Teaching Neural Networks the Rules

A Selection of Solutions to Guarantee Compliance of NNs by Design

Lou Elah Süsslin

lou.suesslin@tuwien.ac.at

192.048 Seminar in Knowledge Representation and Reasoning: Neurosymbolic Artificial Intelligence
Vienna University of Technology (TU Wien)
Summer term 2025

January 19, 2026

Contents

Introduction

Neural Networks: Usefulness & Shortcomings

Usefulness

- Correlations in raw data $\rightarrow$ Accurate predictions
- Widespread success in various domains:
  - Facial Recognition
  - Financial Forecasting
  - Product Recommendation
  - Medical Sciences

Shortcomings

- Many domains require models satisfying proven requirements
- Common (Deep) NNs fail to comply with:
  - Pre-defined requirements
  - Inherent domain/problem knowledge
- Theoretical computer science and others want guarantees
- **Example:** 2D-ConvNet + 3D-RetinaNet on ROAD-R dataset
- High performance but 90% of predictions violate at least one requirement

How this has been addressed

Via Neuro-Symbolic AI methods (3 types):

- **Integrate requirements in loss function (penalize violations)**
  Advantage: reduces violation incidence, improves performance
  Disadvantage: cannot guarantee satisfaction
- **Post-Processing techniques (modify outputs at inference)**
  Advantage: enforce compliance in final output
  Disadvantage: fix-it later approach, with decay in performance
- **Integrate constraints in network topology**
  ("Guaranteed Compliance by Design", "Coherent-by-Construction")
  Advantage: guarantee compliance, no need for post-processing

Presentation Goal

- Showcase family of neuro-symbolic frameworks
- All share common goal: make existing NNs compliant by design to given requirements or sets thereof
- Leverage existing learning capabilities of neural networks

HMC problems and C-HMCNN(h)

Introduction

Multi-label classification (MC) problem: pair $(\mathcal{A}, \mathcal{X})$ where

- $\mathcal{A}$: finite set of classes/labels $\{A_1, A_2, \ldots\}$
- $\mathcal{X}$: finite dataset of labeled examples $\{(x, y)\}$
- $x \in \mathbb{R}^D$, label $y \subseteq \mathcal{A}$
- Model $m : \mathcal{A} \times \mathbb{R}^D \to [0, 1]$, outputs score $m_A(x)$
- Prediction: $m_A(x) \geq \theta \Rightarrow$ predict label A for x

Hierarchical Multi-label Classification (HMC): add constraints Π to MC problem P

- Π: finite set of hierarchy rules $A_1 \to A$ (subclass relation)
- Constraints induce an acyclic graph
- Model coherence: if $m_{A_1}(x) \geq \theta$, then $m_A(x) \geq \theta$
- Logical violation: subclass predicted but not parent
- Stronger hierarchy violation: $m_{A_1}(x) \leq m_A(x)$ must hold
- Note: logical violation $\Rightarrow$ hierarchy violation; converse is false

HMC: Real Example

- HMC appears in domains like image classification; e.g., annotating radiological images using IRMA codes.
- Example constraint: `stomach` $\rightarrow$ `gastrointestinalSystem`.
- Whenever `stomach` is predicted, `gastrointestinalSystem` must also be predicted.
- Logical violation example: predicting `stomach` but not `gastrointestinalSystem`.

C-HMCNN(h): Two-Step Architecture

Has a two-step architecture to solve HMC problems

1. Step: Two modules

- Bottom module h: neural network of choice
  Outputs: $h_{A_1}, h_A \in [0,1]$
- Upper module: Constraint Module (CM)
  - Input: h_{A_1}, h_A from h
  - Output:
  $$\text{CM}_{A_1} := h_{A_1}, \quad \text{CM}_A := \max(h_A, h_{A_1})$$
  - For any θ: if $\text{CM}_{A_1} \geq \theta$, then $\text{CM}_A \geq \theta$
  - Final prediction is given by CM output

C-HMCNN: Max Constraint Loss (CLoss)

2. Step: Novel loss function exploits hierarchy during training

- Goal: Use hierarchy constraints to guide h during training
- Approach: Modify standard BCE loss $\mathcal{L}$ into Max Constraint Loss (CLoss)
- Decomposition:

$$\mathsf{CLoss} = \mathsf{CLoss}_{A_1} + \mathsf{CLoss}_{A}, \quad \mathcal{L} = \mathcal{L}_{A_1} + \mathcal{L}_{A}$$

