Deep Learning with Hard Logical Constraints

Eleonora Giunchiglia

Keble College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Hilary 2022

To my family
Alla mia famiglia

Acknowledgements

First of all, I would like to thank my supervisor, Thomas Lukasiewicz, for his continuous guidance and support throughout my DPhil. His continuous belief in me and my research made me persevere even in those times where nothing seemed to work.

I would also like to thank all the members of the Intelligent System group, for the many inspiring research conversations and the many more entertaining ones.

A very special thanks goes to my family, who has always encouraged me and has always been my strength and my rock. Another special thanks goes to Yashovardhan, who has been the best companion in adventure I could have asked for.

Last but not least, I would like to thank all the amazing people that I have met in throughout my Oxford journey, because they have made it absolutely magical, and all my long-time friends, who made me who I am.

Abstract

Deep learning is becoming increasingly ubiquitous and thanks to its successes, it is likely to be applied in almost every aspect of our lives in the next few years. Its success stories however overshadow the dangers that come with its careless application in the real world. Indeed, even if deep learning models report astonishingly high-performance in terms of accuracy (or any other chosen metric), they do not give any guarantees that the model will not have any unintended behaviour when used in practice. This is particularly dangerous in safety-critical applications, where even a single unforeseen mistake can have severe consequences. Further, with each wrong move, human confidence in this technology falters, slowing down its adoption. Thus, it is extremely important to improve the trustworthiness of these models by reducing, if not completely ruling out, all unintended behaviors.

In this thesis, I address the problem of how to build deep learning based models able to (i) guarantee the satisfaction of a given set of requirements, which state the correct behaviour of the model, and (ii) learn from the background knowledge specified in the requirements themselves to improve performance. In particular, I focus on (i) deep learning models for multi-label classification problems, and (ii) requirements modelled as hard logical constraints. In order to achieve such a goal, I started by considering multi-label classification problems with hierarchical constraints, and then incrementally increased the expressivity of the constraints.

In the first phase of the project, I focused on hierarchical multi-label classification problems, which are multi-label classification problems with hierarchical constraints over the output space of the form $A_1 \rightarrow A$ expressing that A_1 is a subclass of A. For such problems, I developed a novel model, C-HMCNN(h), which, given a network h for the underlying multi-label classification problem, exploits the hierarchy information to

produce predictions guaranteed to satisfy the hierarchy constraints and to improve over h's performance.

In the second phase of the project, I considered constraints expressed as normal logic rules, i.e., expressions of the form $A_1, ..., A_k, \neg A_{k+1}, ..., \neg A_n \rightarrow A$. This expression imposes that whenever the classes $A_1, ..., A_k$ are predicted, while $A_{k+1}, ..., A_n$ are not, then the class A should be predicted. For this problem I developed CCN(h), which is an extension of C-HMCNN(h). This model, given a network h for the underlying multi-label classification problem, is able to (i) produce predictions guaranteed to satisfy the constraints, and (ii) exploit the information contained in the constraints to improve performance.

Finally, in order to demonstrate the significance of the problem tackled in this thesis, I created the ROad event Awareness Dataset with logical Requirements (ROAD-R), the first publicly available dataset for autonomous driving with requirements modelled as constraints over the output space and expressed as propositional logic formulas. By virtue of creating ROAD-R, I was able to show that the current state-of-the-art models do not learn the requirements from just the data points. My experimental results indicate that more than 90% of their predictions violate the constraints, and that it is possible to exploit the given requirements to create models that (i) have a better performance, and (ii) are guaranteed to be compliant with the given requirements.

Publications

This thesis is based on the following publications:

1. Eleonora Giunchiglia and Thomas Lukasiewicz. Coherent hierarchical multi-label classification networks. In *Proceedings of NeurIPS*, 2020.

2. Eleonora Giunchiglia and Thomas Lukasiewicz. Multi-label classification neural networks with hard logical constraints. *Journal of Artificial Intelligence Research*, 72, 2021.

3. Eleonora Giunchiglia, Mihaela Catalina Stoian, Salman Khan, Fabio Cuzzolin, and Thomas Lukasiewicz. ROAD-R: The autonomous driving dataset for learning with requirements. *Artificial Intelligence for Autonomous Driving Workshop at IJCAI (Best paper award)*, 2022.

4. Eleonora Giunchiglia, Mihaela Catalina Stoian, and Thomas Lukasiewicz. Deep learning with logical constraints. In *Proceedings of IJCAI*, 2022.

During my DPhil, I also co-authored the following publications, which are not incorporated in this thesis:

1. Eleonora Giunchiglia, Michele Colledanchise, Lorenzo Natale, and Armando Tacchella. Conditional behavior trees: Definition, executability, and applications. In *Proceedings of IEEE SMC*, 2019.

2. Oana-Maria Camburu[1], Eleonora Giunchiglia[1], Jakob Foerster, Thomas Lukasiewicz and Phil Blunsom. Can I Trust the Explainer? Verifying Post-hoc Explanatory Methods. In *Workshop on Safety and Robustness in Decision Making at NeurIPS*, 2019.

3. Louis Mahon, Eleonora Giunchiglia, Bowen Li and Thomas Lukasiewicz. Knowledge Graph Extraction from Videos. In *Proceedings of IEEE ICMLA*, 2020.

[1]Equal contribution.

4. Oana-Maria Camburu, Eleonora Giunchiglia, Jakob Foerster, Thomas Lukasiewicz and Phil Blunsom. The struggles of feature-based explanations: Shapley values vs. minimal sufficient subsets. In *AAAI Explainable Agency in Artificial Intelligence Workshop*, 2021.

5. Sai Vidyaranya Nuthalapati, Ramraj Chandradevan, Eleonora Giunchiglia, Bowen Li, Maxime Kayser, Thomas Lukasiewicz and Carl Yang. Lightweight Visual Question Answering using Scene Graphs. In *Proceedings of ACM CIKM*, 2021.

Further, during my DPhil, I had the pleasure to co-supervise two MSc theses:

1. *Visual Question Answering Using Scene Graphs.* Thesis written by Sai Vidyaranya Nuthalapati for the degree of Master of Science, University of Oxford, 2019.

2. *Logical Video Annotation with Background Knowledge.* Thesis written by Louis Mahon for the degree of Master of Science, University of Oxford, 2019.

Contents

List of Figures

Chapter 1

Introduction

Deep Learning (DL) is becoming increasingly ubiquitous and it is likely to be applied in almost every aspect of our lives in the next few years. This is happening thanks to its successes, which have enabled solutions to problems that were previously thought to be unsolvable. Astonishing examples of such trend are given by: (i) AlphaFold [104], a DL model that solved the "protein folding problem", a grand challenge in the field of biology for more than half a century, (ii) InceptionNet(v3) [117], an established DL model that has been shown, in [22], to be able to not only diagnose lung cancer with the same accuracy of a human pathologist, but also to be able to identify the genetic mutations of a given tissue sample—a task that even trained doctors cannot accomplish—and (iii) halicin [116], the first antibiotic discovered using DL, which could help in the battle against bacterial resistance. The above results, though groundbreaking, overshadow the dangers that come with the careless application of DL models in the real world. Indeed, these models report astonishingly high-performance in terms of accuracy (or alternatively chosen metric), however, they do not give any guarantee that the model will not have any unintended behaviour when used in practice. As pointed out by Rudin in [100], there have already been cases of people erroneously denied parole [133], DL-based pollution models stating that highly polluted air could be safely breathed [82] and, more in general, poor use of limited resources in medicine, criminal justice, finance and in other domains [128]. Such undesirable outcomes (i) are dangerous in safety-critical applications, where even a single unforeseen mistake can lead to dramatic consequences, and (ii) undermine the human confidence in the models themselves, slowing down their adoption. It is thus extremely important to improve the trustworthiness of the models by reducing, if not ruling out, such unintended behaviors. As pointed out by Huang et al. in [59], in order to address such issue, DL researchers and practitioners can take inspiration from established practices in industries traditionally operating in safety critical domains, such as the automotive

and avionics industries, where trustworthiness is addressed predominantly within two processes:

1. the *certification process*: during which, the manufacturer needs to demonstrate to the relevant certification authority, (see, e.g., the European Aviation Safety Agency or the Vehicle Certification Agency) that the product behaves correctly with respect to a set of high-level requirements, and

2. the *explanation process:* which is held whenever needed during the lifetime of the product. The user manual provides a set of expected behaviours of the product that its user may frequently experience together with the steps to follow in order to understand any unexpected behaviour of the product.

