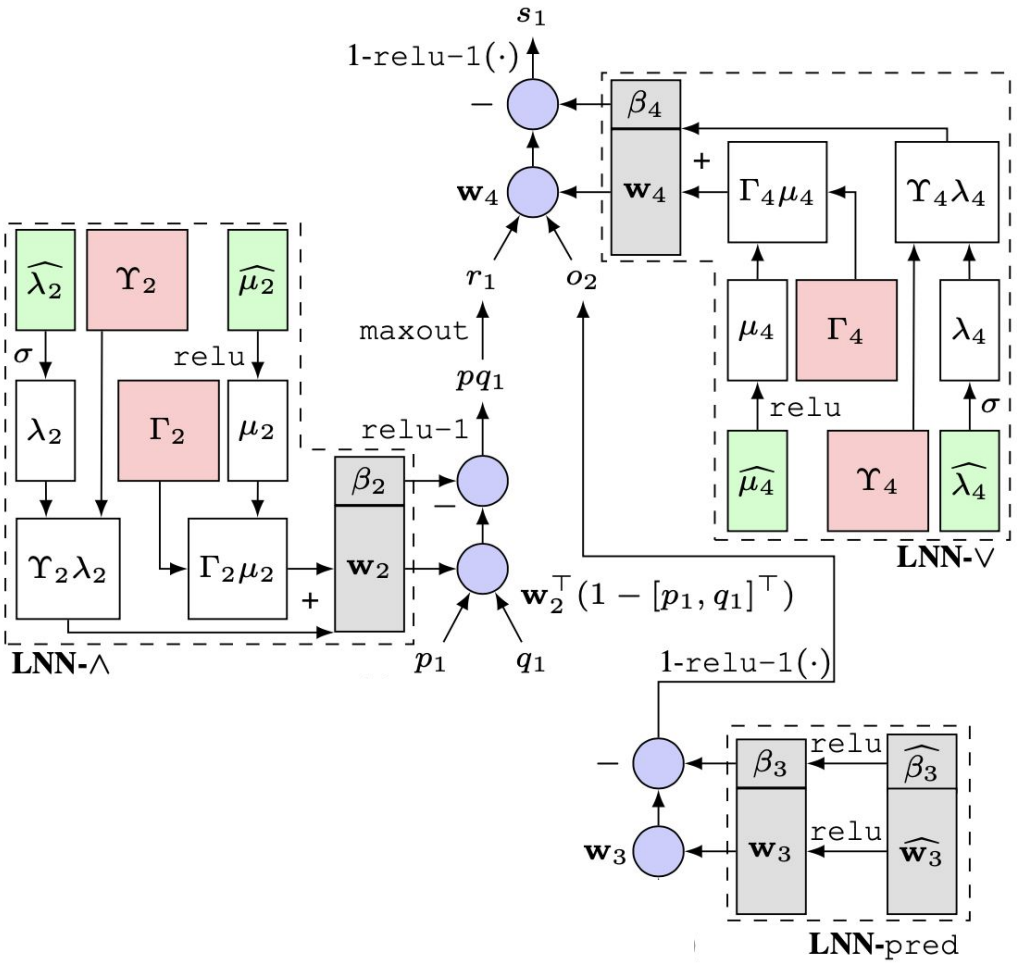
Final Architecture



Recall that rays Υ and vertices Γ and λ, μ are used to maintain constraints of weights and biases

Motivation:
βi, wi, need to satisfy the respective constraints associated with the LNN operators they form parameters for

Final Architecture - Constrained Learning



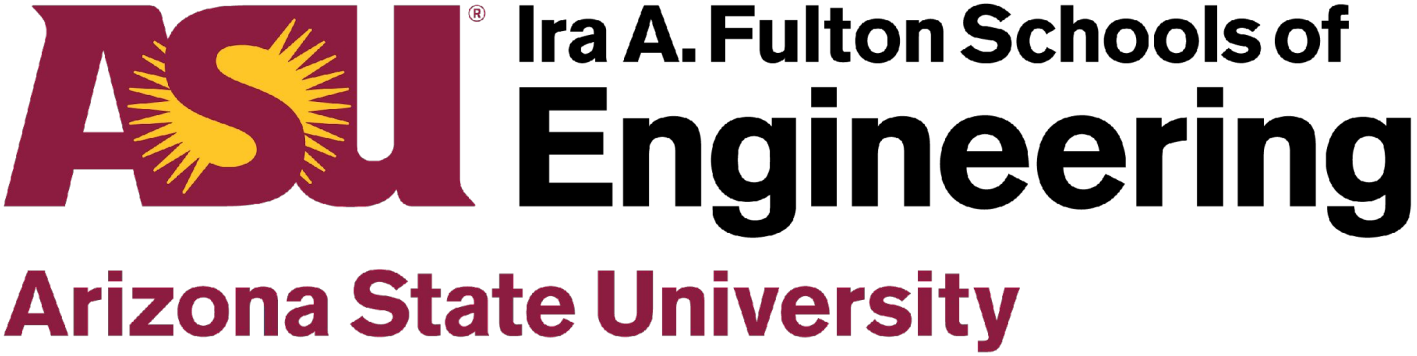
Applying the same for LNN-V for all parts of the network

Future work and limitations

- Humans specify templates (improve LNN)
  - Can templates be more flexible (pruning/insertion)?
  - Can templates be entirely learned instead?
- LNN-ILP's upward, forward pass is similar to a traditional logic program
  - What else can be done with the learned program?

References

- Sen, Prithviraj, et al. "Neuro-symbolic inductive logic programming with logical neural networks." *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. No. 8. 2022.
- Evans, Richard, and Edward Grefenstette. "Learning explanatory rules from noisy data." *Journal of Artificial Intelligence Research* 61 (2018): 1-64.
- Riegel, Ryan, et al. "Logical neural networks." *arXiv preprint arXiv:2006.*13155 (2020).


Ira A. Fulton Schools of
Engineering
Arizona State University