- Shared term:

$$\mathsf{CLoss}_{A_1} = -y_{A_1} \ln(\mathsf{CM}_{A_1}) - (1 - y_{A_1}) \ln(1 - \mathsf{CM}_{A_1})$$

- Differences in parent term:

$$\begin{aligned} \mathsf{CLoss}_{A} &= -y_{A} \ln\left(\max(\mathbf{h_A}, \mathbf{h_{A_1}} \cdot \mathbf{y_{A_1}})\right) \\ &\quad - (1 - y_{A}) \ln(1 - \mathsf{CM}_{A}) \\ \mathcal{L}_{A} &= -y_{A} \ln\left(\mathsf{CM}_{\mathbf{A}}\right) - (1 - y_{A}) \ln(1 - \mathsf{CM}_{A}) \end{aligned}$$

- **Why:** Ground truth y_{A_1} acts as switch for subclass influence
  - If $y_{A_1} = 1$: subclass present $\Rightarrow$ parent gets boosted
  - If $y_{A_1} = 0$: subclass absent $\Rightarrow$ fallback to h_A

C-HMCNN(h): Problem Setup & Baselines

Setup:

- Hierarchical multi-class classification in 2D input space
- Two rectangles $R_1 < R_2$ defining classes $A_1 = R_1$, $A = R_1 \cup R_2$
- Goal: learn nested and disjoint class structures effectively

Baseline approaches:

- **f-network + post-processing f^+:** Enforces subset relation ($\min / \max$ constraints), suited for $R_1 \subset R_2$
- **g-network + post-processing g^+:** Models A as union of disjoint A_1 and $A \setminus A_1$, suited for disjoint R_1, R_2

Training:

- Feedforward nets, BCE loss, Adam optimizer
- Synthetic uniform data, 20,000 epochs

C-HMCNN(h): Experimental Results

Scenario performance:

- **Subset scenario** ($R_1 \subset R_2$): C-HMCNN matches f^+, designed for hierarchical subset coherence
- **Disjoint scenario** ($R_1 \cap R_2 = \emptyset$): C-HMCNN matches g^+, designed for disjoint union of classes
- **Partial-overlap scenario** ($R_1 \cap R_2 \neq \emptyset, \neq R_1$): C-HMCNN outperforms both f^+ and g^+ post-processing methods

Additional observations:

- C-HMCNN learns to delegate points effectively between classes
- Produces stable predictions without explicit post-processing constraints

LCMC problems and CCN(h)

Logically Constrained Multi-Label Classification (LCMC)

LCMC extends multi-label classification by adding logical constraints Π expressed in *normal form*:

$$A_1, \ldots, A_k, \neg A_{k+1}, \ldots, \neg A_n \to A, \quad 0 \leq k \leq n$$

where n is the total number of literals and k the count of positive literals.

Example: "If an image contains `abdomen` but neither `middleAbdomen` nor `upperAbdomen`, then it must be `lowerAbdomen`":

$$\texttt{abdomen}, \neg\texttt{middleAbdomen}, \neg\texttt{upperAbdomen} \to \texttt{lowerAbdomen}$$

Key assumption: All class literals are distinct within each constraint to avoid redundancy.

Crucial observation: Normal form constraints **generalize hierarchical multi-class constraints** (HMC corresponds to the special case $n = k = 1$).

Implication: LCMC uses these constraints to extend HMC's expressiveness and logically consistent prediction notions.

Stratification of Constraint Sets Π

- Set of constraints Π may contain circular definitions, causing non-unique or non-minimal solutions.
- To avoid this, we stratify Π into $\Pi_1, \ldots, \Pi_s$.
- Stratification is a set of pairwise disjoint, non-empty subsets $\Pi_1, \ldots, \Pi_s$ of Π called strata, with
$$\Pi = \bigcup_{i=1}^{s} \Pi_i.$$
- Note Π_1 may be empty; thus stratification is not a strict partition.
- This ensures no circular definitions.
- Example: For
$$\Pi = \{A_1 \to A,\ A_2 \to A,\ A, \neg A_1 \to A_2\},$$
A_1 appears only in rule bodies, so
$$\Pi_1 = \emptyset.$$

A new model to tackle LCMC problems: Motivation

Example problem for a new model:

- MC problem with classes A, A_1, A_2
- Known constraints (expressed as normal form rules):
  1. $A_1 \to A$, 2. $A_2 \to A$, 3. $A \wedge \neg A_1 \to A_2$