However, while a lot of work (see e.g., [98, 76]) has been done to make models more explainable—and thus to create models able to pass the explanation process—comparatively very little has been done to create models which are guaranteed to comply with a given set of requirements—and thus to create models able to pass the certification process. This is surprising given that requirements specification and verification are the first fundamental steps in multiple software development life cycle models (see e.g., waterfall, iterative, etc.). Further, the work that has been done in order to create models able to pass the certification process has been mostly focused on two problems:

1. *verification problem* (see e.g., [93, 75]): given a neural network n and a property p, how to check whether p holds for n and return a mathematical proof (resp. a counterexample) if p holds (resp. does not hold), and

2. *testing problem* (see e.g., [77, 90]): how to exercise neural networks with large sets of test cases with the aim to either find bugs (i.e., counterexamples to a property) or provide assurance cases.

While the techniques developed for the above two problems certainly opened the path to create models able to pass the certification process, they suffer from different drawbacks. In particular, verification techniques typically suffer from a scalability problem, and testing techniques cannot give any guarantee that a neural network does not contain a bug.

In order to address the above limitations, in this thesis I propose to build models that are guaranteed to satisfy a given set of constraints *by design*. This is still a completely unexplored direction, and, as I show in the remaining of the thesis,

tackling the problem from this perspective allows for the creation of models (i) that are guaranteed to be compliant with the requirements, and (ii) that are scalable in the size of the network considered. Since the problem is very broad, in particular I focus on:

1. DL models for multi-label classification (MC) problems, and
2. requirements modelled as hard logical constraints.

The goal of the thesis is to create DL-based models for multi-label classification problems which

1. are guaranteed to satisfy the given set of requirements, and
2. are able to learn from the background knowledge specified in the constraints to get better performance.

In order to achieve such goal, I started by considering multi-label classification problems with simple hierarchical constraints and then I incrementally increased the expressivity of the constraints.

In the first phase of the project, I thus focused on hierarchical multi-label classification (HMC) problems, which are multi-label classification problems with hierarchical constraints over the output space of the form

$$A_1 \rightarrow A \tag{1.1}$$

expressing that A_1 is a subclass of A, i.e., that if a data point is associated with the class A_1, then it is also associated with the class A. For such problem, I developed a novel model called *coherent hierarchical multi-label classification neural network* (C-HMCNN(h)), which, given a network h for the underlying MC problem, exploits the hierarchy information to produce predictions guaranteed to satisfy the hierarchy constraints and has improved performance. C-HMCNN(h) is built upon two basic ingredients:

1. a constraint layer built on top of h, which extends to the upper classes the predictions made by h on the lower classes in the hierarchy, in order to ensure that the final outputs are coherent by construction with the hierarchy constraints, and
2. a loss function teaching C-HMCNN(h) when to exploit the hierarchy constraints, that is, when the prediction on the classes at lower levels of the hierarchy can be exploited to make predictions also for ones at the upper levels.

C-HMCNN(h) significantly differs from previous approaches for HMC problems based on neural networks. Indeed, the constraint layer is not a simple post-processing meant to guarantee the satisfaction of the hierarchy constraints, decoupled from the rest of the system. In C-HMCNN(h), the constraint layer (which is added at training time) and the underlying neural network h are tightly integrated, and I later show that it does not make sense to modify the constraint layer without modifying the procedure used for training h. Thanks to such integration, C-HMCNN(h) is able to outperform the current state-of-the-art models on 20 real-world and publicly available datasets.

Then, I considered constraints expressed as normal logic rules [74], i.e., expressions of the form:

$$A_1, \ldots, A_k, \neg A_{k+1}, \ldots, \neg A_n \rightarrow A, \qquad (0 \leq k \leq n), \tag{1.2}$$

which imposes that whenever the classes $A_1, \ldots, A_k$ are predicted, while $A_{k+1}, \ldots, A_n$ are not, then also the class A should be predicted. These constraints generalize hierarchy constraints (corresponding to the case $n = k = 1$). I call MC problems with a set of constraints in such an extended syntax *logically constrained multi-label classification* (LCMC) problems. For this class of problems, I proposed *coherent-by-construction network* (CCN(h)), a model that, given a network h for the underlying MC problem, is able to (i) produce predictions guaranteed to satisfy the constraint and (ii) exploit the information contained in the constraints to improve performance. CCN(h) is the first model able to deal with MC problems with such expressive hard constraints. In particular, given a LCMC problem with stratified constraints [2] and a set $\mathcal{H}$ of classes initially predicted by h, CCN(h) is able to compute in linear time in the number of constraints the unique minimal set of classes $\mathcal{M}$ that

1. extends $\mathcal{H}$, that is, such that $\mathcal{H} \subseteq \mathcal{M}$, and

2. is coherent with (satisfies) the constraints, that is, such that, given (1.2), $A \in \mathcal{M}$ whenever $\{A_1, \ldots, A_k\} \subseteq \mathcal{M}$ and $\{A_{k+1}, \ldots, A_n\} \cap \mathcal{M} = \emptyset$.

Indeed, for a non-stratified set of constraints expressed as normal rules, there can be no or more than one minimal set of classes having the above two properties, and determining the non-existence or computing one of them can take exponential time. CCN(h) has the same two basic ingredients of C-HMCNN(h):

1. a constraint layer built on top of h, which extends the predictions made by h in order to ensure that the predictions are coherent by construction with the constraints, and

2. a loss function, teaching CCN(h) when to exploit the constraints, that is, in the presence of (4.1), when to exploit the prediction on $\{A_1, \ldots, A_n\}$ to make predictions on A.

In CCN(h), like in C-HMCNN(h), the constraint layer and h are tightly integrated, and the result is a system that significantly differs from what it is considered the standard approach to LCMC problems, consisting in applying the constraint layer as a simple post-processor to a state-of-the-art MC system. Thanks to such integration, CCN(h) is able to beat the state-of-the-art models on 18 real-world and publicly available datasets according to six different metrics. Further, CCN(h) generalises C-HMCNN(h), and thus CCN(h) is able to outperform the specialised state-of-the-art HMC models on the already mentioned 20 different HMC datasets.

Finally, in order to demonstrate the importance of the problem handled, I created the ROad event Awareness Dataset with logical Requirements (ROAD-R), the first publicly available dataset for autonomous driving with requirements expressed as logical constraints. ROAD-R extends the ROAD dataset [111], by introducing a set of requirements that specifies the space of admissible outputs. ROAD consists of of 22 relatively long ($\sim$8 minutes each) videos annotated with *road events*. A road event corresponds to a tube, i.e., a sequence of frame-wise bounding boxes linked in time. The goal is to predict the set of classes associated to each bounding box. I manually annotated ROAD-R with 243 constraints expressed as propositional logic formulas. Each constraint has been verified to hold for each bounding box. A typical constraint is thus "a traffic light cannot be red and green at the same time", while there are no constraints like "pedestrians should cross at crossings", which should always be satisfied in theory, but which might not be in real-world scenarios. Thanks to ROAD-R, I was able to show that current state-of-the-art models do not learn the requirements just from the data points, as more than 90% of the times they produce predictions that violate the constraints, and that is possible to exploit the given requirements to create models that (i) have a better performance, and (ii) are guaranteed to be compliant with the requirements themselves.

The main contributions of this thesis can thus be briefly summarized as follows:

1. I propose a novel model for HMC problems, denoted C-HMCNN(h), and a novel model for LCMC problems, denoted CCN(h), which I prove to be an extension of C-HMCNN(h).

2. I prove that CCN(h)'s predictions are guaranteed to be coherent with the constraints expressed as normal rules, and hence that C-HMCNN(h)'s predictions are guaranteed to be coherent with the hierarchical constraints.

3. I show that CCN(h) (and hence C-HMCNN(h)) can be implemented on GPUs using standard libraries.

4. I perform an extensive experimental analysis on 38 real-world datasets, showing that CCN(h) outperforms the current state-of-the-art models on both HMC and LCMC problems.

5. As a case study, I introduce a new dataset, called ROAD-R, which represents the first publicly available dataset for autonomous driving with requirements expressed as logical constraints.

6. Thanks to ROAD-R, I show that current state-of-the-art models do not learn the requirements just from the data points, and that they often produce predictions that violate the constraints.

7. I show how to create autonomous driving models which are always compliant with the annotated requirements and that exploit the background knowledge expressed by the requirements to achieve better performance.

The structure of this thesis replicates the chronological development of my DPhil project.

1. In Chapter 2, I introduce the preliminary concepts necessary to understand the rest of the thesis and describe the most important and/or closely related developed models. The latter is used throughout the dissertation to comparatively discuss and evaluate my own results.

2. In Chapter 3, the model for multi-label classification problems with hierarchical constraints (C-HMCNN(h)) is introduced. The results showed in this chapter were published in [48].

3. In Chapter 4, the model for multi-label classification problems with constraints expressed as normal logic rules (CCN(h)) is introduced. The results showed in this chapter were published in [49].

4. In Chapter 5, ROAD-R, the limitations of current state-of-the-art models, and some possible solutions exploiting the constraints to address those limitations are introduced. The paper introducing the results showed in this chapter is currently under submission [50].

5. In Chapter 6, a bird-eye view of the related work published in the field of deep learning with logical constraints is given. The related survey paper was published in [51].

6. In Chapter 7, the general conclusions and perspectives are exposed.

Chapter 2

Preliminaries

In this chapter, I introduce the preliminary concepts necessary to understand the rest of the thesis. In particular, I first give an introduction to multi-label classification problems, then I focus on hierarchical multi-label classification problems, and then I conclude by giving an overview of the more general problem of learning with logical constraints. For each problem, I describe the most important and/or closely related developed models, which are used in the next chapters to comparatively discuss and evaluate the results being presented.

2.1 Multi-label Classification

A *multi-label classification* (*MC*) problem $\mathcal{P}$ is a pair $(\mathcal{A}, \mathcal{X})$ where $\mathcal{A}$ is a finite set of *classes* (also called *class labels* or simply *labels*), denoted by $A, A_1, A_2, \ldots$, and $\mathcal{X}$ is a finite set of pairs (x, y) where $x \in \mathbb{R}^D (D \geq 1)$ is a *data point*, and $y \subseteq \mathcal{A}$ is the *ground truth* of x, that is, the set of classes associated with x. MC problems are thus a generalization of the standard multi-class classification problem in which the classes are assumed to be mutually exclusive.

Many models have been developed for this class of problems, and they can be broadly divided in two categories:

1. **Problem transformation models**: these models transform our MC problem into possibly many binary or multi-class classification problems.

2. **Algorithm adaptation models**: these models rely on adapting the standard algorithm normally used for binary classification/multi-class classification to directly perform multi-label classification.

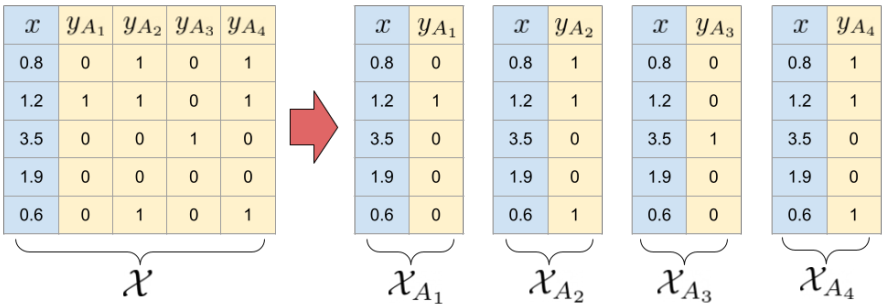


Figure 2.1: Example of problem transformation via binary relevance.

The algorithm adaptation methods are different from the problem transformation approaches because they directly adapt the algorithm rather than transforming the problem into different subsets of problems. I will now present some of the most popular methods for each of these categories.

2.1.1 Problem Transformation Models

Models in this category are mostly based on the following techniques:

1. Binary Relevance (BR) [9],
2. Classifier Chains (CC) [97], and
3. Label Powerset (LP) [9].

I will now explain in detail each one of them.

Binary Relevance (BR) Given an MC problem $\mathcal{P} = (\mathcal{A}, \mathcal{X})$, the BR approach transforms $\mathcal{P}$ into a set of binary classification sub-problems $\{(A, \mathcal{X}_A) \mid A \in \mathcal{A}\}$ where $\mathcal{X}_A$ can be built from $\mathcal{X}$ by taking each pair (x, y) and creating a new pair (x, y_A) where $y_A = 1$ if $A \in y$, and $y_A = 0$ otherwise. For each subproblem $(A, \mathcal{X}_A)$ we can then use a binary classifier (e.g., logistic regression model), normally called *base classifier*, to make predictions for each subproblem. Given a datapoint, each base classifier returns 1 if the datapoints belongs to the class, and 0 otherwise. The final prediction is then the set of classes for which the base classifiers returned 1. The BR approach can lead to good performance, but it does not take into account label correlations. An example of problem transformation via binary relevance can be seen in Figure 2.1. In order to obviate to this problem, Feng et al. in [41] proposed CollAboration based

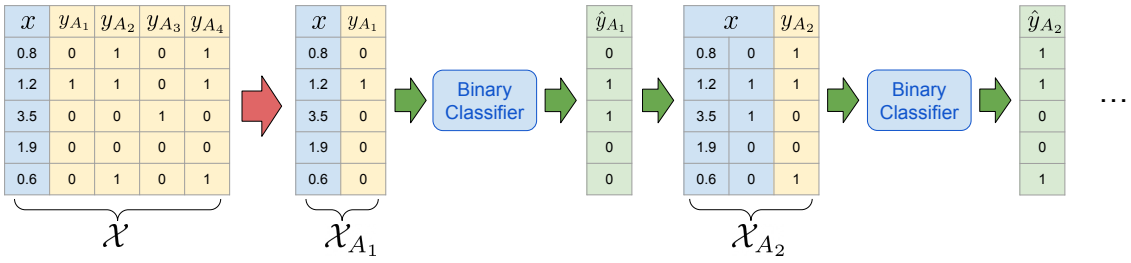


Figure 2.2: Example of problem transformation via classifier chain.

Multi-labEl Learning (CAMEL), which makes the assumption that for each individual label, the final prediction involves the collaboration between its own prediction and the predictions of other labels. Thus, the basic intuition behind their model is that given an MC problem $\mathcal{P}$ and a base classifier f for $\mathcal{P}$, the prediction for each datapoint x and label $A \in \mathcal{A}$ is given by:

$$\alpha f_A(x) + (1 - \alpha) \sum_{B \in \mathcal{A}, B \neq A} s_{AB} f_B(x)$$

where:

- α is a user defined parameter controlling the collaboration degree among labels,
- $f_A(x)$ is the prediction for x returned by the base classifier trained on A, and
- s_{AB} is a learnt parameter that indicates the degree of correlation between the labels A and B.

Classifier Chain (CC) The CC approach builds upon the BR one. The basic intuition behind this method is to transform the MC problem into a chain of binary classification problems, where a subsequent binary classifier in the chain is built upon the predictions of the preceding ones (the first classifier on the chain is learned using only the datapoint features). Thus, given an MC problem $\mathcal{P} = (\mathcal{A}, \mathcal{X})$, the CC approach transforms $\mathcal{P}$ into a chain (i.e., totally ordered set) of $|\mathcal{A}|$ binary classification sub-problems where, for $1 \leq j \leq |\mathcal{A}|$, each instance (x, y) of $\mathcal{X}$ is mapped into an instance $((x, \hat{y}_{A_1}, \ldots, \hat{y}_{A_{j-1}}), y_{A_j})$ of the jth binary classification sub-problem in which

1. $\hat{y}_{A_1}, \ldots, \hat{y}_{A_{j-1}}$ are the predictions on the previous sub-problems in the chain, and
2. $y_{A_j} = 1$ if $A_j \in y$ and $y_{A_j} = 0$, otherwise.