These imply: $(A_1 \cup A_2) \subseteq A$

Therefore, for any model m and $x \in \mathbb{R}^D$:

$$m_A(x) \wedge \neg m_{A_1}(x) \implies m_{A_2}(x)$$

Goals for CCN:

- Leverage standard neural networks for MC
- Exploit all constraints above
- Guarantee predictions satisfy constraints
- Improve performance
- Extend HMC problems to LCMC problems

CCN(h) — Architecture

Solution proposal: CCN(h), consists of two modules + a custom loss (like C-HMCNN(h))

- **Bottom module h:** Any neural network, outputs one score per class:

$$h_A, \quad h_{A_1}, \quad h_{A_2}$$

- **Upper constraint module (CM):**
  - Inputs: h_A, h_{A_1}, h_{A_2}
  - Imposes constraints from normal form rules
  - Outputs constrained predictions: CM_A, CM_{A_1}, CM_{A_2}
- CM enforces ($A \wedge \neg A_1 \to A_2$):

$$\begin{aligned} CM_A &= \quad \max(h_A, CM_{A_1}, CM_{A_2}) \\ CM_{A_1} &= \quad h_{A_1} \\ CM_{A_2} &= \quad \max\left(h_{A_2}, \min(CM_A, \overline{CM}_{A_1})\right) \end{aligned}$$

Custom Loss Function $CLoss$ vs Standard BCE

Class	Variant	Loss Term
A	CLoss	$-y_A \ln \max(h_A, h_{A_1} y_{A_1}, h_{A_2} y_{A_2}) - \bar{y}_A \ln \overline{CM}_A$
	BCE	$-y_A \ln CM_A - \bar{y}_A \ln \overline{CM}_A$
A_1	CLoss	$-y_{A_1} \ln CM_{A_1} - \bar{y}_{A_1} \ln \overline{CM}_{A_1}$
	BCE	Same as CLoss
A_2	CLoss	$-y_{A_2} \ln \left(\max \left(h_{A_2}, \min(h_A y_A, h_{A_1} y_{A_1}), h_{A_1} y_{A_1} \right) \right)$ $-\bar{y}_{A_2} \ln \left(1 - \max \left(h_{A_2}, \min(h_A y_A + y_A, h_{A_1} y_{A_1} + y_{A_1}), h_{A_1} y_{A_1} + y_{A_1} \right) \right)$
	BCE	$-y_{A_2} \ln CM_{A_2} - \bar{y}_{A_2} \ln \overline{CM}_{A_2}$

Key points:

- $CLoss$ uses **ground truth as switches** to guide conditional prediction delegation
- Ensures training gradients point in the **correct direction**, unlike standard BCE
- Helps model learn to exploit logical constraints for better performance and consistency

CCN(h): Problem Setup & Baselines

Setup:

- Multi-label classification on 2D inputs, classes defined by regions R_1, R_2
- Goal: learn consistent label assignments under constraints (hierarchy or disjointness)
- Three scenarios:
  - **Disjoint:** $R_1 \cap R_2 = \emptyset$
  - **Subset:** $R_1 \subset R_2$
  - **Partial overlap:** $R_1 \cap R_2 \neq \emptyset$

Baselines:

- f: standard feedforward NN, trained with BCE loss
- f^+: f + post-processing enforcing constraints (max/min operations)
- CCN(h): coherent-by-construction model integrating constraints in training (CLoss)

Training:

- Simple NN architecture (1 hidden layer, 4 neurons, `tanh`)
- Adam optimizer, 20,000 epochs, uniform data sampling
- Labels structured to encode logical relations $A_1 \to A$

CCN(h): Results & Insights (1/2)

Standard Neural Network (f):

- **Struggles with decision boundaries**: f struggled to learn class boundaries for A and A_2 in the synthetic experiment.
- **Performance variability**: Strongly dependent on the distribution and relative arrangement of data points (e.g., disjoint vs. overlapping R_1, R_2).
- **Requires post-processing**: Outputs often needed correction to enforce hierarchy or logical constraints.

Post-processed Network (f^+):

- **Performance decay**: Post-processing enforces constraints but can degrade performance.
- **Scenario-dependent accuracy**: In the disjoint case, f^+ wrongly predicts points in R_1 as A_2 due to constraint logic.
- **Less stable than CCN(h)**: Exhibits much higher **standard deviations** in evaluation metrics across scenarios.