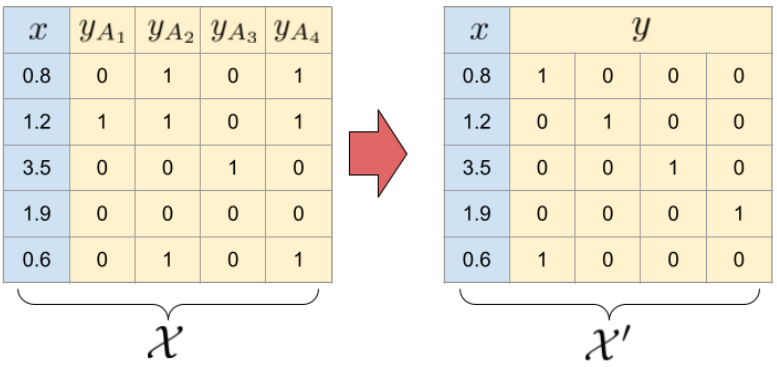


Figure 2.3: Example of problem transformation via label powerset.

The total order is given by the user, and the results can greatly vary for different orders of chains. However, the great advantage of CC with respect to BR is that, thanks to the chaining mechanism, the correlations among labels are here taken into account. An example of problem transformation via classifier chain can be seen in Figure 2.2. CC models are often used in an ensemble, normally called Ensemble of Classifier Chains (ECC). The ensemble is obtained by taking a finite number of CC classifiers, each CC classifier is trained with:

- a random chain ordering, and
- a random subset of the dataset,

and then, at inference time, majority voting is used for each label to decide whether to predict it or not.

Label Powerset (LP) The LP approach transforms the MC problem into a multi-class classification problem by treating every possible combinations of classes appearing in the training set as a different class. Thus, given an MC problem $\mathcal{P} = (\mathcal{A}, \mathcal{X})$, the LP approach transforms $\mathcal{P}$ into a new problem $\mathcal{P}$'$= (\mathcal{A}', \mathcal{X}')$ where $\mathcal{A}'$ is now a set of mutually exclusive classes, each corresponding to a different combination of labels appearing in the training set, and $\mathcal{X}'$ is built from $\mathcal{X}$ by taking each pair (x, y) and creating a new pair (x, y') where y' is the unique class in $\mathcal{A}'$ corresponding to the combination of classes in y. LP has the advantage that the correlations among labels are again taken into account, however, it suffers of scalability problems (since in the worst case scenario we have to create $2^{|\mathcal{A}|}$ classes), and of the incompleteness problem, given by the fact that the model will be able to predict only those combinations of classes that appear in the training set. An example of problem transformation via

classifier chain can be seen in Figure 2.3. In order to mitigate thescalability problem, Tsoumakas and Vlahavas (2007) proposed RAndom k-LabELsets (RAKEL) which is an ensemble of LP models. RAKEL divides the label space into equal partitions of size k (a user-defined hyperparameter), trains a LP classifier per partition and makes the final prediction on the basis of the result of all the trained classifiers.

2.1.2 Algorithm Adaptation Models

I will now show how some popular binary and multi-class classification methods can be adapted for the MC task. In particular, I will focus on the following models:

1. Multi-label Lazy k-Nearest Neighbours (ML-KNN) [140],
2. Multi-label Classification Decision Trees (MC-DT) [20], and
3. Multi-label Classification Neural Networks.

Multi-label Lazy k-Nearest Neighbours The ML-KNN algorithm is adapted from the traditional k-nearest neighbour (k-NN algorithm). In the standard k-NN method, each datapoint is assigned to the most common class among its k nearest neighbors. In the ML-KNN algorithm, on the other hand, given a multi-label classification problem $\mathcal{P}$=$(\mathcal{A}, \mathcal{X})$ Zhang and Zhou in [140] define for each datapoint x and class $A \in \mathcal{A}$,:

- H_1^A (resp. H_0^A) as the event that the datapoint x is associated (resp. is not associated) to class A,
- $C_x(A)$ as the number of nearest neighbours of x associated to class A,
- $E^A_{C_x(A)}$ as the event that exactly $C_x(A)$ datapoints in the nearest neighbours of x are associated to A.

Then, it is possible to determine whether x is associated to the class A using the maximum a posteriori principle:

$$\hat{y}_A = \text{argmax}_{b \in \{0,1\}} P(H_b^A \mid E_{C_x^A(A)})$$

which, using the Bayesian rule, can be calculated as:

$$\hat{y}_A = \text{argmax}_{b \in \{0,1\}} P(H_b^A) P(E_{C_x^A(A)} \mid H_b^A).$$

Both the above prior and posterior probabilities can be directly estimated from the training set based on frequency counting.

Multi-label Classification Decision Trees MC-DT is an adapted C4.5 algorithm [94] for multi-label classification. C4.5 is an algorithm for building decision trees in which the tree is constructed top down. For each node the attribute is chosen which best classifies the remaining training examples. This is decided by considering the information gain, i.e., the difference between the entropy of the whole set of remaining training examples and the weighted sum of the entropy of the subsets caused by partitioning on the values of that attribute. Since in C4.5 there is the assumption that each datapoint belongs to a single class, then the entropy can be calculated as:

$$-\sum_{A\in\mathcal{A}} p(A)\log(p(A))$$

$\mathcal{A}$ being the set of classes of the considered problem and $p(A)$ being the relative frequency of class A in the training set. In order to extend the algorithm to the multi-label case, Clare and King in [20] instead estimated the number of bits needed to describe membership or non-membership of each class and thus calculated:

$$-\sum_{A\in\mathcal{A}} p(A)\log(p(A)) + (1-p(A))\log(1-p(A)),$$

where $p(A)$ has the same reading as in the above equation.

Multi-label Classification Neural Networks Multi-label Classification Neural Networks are the adaptation of neural networks for multi-label classification problems. Given a problem $\mathcal{P}= (\mathcal{A}, \mathcal{X})$, the simplest neural-based model for $\mathcal{P}$ is a simple feed-forward neural networks with $|\mathcal{A}|$ neurons in the output layer. In order to clip the output values in $[0, 1]$, each of the output neurons is built with *sigmoid* (σ) non-linearity, whose function is defined as follows:

$$\sigma(x) = \frac{1}{1-\exp(-x)}. \tag{2.1}$$

The standard loss function to train this network is the *binary cross-entropy loss* which, given a prediction $\hat{y} \in [0, 1]^{|\mathcal{A}|}$ and its ground truth $y \in \{0, 1\}^{|\mathcal{A}|}$, is defined as:

$$\mathcal{L} = \sum_{i=1}^{|\mathcal{A}|} l(\hat{y}_i, y_i)$$

$$l(\hat{y}_i, y_i) = -y_i\log(\hat{y}_i) - (1-y_i)\log(1-\hat{y}_i).$$

As expected and as we will see in the remaining of the thesis, more complex neural networks can be built for multi-label classification problems.

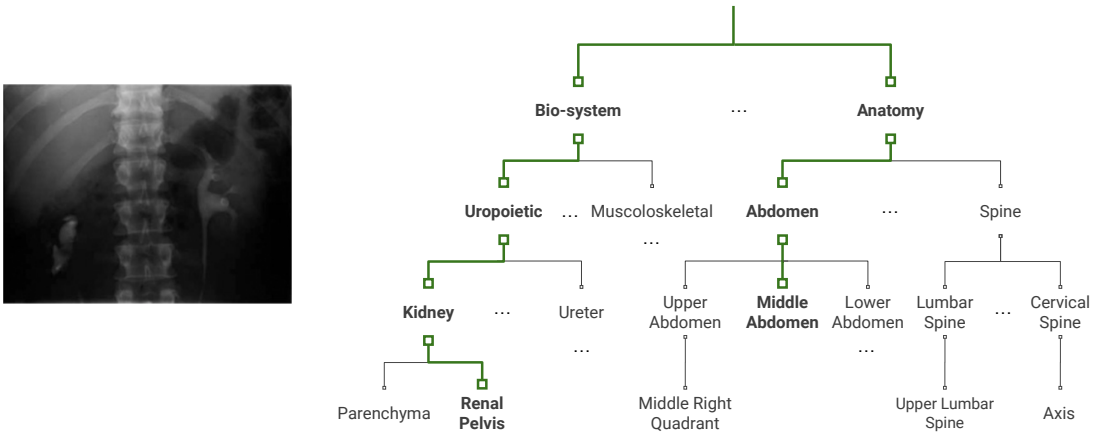


Figure 2.4: Example of an HMC task in the medical domain. On the left, a medical image to be annotated (image from [33]). On the right, part of the hierarchy in which the classes are arranged for the task. The labels associated to the image shown are in bold.