CCN(h): Results & Insights (2/2)

Coherent-by-Construction Network (CCN(h)):

- **Integrates constraints directly** into training via a Constraint Module and Loss.
- **Adapts intelligently** to different class relationships.
- **Learns only necessary outputs**, deriving others via constraints.
- Ensures **coherent predictions by design** — no post-processing needed.
- Provides **stable and consistent** performance across scenarios.

Conclusion: Embedding **hard logical constraints** into training yields **more robust, coherent, and interpretable** multi-label models compared to relying on post-processing corrections.

Integral Part of CCN(h): Procedure CompStrata(Π)

Input: Stratified set of constraints Π (Multiple can exist, logically equal)

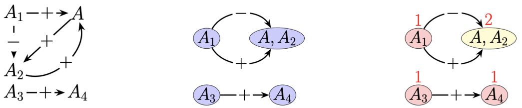
Job 1: Resolve circular definitions through a dependency graph, with:

- Nodes $=$ classes
- For each constraint $r \in \Pi$:
  - Positive edges from classes in $body^{+}(r)$ to $head(r)$
  - Negative edges from class A with $\neg A \in body^{-}(r)$ to $head(r)$
- Π is stratified iff the graph contains no cycles with a negative edge

Job 2: Compute stratification $\Pi_1, \ldots, \Pi_s$ and class partitions $\mathcal{A}_1, \ldots, \mathcal{A}_s$ with smallest n of strata, where:

- $\mathcal{A}_i$ pairwise disjoint, non-empty subsets, $\bigcup_i \mathcal{A}_i = \mathcal{A}$

Example:



(a) G_Π (b) DAGs from step 1 (c) Numbers from step 2

Figure 5: Given Π as in Example 3.14, visual representation of (a) G_Π, (b) the acyclic component graph of G_Π, (c) the number assigned to each class: 1 to A_1, A_3, A_4, and 2 to A, A_2.

Propositional Logic and CCN$^+$

CCN^+: Coherent-by-construction network$^+$

- Extends CCN to multi-class (MC) problems
- Moves from set of constraints Π as normal rules (CCN)
- To set of **requirements** Π as propositional logic formulas ("set of clauses") over labels $\mathcal{A}$

CCN$^+$: Requirements as Clauses

- Clause example: $l_1 \vee l_2 \vee \ldots \vee l_n$
- Each l_i is a **literal** (label $A \in \mathcal{A}$ or negation $\neg A$)
- Clause means: at least one literal true, model must predict at least one
- Assumption: each label appears once only (pos. or neg.)
- Example: $A_1 \vee A_2$
  - At least one label predicted positive
  - Equivalent to: $\neg A_1 \rightarrow A_2$ and $\neg A_2 \rightarrow A_1$
- General rule for literal l_n: $l_1 \wedge \ldots \wedge \neg l_{n-1} \rightarrow l_n$
- Rules help compute label predictions bottom-up

CCN$^+$: Architecture

- Input: MC problem with requirements $(\mathcal{P}, \Pi)$
- Standard neural network h produces initial output $h(x)$, possibly incoherent
- Add requirement layer **ReqL** on top of h
  - Enforces coherence with Π
  - Produces output $ReqL(x)$ using rules from Π
- Add requirements loss **ReqL loss** for training CCN$^+$
  - Teaches h to respect constraints and improve predictions

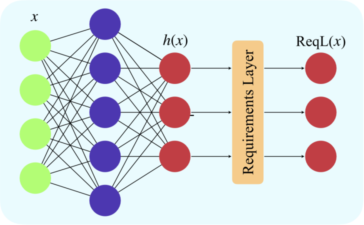


Fig. 1. Basic intuition behind CCN$^+$.

Basic Case Setup

- MC problem $\mathcal{P}$ with two labels: $\mathcal{A} = \{A_1, A_2\}$
- Requirements: $\Pi = \{\neg A_1 \vee A_2\}$
- Reformulations:
    1. $A_1 \to A_2$
    2. $\neg A_2 \to \neg A_1$
- **Important:** Cannot choose both reformulations simultaneously

Reformulation 1: $A_1 \to A_2$

- This is a normal rule: compute A_2 on grounds of A_1
- Requirement layer:

$$ReqL_{A_1} = CM_{A_1} = h_{A_1}$$

$$ReqL_{A_2} = CM_{A_2} = \max(h_{A_2}, h_{A_1})$$

- Training loss uses standard BCE similar to C-HMCNN(h):

$$\begin{cases} ReqLoss_{A_1} = -y_{A_1} \log ReqL_{A_1} - (1 - y_{A_1}) \log(1 - ReqL_{A_1}) \\ ReqLoss_{A_2} = -y_{A_2} \log \max(h_{A_2}, h_{A_1} y_{A_1}) - (1 - y_{A_2}) \log(1 - ReqL_{A_2}) \end{cases}$$

Reformulation 2: $\neg A_2 \rightarrow \neg A_1$

- Need negation function $f_\neg$ with strict negation property:

$$f_\neg(v) = 1 - v, \quad v \in [0,1]$$

$$\text{and for } \theta \in (0,1): \; f_\neg(v) > \theta \Leftrightarrow v < \theta$$

  - Ensures that for any label A, only one of m_A or $m_{\neg A}$ exceeds the threshold θ
  - Implies that model m predicts either A or $\neg A$, and neither equals θ
  - From now: **standard negation**: $f_\neg(v) = 1 - v$ for all $v \in [0,1]$, entailing $\theta = 0.5$
- Now compute ReqL (for $\neg A_2 \rightarrow \neg A_1$):

$$ReqL_{A_1} = 1 - \max(1 - h_{A_1}, 1 - h_{A_2}) = \min(h_{A_1}, h_{A_2})$$

$$ReqL_{A_2} = h_{A_2}$$

- Training loss updates accordingly (inserting ReqL):

$$\begin{cases} ReqLoss_{A_1} = -y_{A_1} \log ReqL_{A_1} - (1 - y_{A_1}) \log(1 - \min(h_{A_1}, h_{A_2}(1 - y_{A_2}))) \\ ReqLoss_{A_2} = -y_{A_2} \log ReqL_{A_2} - (1 - y_{A_2}) \log(1 - ReqL_{A_2}) \end{cases}$$

Important Conclusion

- Both reformulations represent the same clause logically
- **However, choosing both simultaneously is impossible**
- The direction of dependency impacts:
  - Model expressiveness
  - Training feasibility
  - Allowed constraints (hierarchical vs. propositional)
- Next: Visualize differences between the two reformulations

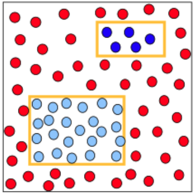
Visualizing Differences – Setup

Datasets

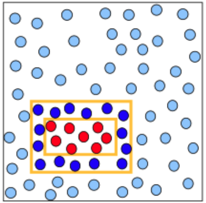
- $\mathcal{X}_1, \mathcal{X}_2 \subset \mathbb{R}^2$
- Each point has labels from $\mathcal{A} = \{A_1, A_2\}$
- Color code:
  - Blue: A_1, A_2
  - Light blue: A_2 only
  - Red: none

Models

- Some standard feedforward NN f
  - 1 hidden layer (4 tanh), sigmoid output
  - Loss: Std. BCE
- p-CCN$^+$: clause as $A_1 \to A_2$
- n-CCN$^+$: clause as $\neg A_2 \to \neg A_1$
- All models trained 20k steps with Adam, lr $= 10^{-2}$



(a) $\mathcal{X}_1$

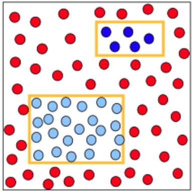


(b) $\mathcal{X}_2$

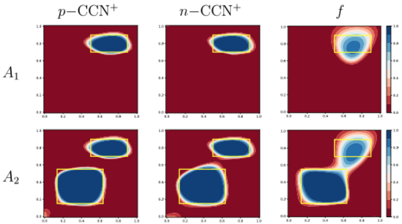
Procedure

- Train each model on $\mathcal{X}_1$ and $\mathcal{X}_2$
- Compare performance with and without Requirement Layer
- Observe how different clause formulations affect decision boundaries

Results on Dataset $\mathcal{X}_1$



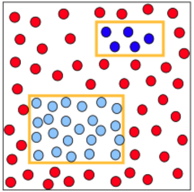
(a) $\mathcal{X}_1$



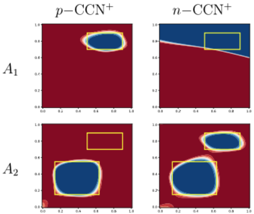
- p-CCN$^+$ and n-CCN$^+$ outperform standard network f
- Coherent decision boundaries reflect the incorporated background knowledge
- p-CCN$^+$ yields slightly better fit for class A_2