2.2 Hierarchical Multi-label Classification

A *hierarchical multi-label classification* (*HMC*) problem $(\mathcal{P}, \Pi)$ consists of an MC problem $\mathcal{P}$ and a finite set Π of *(hierarchy) constraints* of the form

$$A_1 \rightarrow A, \tag{2.2}$$

where A_1 and A are classes, such that the graph associated to Π with an edge from A_1 to A for each such constraint in Π is acyclic. Thus, given an HMC problem $(\mathcal{P}, \Pi)$, a model m for $(\mathcal{P}, \Pi)$ has to be coherent with the hierarchy constraints Π in $\mathcal{P}$, that is, m has to predict A whenever it predicts A_1, for each constraint (2.2) in Π.

HMC problems naturally arise in many domains, such as image classification [28, 33, 34], text categorization [65, 69, 99], and functional genomics [5, 19, 129]. They are very challenging for two main reasons: (i) they are normally characterized by a great class imbalance, because the number of data points per class is usually much smaller at deeper levels of the hierarchy, and (ii) the predictions must be coherent with (i.e., satisfy) the hierarchy constraints. Consider, for example, the task proposed by [33], where a radiological image has to be annotated with an IRMA code specifying, among others, the biological system examined. In Figure 2.4, I show an image present in the dataset (on the left) and part of the IRMA hierarchy used to annotate the

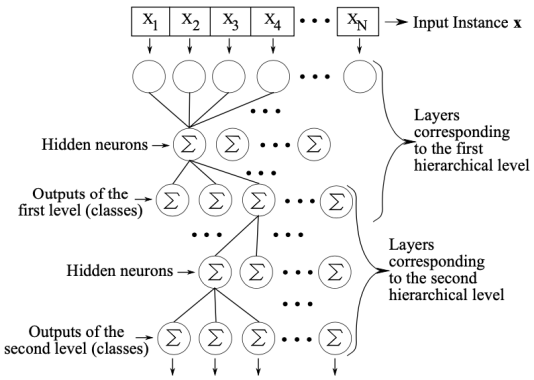


Figure 2.5: Topology of HMC-LMLP for a two-level hierarchy (figure from [13]).

images (on the right). As can be deducted from the hierarchy in Figure 2.4, it is expected to have many more *bio-system* images than *renalPelvis* images, making the class *renalPelvis* harder to predict. Furthermore, the prediction {*renalPelvis*} alone should not be possible given the constraint

$$renalPelvis \rightarrow bio\text{-}system, \tag{2.3}$$

stating that the renal pelvis is a bio-system, that is, that whenever *renalPelvis* is predicted, also *bio-system* should be predicted.

According to Silla and Freitas in [109] the HMC methods can be broadly divided in two categories:

1. local methods, which decompose the problem into smaller classification ones, and then the solutions are combined to solve the main task, and
2. global methods, which consist of single models able to map objects with their corresponding classes in the hierarchy as a whole.

I will now present the most important models for each of the above categories.

2.2.1 Local Methods

Local approaches can be further divided based on the strategy that they deploy to decompose the main task. In this way, three subcategories are obtained:

1. local classifiers per level,
2. local classifiers per node, and
3. local classifiers per parent node.

Local Classifiers per Level Local classifiers per level train a different classifier for each level of the hierarchy. The most famous model belonging to this category is Hierarchical Multi-Label Classification with Local Multi-Layer Perceptron (HMC-LMLP) [13]. HMC-LMLP consists of one multi-layer perceptron for each level of the hierarchy. Additionally, the predictions for each level of the hierarchy are then used as inputs for the neural network associated to the next level. A visual representation of the topology of the network is given in Figure 2.5. Notice that, after the final predictions for a datapoint are provided by HMC-LMLP, a post-processing phase is necessary is order to correct inconsistencies which may have occurred during the classification, i.e., when a subclass is predicted but one of its superclass is not.

HMC-LMLP was later extended in two different models, HMCN-F and HMCN-R, both presented in [132]. In these models, instead of having a perceptron for each hierarchical level, there is a unique model whose weights are either "global weights", i.e., trained to learn the entire hierarchy, or "local" weights, i.e., trained to learn the output for a specific level of the hierarchy. For this reason, these models are considered to be "hybrid" models: as the global methods they are a single model trying to learn the entire hierarchy, however, like the local models, some of their weights are trained for a specific level of the hierarchy. The difference between the two models lies in the fact that HMCN-F has a distinct layer for each level of the hierarchy (and thus its number of parameters grows with the depth of the hierarchy), while HMCN-R exploits a recurrent neural network based architecture to share the weights for all the level of the hierarchy (thus keeping the number of its parameters constant with respect to the depth of the hierarchy). As we will see later, while HMCN-R is convenient from a size perspective, it also leads to worse results in terms of performance.

Local Classifiers per Node Local classifiers per node train a classifier for each node of the hierarchy. For example, in [16], a linear classifier is trained for each node of the hierarchy, and the training set for each node is built such that it only consists of samples belonging to its parent. At inference time, these node classifiers are evaluated—starting from each root—in the following top-down fashion: the root is labelled by evaluating its node classifier; if a node has been labelled 1, then each child is labelled by evaluating its node classifier. On the other hand, if a node is labelled 0 then all of its descendants are labelled 0. Note that this evaluation scheme can only generate set of labels that respect the underlying taxonomy.

Local Classifiers per Parent Node Local classifiers per parent node train a different classifier per parent node in the hierarchy. An example of a model belonging to this category has been proposed by Wu et al. in [134]. In this work, the authors propose to divide the hierarchy in *splits*, i.e., one level sub-trees including one parent node and all its children nodes, together with the links between the parent node and children nodes, and then train a classifier for each split. In order to train each classifier, the authors transform the multi-label classification problem into a standard classification problem by using the label powerset problem transformation method explained previously in this chapter (Section 2.1.1). As for the model proposed in [16], at inference time the split classifiers are evaluated in a top-down fashion: the classifiers for the root nodes are evaluated first, and then each split-based classifier is run only when a positive prediction is returned by its parent split-based classifier.

2.2.2 Global Methods

Global methods consist of single models able to map objects with their corresponding classes in the hierarchy as a whole. One of the most well-known models in the HMC literature—called Clus-HMC [129]—is indeed a global model. Clus-HMC consists of a single predictive clustering tree for the entire hierarchy. The algorithm at the base of the construction of the decision tree is the standard top-down induction of decision trees—like C4.5 [94]—however, the authors define a new variance function to minimize every time a new node in the tree is created. Such variance has been defined taking into account the hierarchical structure of the labels, and thus, given a set of n datapoints $\mathcal{S}$, each having ground truth $y_i \in [0,1]^m$ $(1 \leq i \leq n)$ we can define the variance of this set as:

$$Var(\mathcal{S}) = \frac{\sum_{i=1}^{n} d(v_i, \overline{v}(\mathcal{S}))}{|\mathcal{S}|}$$

where $\overline{v}(\mathcal{S})$ is the mean set label of $\mathcal{S}$ (i.e., the mean over all v_i for $1 \leq i \leq n$) and:

$$d(v_1, v_2) = \sqrt{\sum_{j=1}^{m} w(c_j)(v_{1,j} - v_{2,j})}$$

where the class weights $w(c)$ decrease with the depth of the class in the hierarchy (e.g., $w(c) = w_0^{depth(c)}$, with $0 < w_0 < 1$). Further, at inference time, in standard decision trees the majority class is the tree's prediction for examples arriving in the leaf. However, this does not apply in the HMC setting. For each leaf node and set of datapoints $\mathcal{S}$ assigned to that node, the authors then take the mean set label $\overline{v}(\mathcal{S})$, and, for each class j they say that a datapoint ending in that leaf node belongs to

the class c_j ($1 \leq j \leq m$), if $\overline{v}_j(\mathcal{S}) > \theta_j$, θ_j being a user defined threshold. To ensure that the predictions fulfill the hierarchy constraint (whenever a class is predicted its superclasses are also predicted), it suffices to choose $\theta_k \leq \theta_j$ whenever c_k is at higher levels of the hierarchy than c_j.