(Color code: Blue = A_1, A_2; Light blue = A_2 only; Red = none)

Understanding $\mathcal{X}_1$ – Base Networks



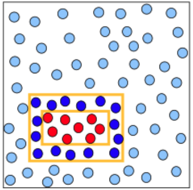
(a) $\mathcal{X}_1$



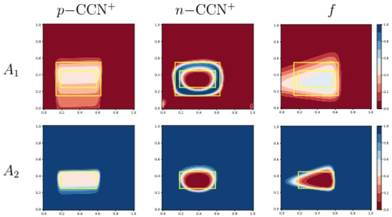
- Key findings:
  - p-CCN$^+$: A_2 output fades in smaller rectangle, relying on A_1 via ReqL
  - n-CCN$^+$: simpler boundary for A_1, needs ReqL to become coherent

(Color code: Blue = A_1, A_2; Light blue = A_2 only; Red = none)

Results on Dataset $\mathcal{X}_2$



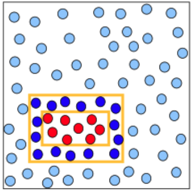
(b) $\mathcal{X}_2$



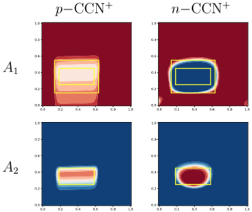
- Observations:
  - n-CCN$^+$ $>$ all others
  - Only n-CCN$^+$ captures the "donut" (hollow) shape
  - p-CCN$^+$ and f both fail to separate inner region

(Color code: Blue $= A_1, A_2$; Light blue $= A_2$ only; Red $=$ none)

Understanding $\mathcal{X}_2$ – Base Networks



(b) $\mathcal{X}_2$



- Key findings:
  - Only n-CCN$^+$ learns to suppress A_1 inside A_2's area
  - When trained without ReqL, n-CCN$^+$ loses this capability

(Color code: Blue = A_1, A_2; Light blue = A_2 only; Red = none)

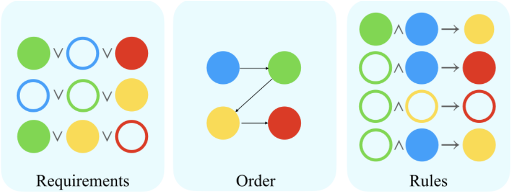
Model Performance

Accuracy over five runs with different dataset seeds

Model	$\mathcal{X}_1$ Accuracy	$\mathcal{X}_2$ Accuracy
p-CCN$^+$	0.984 ± 0.006	0.924 ± 0.012
n-CCN$^+$	0.963 ± 0.015	0.966 ± 0.004
Standard NN f	0.941 ± 0.004	0.915 ± 0.010

- p-CCN$^+$ performs best on $\mathcal{X}_1$
- n-CCN$^+$ performs best on $\mathcal{X}_2$
- Both CCN variants outperform standard NN f

Label Ordering in Requirement Layer



- **Color** = label, **Outline** = negated literal
- Ordering is assumed in the figure (not inferred)
- Rule head = literal with the **highest-level label**
- Example: $\neg$Blue $\vee$ $\neg$Green $\vee$ Yellow $\Rightarrow$ Green $\wedge$ Blue $\rightarrow$ Yellow

- **Rule ordering:** Turning requirements in propositional logic into a set of rules impacts model performance
- **CCN$^+$ formalizes this process with a function:** establishes an ordered sequence of the labels for their computation within the requirements layer
- **Level assignment (Integer to each label, not literal):**
  - Build DAG: labels = nodes; edge from rule head to body labels
  - Level 0: no incoming edges; others = longest path from any level 0 label

Result: Set of operational rules (See example for one rule)

Summary

Summarizing Coherent-by-Design Models

Findings: Where they excel:

- **Coherent-by-construction** with hard logical constraints
- Can be expressed in full **propositional logic** or stratified normal rules
- **Superior performance** by embedding domain knowledge
- **Reliably** outperforms state-of-the-art models
- **Versatility**: Use with any neural network architecture
- **Diverse applicability**: Genomics, tabular data generation, multi-label image classification
- **GPU optimization** possible (e.g., CCN+), not covered here

Coherent-by-Design Future Research

- **Label order in propositional logic** $\rightarrow$ Influences model performance
  - Systematically determine best order
- Support other **numerical relations** than linear inequalities
- Combine **soft and hard constraints**
- Add to **Explainable AI** $\rightarrow$ Leverage reasoning capabilities
  - Generate more natural explanations for network predictions

Thank you for following!

References

- Giunchiglia, E., & Lukasiewicz, T. (2021). Multi-label classification neural networks with hard logical constraints. *Journal of Artificial Intelligence Research, 72*, 759–818.
- Stoian, M. C., Tatomir, A., Lukasiewicz, T., & Giunchiglia, E. (2024). PiShield: A PyTorch Package for Learning with Requirements. *arXiv preprint arXiv:2402.18285*.
- Giunchiglia, E., Tatomir, A., Stoian, M. C., & Lukasiewicz, T. (2024). CCN+: A neuro-symbolic framework for deep learning with requirements. *International Journal of Approximate Reasoning, 171*, 109124.
- Fiaschi, L., & Cococcioni, M. (2024). Informed deep hierarchical classification: a non-standard analysis inspired approach. *arXiv preprint arXiv:2409.16956*.
- Giunchiglia, E., Stoian, M. C., & Lukasiewicz, T. (2022). Deep learning with logical constraints. *arXiv preprint arXiv:2205.00523*.

Thank you for following! Questions?

(Extra: PiShield)

PiShield: Introduction

- Python-based ML library, extends **PyTorch**
- Focus: **Requirements-driven ML**
- Goal: Help DL models meet **safety requirements** for outputs
- How: Integrate domain requirements into NN topology to ensure **compliance regardless of input**
- Adds new PyTorch layers called **Shield Layers** on top of any NN (next slide)

Implementing a Shield Layer

Training Time: Integrate after the layer computing model output

Inference Time:

- Integrate outside NN; apply to outputs to enforce compliance
- Needs:
  - Input dimension (usually network output size)
  - Path to file with requirements

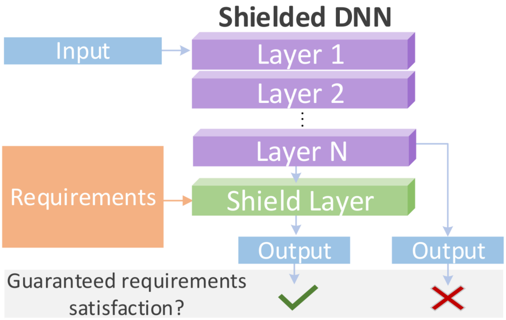


Figure 1: PiShield overview.

Requirements Format

- Each requirement on a separate text line
- Expressed as either:
  - **Propositional logic formula**: in CNF, i.e., single OR-clauses combined by ANDs
  - **Linear inequalities**

Logic Formula Example

Given: Images of traffic lights from within a car
Question: Is there a traffic light? What color?
Setup: Multi-class problem, 4 labels

- y_0 — presence of traffic light
- y_1 — red light
- y_2 — yellow light
- y_3 — green light

Each image → 4-element output vector, e.g. `[1,0,0,1]`
Requirements:

- $y_0 \rightarrow (y_1 \vee y_2 \vee y_3)$
- No two colors can be TRUE simultaneously

```
not y_0 or y_1 or y_2 or y_3
not y_0 or not y_1 or not y_2
not y_0 or not y_1 or not y_3
not y_0 or not y_2 or not y_3
```

Linear Inequalities Example

Given: Synthetic tabular data for clinical trial
Goal: Enforce domain knowledge, e.g.,

- `MaxHemoglobin` $\geq$ `MinHemoglobin`
- `MaxTemp` $\geq$ `MinTemp`

Expressed as linear inequalities to incorporate constraints into learning

```
y_0 - y_1 >= 0
y_2 - y_3 >= 0
```

Real-World Examples Overview

Domain	Scenario	Task type	Usage of Constraints
Genomics/ Bioinformatics	Functional Genomics	HCMC problem	**Propositional logic reqs./CNF**: Respect biological hierarchies when predicting gene functions: If a gene has specific function, it must have broader related functions (consistent with hierarchy)
Autonomous driving	Road Event Detection	MC problem	**Propositional logic reqs./CNF** (n=243): Guarantee detected road events obey safety rules
Structured data modeling	Tabular Data Generation	Deep generative modeling	**Linear inequalities**: Ensure generated synthetic data respects real-world relationships between features (e.g. never min. value > max. value). Helps making data realistic

Real-World Examples Performance

Advantages:

- **Scenario 1:** Preserving hierarchical structure
- **Scenario 2:** Guarantee satisfaction of requirements + Performance boost
- **Scenario 3:** Compliance with background knowledge + Realistic data generation

Scenario	Baseline	PiShield
Functional genomics(AU($\overline{\text{PRC}}$))	0.225	**0.241**
Autonomous driving (f-mAP)	0.288	**0.303**
Tabular data generation (Utility-F1)	0.430	**0.458**

Table 1: Aggregated performance. The best results are in **bold**.