This work was later extended in [103], where Clus-Ens, an ensemble of Clus-HMC, is proposed. In order to create the ensemble method, the authors chose the bagging method [10]. In the paper, they declare that, in the preliminary experiments, they tested three different ways to create the ensemble model: (i) bagging [10], (ii) random forest [11], and (iii) an adapted version of the boosting approach for regression trees proposed in [38], and that the simple bagging led to the best performance. Clus-Ens takes as input the parameter k, denoting the number of trees in the ensemble. In order to make predictions, the average of all class vectors predicted by the k trees in the ensemble is computed, and then the threshold is applied as before. This ensures that the hierarchy constraint holds.

2.3 Learning with Logical Constraints

The general problem of *learning with logical constraints* can be formalized as a couple $(\mathcal{P}, \Pi)$ where:

1. $\mathcal{P}$ represents a standard machine learning problem (e.g., regression, multi-label classification etc.) and is a pair $(\mathcal{C}, \mathcal{X})$ where:

   - $\mathcal{C}$ is a pair $(\mathcal{I}, \mathcal{O})$, where $\mathcal{I} = I_1, I_2, \ldots, I_d$ ($d \geq 1$) are the *input features*, and $\mathcal{O} = O_1, O_2, \ldots, O_n$ ($n \geq 1$) are the *outputs*. Each input feature I (resp., output O) is associated with a non-empty domain D_I (resp., D_O) of *values*, and I (resp., O) is *Boolean* when $D_I = \{0, 1\}$ (resp., $D_O = \{0, 1\}$). $D_{\mathcal{I}} = D_{I_1} \times \ldots \times D_{I_d}$ (resp., $D_{\mathcal{O}} = D_{O_1} \times \ldots \times D_{O_n}$) is the set of the possible inputs (resp., outputs). A *data point* is an element of $D_{\mathcal{I}}$.
   - $\mathcal{X}$ is a finite set of pairs (x, y), where x is a (possibly partially specified) data point, and y is its (possibly partially specified) *ground truth*.

2. Π is a finite non-empty set of first-order constraints, which delimit the set of outputs that can be meaningfully associated with each data point. We assume that the constraints are written using (i) for each Boolean (resp., non-Boolean) input feature I a corresponding d-ary predicate (resp., function) I: intuitively, $I(x)$ (resp., $I(x) = z$) means that the value of the input feature I in the data

Operation	Minimum/Gödel	Product	Lukasiewicz
$\neg x$	$1 - x$	$1 - x$	$1 - x$
$x \wedge y$	$\min(x, y)$	$x \cdot y$	$\max(0, x + y - 1)$
$x \vee y$	$\max(x, y)$	$x + y - x \cdot y$	$\min(1, x + y)$
$x \Rightarrow y$	$x < y ? 1 : y$	$\min(1, \frac{y}{x})$	$x < y ? 1 : 1 - (x - y)$

Table 2.1: Mapping from basic Boolean operations to continuous functions according Gödel, Product, and Lukasiewicz t-norms.

point x is 1 (resp., z), and analogously for each Boolean (resp., non-Boolean) output O, (ii) variables ranging over data points, (iii) specific values, and (iv) possibly other logical symbols (like "$\neg$", "$\wedge$", "$\vee$", "$\rightarrow$", "$\forall$", "$\exists$", "$=$", "$\geq$", and "$+$"). To simplify the formal treatment and save space, we assume that each constraint is closed and in prenex form (i.e., that each variable is either existentially or universally quantified at the beginning of the constraint).

Notice that our definitions of both $\mathcal{X}$ and Π are very general. This allows for the specification of (i) both supervised and semisupervised problems thanks to the generality of $\mathcal{X}$, and (ii) logical constraints with different level of expressivity thanks to the generality of Π.

Many models fit in the very general definition given above, in the remaining of this section I will describe the ones that are most closely related to my thesis. In particular, I will describe the following models:

1. Semantic Based Regularization [30],
2. Semantic Loss [137],
3. Logic Tensor Networks [105], and
4. MultiPlexNet [55].

Semantic Based Regularization Semantic Based Regularization (SBR) was initially proposed in [30] as a way of using logical constraints to guide the training of kernel machines. The work was later extended in [32] to use logical constraints to guide the training of neural networks. Here we will focus on this second application. Given a generic multi-label classification problem $\mathcal{P}$ (which can be either fully supervised or semi-supervised) and a set of constraints expressed in first-order logic Π, Diligenti et al. first states that it is possible to convert any propositional logic formula into a

continuous and differentiable expression by using one the t-norms listed in Table 2.1. The authors generalise the above to first-order logic constraints by considering the set of grounded instances of the formula obtained by replacing a variable x with a datapoint, and then returning the average (resp. the maximum) of the t-norms of the grounded instances in the case x is universally (resp. existentially) quantified. When multiple universally or existentially quantified variables are present, the conversion is recursively performed from the outer to the inner variables.

Example 2.3.1. Let $\mathcal{P}= (\mathcal{A}, \mathcal{X})$ with $\mathcal{A} = \{A, B, C\}$ and $\mathcal{X} = \{(x_1, y_1), (x_2, y_2)\}$, and $\Pi = \{\forall x.((A(x) \wedge \neg B(x)) \vee C(x))\}$ stating that, for each datapoint in $\mathcal{X}$, x is associated to the class C, or to the class A but not to B. Further, let f be a neural network that maps each datapoint in $[0, 1]^3$. It is then possible to convert Π in a continuous and differentiable form able to quantify its degree of satisfaction for f when applied on $\mathcal{X}$ using one of the t-norms above. Here we use the Gödel t-norm:

$$\frac{1}{2}[\max(\min(f_A(x_1), 1 - f_B(x_1)), f_C(x_1)) + \max(\min(f_A(x_2), 1 - f_B(x_2)), f_C(x_2))]$$

$f_A(x), f_B(x)$ and $f_C(x)$ being the outputs of the neural network for a datapoint x and classes A, B and C, respectively.

The great advantage of this method is that it is relatively easy to understand and exploit during training of any neural network. However, it is syntax dependent (i.e., logically equivalent constraints might have different t-norm values), and also it does not guarantee that the constraints in the end will be satisfied, as the constraints are simply added as a regularization term to the loss function.

Semantic Loss Semantic Loss is another way of mapping logical constraints to a neural network's loss function. In their framework Xu et al. suppose to have a supervised or semi-supervised multi-label classification problem $\mathcal{P}=(\mathcal{A}, \mathcal{X})$ and a set of constraint Π over $\mathcal{A}$ expressed in propositional logic. Given a variable instantiation $\mathbf{x}$ of $\mathcal{A}$ (i.e., an assignment of each $A \in \mathcal{A}$ to either $True$ or $False$) the semantic loss function and a neural network f for the problem:

$$\mathcal{SL} = -\log \sum_{\mathbf{x} \models \Pi} \prod_{\mathbf{x}_A \models A} f_A(x) \prod_{\mathbf{x}_A \models \neg A} (1 - f_A(x)),$$

where $\mathbf{x}_A$ represents the instantiation of the variable A in $\mathbf{x}$ and, given two formulas a and b, $a \models b$ means that a entails b. The final loss function used to train the network is given by the standard supervised loss function used to train the network (e.g., binary cross-entropy loss) added to $\mathcal{SL}$.

The advantage of this method with respect to SBR is that it is syntax independent. However, it can suffer of scalability problems, as it is necessary to list all the models of Π. Additionally, much like SBR, since the constraints are just mapped in the loss function there is no guarantee that the constraints will be actually satisfied for all datapoints.

Logic Tensor Networks Logic Tensor Networks (LTN) is a framework proposed in [105] (and later extended in [3]). LTNs are one of the possible realizations of Real Logic, which is a logic formalism introduced in the same work. Real Logic presents the same syntax as first-order logic, however, it has a different semantics that the authors call *concrete semantics*. To emphasise the fact that it is interpreted in a "real" world, the authors use the term (semantic) grounding, denoted by $\mathcal{G}$, instead of the more standard interpretation term. A grounding $\mathcal{G}$ for a Real Logic $\mathcal{L}$ with signature $(\mathcal{C}, \mathcal{F}, \mathcal{P}r)$ is a function defined on the signature of $\mathcal{L}$ satisfying the following conditions:

- $\mathcal{G}(c) \in \mathbb{R}^n$ for every constant symbol $c \in \mathcal{C}$,
- $\mathcal{G}(f)$ is a map $\mathbb{R}^{n \cdot \alpha(f)} \to \mathbb{R}^n$ for every $f \in \mathcal{F}$, $\alpha(f)$ being the arity of f, and
- $\mathcal{G}(P)$ is a map $\mathbb{R}^{n \cdot \alpha(P)} \to [0, 1]$ for every $P \in \mathcal{P}r$, $\alpha(P)$ being the arity of P.