(Extra: Hierarchy graph problems and LH-DNN)

Motivation

Neural network models should solve the **hierarchy graph problem** $HC = \langle T, S, F \rangle$:

- Graph $T =$ Tree structure organizing classes/labels
- Each data point associated with a single path S through the hierarchy
- Classification path must extend fully to the deepest possible tree level F

T organizes labels in hierarchical categories:

- Level $1 =$ Upper category (e.g., *clothing*)
- Level $2 =$ Subcategory (e.g., *bottom*)
- Level $3 =$ Subsubcategory (e.g., *capri pants*)

Reformulating the Problem

Reframe the HC problem in two steps:

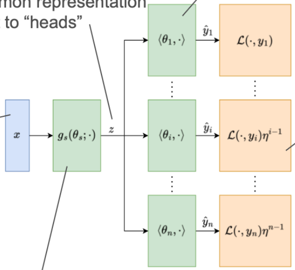
1. **Lexicographic Multi-Objective Optimization Problem (LMOP):**
   - LMOP $=$ Standard optimization with multiple strictly ordered objectives
   - Objectives $=$ Correct prediction ($\min$ loss) at each hierarchy level
   - Strict priority: Higher levels optimized before deeper levels
2. **Theorem from Non-Standard Analysis:**
   - Reformulate LMOP as a single *non-standard scalar program*
   - Single objective for NN: Minimize weighted sum of per-level losses, weights expressed as powers of an infinitesimal ϵ
   - Enforces strict lexicographic priority by assigning ∞-greater importance to higher-level errors

LH-DNN Model

Per-level classification heads
→ indep. Learning flows
→ specialized sub-network for
"per-layer categorization" (focused on level)
own set of params (\theta_i)

Hidden layer
→ Common representation
→ Input to "heads"

Raw
input



Per level non-standard loss function (Core novelty!)
→ Calculate individual loss/error for each classification head
→ based on: Prediction at hierarchy level and ground truth y_i

Common/initial NN
→ Feature learning
→ Extracts features from across all levels (shared weights)
→ e.g. has fur/wings (distinguish mammal and bird

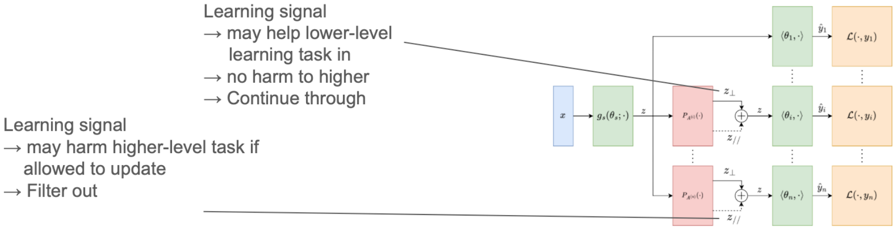
Finally:
→ Individual weighted losses summed
→ Form single objective function (error to be minimized) for the entire network.

Standard DL techniques (e.g. backpropagation)
→ Try minimize this one big sum
→ Will first prioritize higher levels improving higher levels due to weights

Gradient Projection for Lexicographic Priority

What if: Derive gradient from non-standard loss?

- Problem: Updating standard NN parameters naively is not possible
- Solution: **Projector operator** P_A modifies common representation z to ensure lexicographic priorities
- Goal: Update shared params θ_s so that learning on lower-level tasks *does not degrade* performance on higher-level tasks



Performance

Tested against B-CNN:

- Convolutional neural network tailored for **hierarchical classification tasks**
- Benchmarks: Fashion catalog; small everyday objects (100, 1000 categories)

Results:

- Faster learning, even at **half the parameters**
- More **stable learning**
  - e.g., no accuracy drop across all levels when optimizing lower levels
- No accuracy compromise on **higher levels** when improving lower levels
  - B-CNN exhibits compromise

Downside:

- Projection blocks require **recomputation each batch during training → time overhead**
- Calculation depends on parameters from previous levels (constantly changing)