Given a grounding $\mathcal{G}$ respecting all the above properties, the semantics of closed terms and atomic formulas is defined as follows:

$$\begin{aligned} \mathcal{G}(f(t_1, \ldots, t_m)) &= \mathcal{G}(f)(\mathcal{G}(t_1), \ldots, \mathcal{G}(t_m)), \\ \mathcal{G}(P(t_1, \ldots, t_m)) &= \mathcal{G}(P)(\mathcal{G}(t_1), \ldots, \mathcal{G}(t_m)), \end{aligned} \tag{2.4}$$

while the semantics for connectives can be defined according to some fuzzy logics. If, for example, we choose the Lukasiewicz t-norm, then we obtain:

$$\begin{aligned} &\mathcal{G}(\neg\phi) = 1 - \mathcal{G}(\phi), \\ &\mathcal{G}(\phi \wedge \psi) = \max(0, \mathcal{G}(\phi) + \mathcal{G}(\psi) - 1), \\ &\mathcal{G}(\phi \vee \psi) = \min(1, \mathcal{G}(\phi) + \mathcal{G}(\psi)), \\ &\mathcal{G}(\phi \implies \psi) = \min(1, 1 - \mathcal{G}(\phi) + \mathcal{G}(\psi)), \end{aligned} \tag{2.5}$$

where ϕ and ψ are closed clauses.

Given the above, it is possible to define a LTN as a realization of Real Logic. For example, in [105], a LTN is built in the following way:

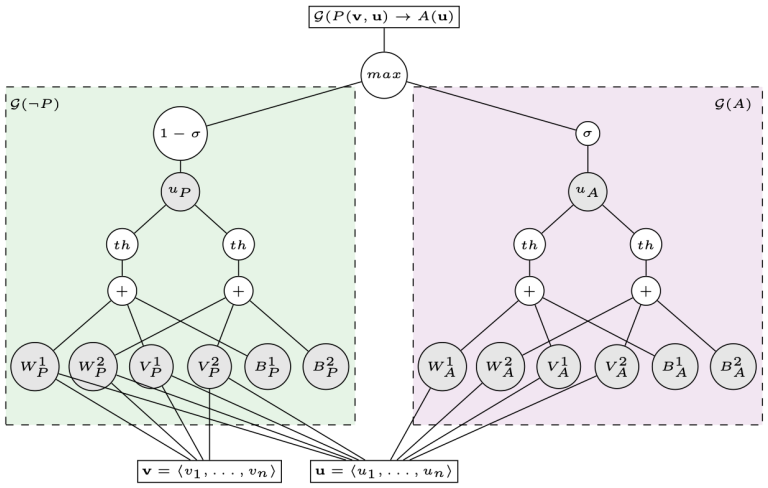


Figure 2.6: LTN for $\neg P(x, y) \to A(y)$, with $\mathcal{G}(x) = \mathbf{v}$, $\mathcal{G}(y) = \mathbf{u}$ and $k = 2$. Image from [106].

- if f is a function symbol of arity m and $\mathbf{v}_1, \ldots, \mathbf{v}_m \in \mathbb{R}^n$ are real vectors corresponding to the grounding of m terms then:

$$\mathcal{G}(f)(\mathbf{v}_1, \ldots, \mathbf{v}_m) = M_f \mathbf{v} + N_f$$

  where M_f is a $n \times mn$ matrix, N_f is a n vector and $\mathbf{v}$ is given by the concatenation of $\mathbf{v}_1, \ldots, \mathbf{v}_m$.

- if P is a predicate symbol of arity m and $\mathbf{v}_1, \ldots, \mathbf{v}_m \in \mathbb{R}^n$ are real vectors corresponding to the grounding of m terms then:

$$\mathcal{G}(P)(\mathbf{v}_1, \ldots, \mathbf{v}_m) = \sigma(u_P^T \tanh(\mathbf{v}^T W_P^{[1:k]} \mathbf{v} + V_P \mathbf{v} + B_P)),$$

  where W_P is a 3-D tensor in $R^{mn \times mn \times k}$, V_P is a matrix in $\mathbb{R}^{k \times mn}$, B_P is a vector in $\mathbb{R}_k$, σ is the sigmoid function as defined in Equation 2.1, and k is a user defined parameter.

- define the grounding of each connective according to the Gödel t-norm.

For example, if we chose $k = 2$, an example of a LTN for the constraint $\neg P(x, y) \to A(y)$, with $\mathcal{G}(x) = \mathbf{v}$ and $\mathcal{G}(y) = \mathbf{u}$ is given in Figure 2.6.

The advantage of this model with respect Semantic Loss is that it allows for more expressive constraints, while with respect to SBR is that existential quantifiers are not grounded into a finite disjunction, but are skolemized, and thus the closed world assumption is not required. However, like Semantic Loss and SBR, in LTN there is no guarantee that the constraints will be actually satisfied for all datapoints.

MultiplexNet MultiplexNet is a very recent model (developed later than the models presented in this thesis) introduced in [55]. MultiplexNet assumes that the constraints are expressed as a propositional logic formula in disjunctive normal form (DNF), in the following way:

$$\boldsymbol{\phi} = \phi_1 \vee \phi_2 \vee \ldots \vee \phi_K, \tag{2.6}$$

where each ϕ_i $(1 \leq k \leq K)$ represents a conjunction over linear inequalities.

Starting from the basic case, suppose that the constraints consists of a single linear inequality stating that, for each datapoint x, the final output $\hat{y}$ of the model should be such that $c\hat{y} \geq b$. Let $\tilde{x}$ be the unconstrained output of the model, it is then possible to obtain the final output $\hat{y}$ compliant to the constraint by:

1. constraining $\tilde{x}$ to be non-negative by applying a function g (e.g., the ReLU function),

2. then applying a linear transformation f such that $cf(g(\tilde{x})) \leq b$.

By construction $\hat{y} = cf(g(\tilde{x}))$ will be greater than or equal to c. It then follows that more complex conjunctions of constraints can be encoded by composing transformations like the ones above.

Let the constraints be of the form (2.6). The intuition behind MultiplexNet is that when K constraints are given in disjunction, the network needs to satisfy only one of them. As we have seen above, for each constraint ϕ_k $(1 \leq k \leq K)$, it is possible to find a transformation h_k that makes the final output $\hat{y}$ compliant with ϕ_k and thus with $\boldsymbol{\phi}$. The problem then becomes which transformation h_k to use for each datapoint. This problem can be solved by maximising for each datapoint x:

$$p_\theta(x) = p(h_k(x)|k)p(k),$$

where θ are the parameters of the model. However, k is a categorical variable and thus we cannot estimate it directly. We can use though a lower bound on the likelihood of the data, which can be obtained by introducing a variational approximation to the latent categorical variable k. This can be done using the standard form of the variational lower bound (ELBO):

$$\log p_\theta(x) = \mathbb{E}_{q(x)}[\log p_\theta(h_k(x)|k) + \log p(k) - \log q(k)] := \text{ELBO}(x).$$

where $q(k)$ is an arbitrary distribution used to approximate the true posterior distribution. Since in all the considered cases the dimensionality of the categorical variable is

small, the authors marginalised out the categorical variable as in [63], thus obtaining the final loss function:

$$\mathcal{L}(\theta, x) = -\sum_{k=1}^{K} q(k)[\log p_\theta(h_k(x)|k) + \log p(k) - \log q(k)].$$

A great advantage of this model is that it guarantees the satisfaction of the constraints, while it presents the disadvantage that the constraints must be written in DNF. Indeed, given a generic propositional logic formula, the size of its DNF representation might be exponential in the number of variables.

Chapter 3

Hierarchical Constraints

In this chapter, I focus on hierarchical multi-label classification (HMC) problems, in which every prediction must satisfy a given set of hard hierarchy constraints of the form (1.1). For an introduction to HMC problems, see Section 2.2. Many models have been specifically developed for HMC problems, and it is possible to distinguish between those that directly output predictions that are coherent with the hierarchical constraints (see, e.g., [7, 81]) and those that allow incoherent predictions and, at inference time, require an additional post-processing step to ensure their satisfaction (see, e.g., [14, 88, 125]). Most of the state-of-the-art HMC models based on neural networks belong to the second category (see, e.g., [14, 15, 132]), and different post-processing techniques can be applied in order to guarantee the coherency of their outputs with the constraints (see, e.g., [88]).

The model I developed and that is discussed in this chapter, C-HMCNN(h), is able to directly output predictions that are coherent with the hierarchical constraints despite being neural network based. In addition to such feature, C-HMCNN(h) presents other three desirable properties: (i) differently from other state-of-the-art models (see, e.g., [132]), its number of parameters is independent from the number of hierarchical levels, (ii) it can be easily implemented on GPUs using standard libraries, and (iii) it outperforms the state-of-the-art models Clus-Ens [103], HMC-LMLP [15], HMCN-F, and HMCN-R [132] on 20 commonly used real-world HMC benchmarks.

The rest of this chapter is organized as follows. In Section 3.1, the notation and terminology used is introduced. Then, in Section 3.2, the core ideas behind C-HMCNN(h) are illustrated on a simple HMC problem with just two classes, followed by the presentation of the general solution in Section 3.3. How to implement our solution on a GPU is discussed in Section 3.4, while the Experimental results are presented in Section 3.5. I conclude with an overview of the results presented in this chapter in Section 3.6.

3.1 Notation and Terminology

As stated in Section 2.1, a *multi-label classification* (*MC*) problem $\mathcal{P}$ is a pair $(\mathcal{A}, \mathcal{X})$ where $\mathcal{A}$ is a finite set of *classes* (also called *class labels* or simply *labels*), denoted by $A, A_1, A_2, \ldots$, and $\mathcal{X}$ is a finite set of pairs (x, y) where $x \in \mathbb{R}^D (D \geq 1)$ is a *data point*, and $y \subseteq \mathcal{A}$ is the *ground truth* of x, that is, the set of classes associated with x. Given a multi-label classification problem $\mathcal{P}$, a *model* m for $\mathcal{P}$ is a function $m(\,\cdot\,,\,\cdot\,)$ mapping every class A and every data point $x \in \mathbb{R}^D$ to $[0, 1]$. For every class A, the function $m_A \colon \mathbb{R}^D \to [0, 1]$ is defined by $x \mapsto m(A, x)$, for every data point $x \in \mathbb{R}^D$. A data point $x \in \mathbb{R}^D$ is *predicted* by m to belong to class A whenever $m_A(x)$ is greater than a user-defined *threshold* $\theta \in [0, 1]$.

As seen in Section 2.2, a *hierarchical multi-label classification* (*HMC*) problem $(\mathcal{P}, \Pi)$ consists of an MC problem $\mathcal{P}$ and a finite set Π of *(hierarchy) constraints* of the form

$$A_1 \to A, \tag{3.1}$$

where A_1 and A are classes, such that the graph associated to Π with an edge from A_1 to A for each such constraint in Π is acyclic. Given an HMC problem $(\mathcal{P}, \Pi)$, a model m for $(\mathcal{P}, \Pi)$ has to be coherent with the hierarchy constraints Π in $\mathcal{P}$. This is formally defined as follows.

Definition 3.1.1. *Let $(\mathcal{P}, \Pi)$ be an HMC problem. Let m be a model for $\mathcal{P}$. If for a data point and for a constraint $A_1 \to A$ in Π, m predicts A_1 but not A, then m commits a logical violation. If m commits no logical violations, then m is coherent with respect to Π.*

Given the above, whenever a model m is not guaranteed to satisfy a constraint (3.1), m is extended with a post-processing step to enforce $m_A(x) > \theta$ whenever $m_{A_1}(x) > \theta$ [14, 88, 125]. However, it is often common practice to require the stronger condition $m_{A_1}(x) \leq m_A(x)$, and the falsification of this condition is referred to as hierarchy violation [129, 132].

Definition 3.1.2. *Let $(\mathcal{P}, \Pi)$ be an HMC problem. Let m be a model for $\mathcal{P}$. If for a data point x and a constraint $A_1 \to A$ in Π, $m_{A_1}(x) > m_A(x)$, then m commits a hierarchy violation.*

If a model commits no hierarchy violations, then it also commits no logical violations (and so is coherent relative to the constraints), while the converse does not necessarily hold.

(a) Basic solution f^+.

(b) Basic solution g^+.

Figure 3.1: Visual representation of the basic solutions f^+ and g^+.

For ease of presentation, I often omit the dependency from data points, and simply write, for example, m_A instead of $m_A(x)$.

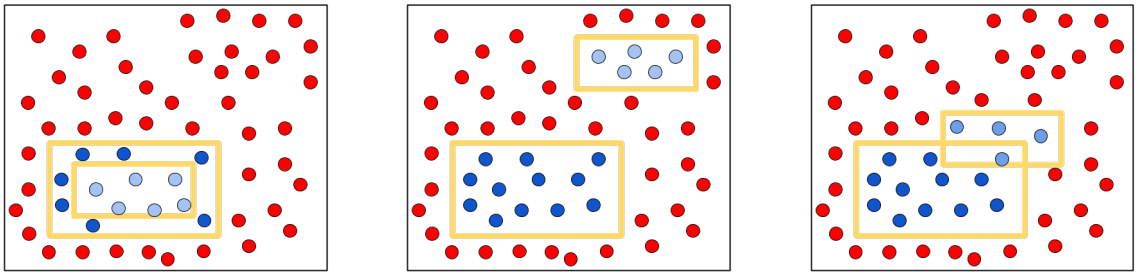
3.2 Basic Case

The final goal is to leverage standard neural network approaches for MC problems and then exploit the hierarchy constraints in order to produce coherent predictions and improve performance. Given such goal, I first present two basic approaches, exemplifying their respective strengths and weaknesses. These are useful to then introduce my solution, which is shown to present both the basic approaches' advantages without exhibiting their weaknesses. In this section, I thus assume to have just two classes A_1, A and the single constraint (3.1).

3.2.1 Basic Approaches

In the first basic approach, I treat the problem as a standard multi-label classification problem and simply set up a neural network f with one output per class to be learned. However, to ensure that no hierarchy violation happens, an additional post-processing step is needed. In this simple case, the post-processing could set the output for A_1 to be $\min(f_{A_1}, f_A)$ or the output for A to be $\max(f_A, f_{A_1})$. In this way, all predictions are always coherent with the unique hierarchical constraint considered: $A_1 \to A$.

The second basic approach is to build a network g with two outputs, one for A_1 and one for $A \setminus A_1$. This is obtained by be teaching a positive supervision to $g_{A\setminus A_1}$ for every datapoint that belongs to A but not to A_1, and a negative supervision in all the other cases. To then obtain the final output, I need an additional post-processing step in which each prediction for the class A is given by $\max(g_{A\setminus A_1}, g_{A_1})$. Considering the two above approaches, depending on the specific distribution of the data points, one solution may be significantly better than the other, and apriori I may not know which one it is.



(a) (b) (c)

Figure 3.2: Visualization of the basic problem. The blue datapoints are associated to the class A, the light blue datapoints are associated to both the classes A and A_1, and the red datapoints are associated with no classes.

To visualize the problem, assume that $D = 2$, and consider two rectangles R_1 and R_2 with R_1 smaller than R_2, like the two yellow rectangles in the subfigures of Figure 3.2. Assume $A_1 = R_1$ and $A = R_1 \cup R_2$. Let f^+ be the model obtained by adding a post-processing step to f setting $f^+_{A_1} = \min(f_{A_1}, f_A)$ and $f^+_A = f_A$, as in the works of [14, 15] and [42] (analogous considerations hold, if I set $f^+_{A_1} = f_{A_1}$ and $f^+_A = \max(f_A, f_{A_1})$ instead). A visual representation of the basic model f^+ is given in Figure 3.1a. Intuitively, I expect f^+ to perform well even with a very limited number of neurons when $R_1 \cap R_2 = R_1$, as in Figure 3.2a. However, if $R_1 \cap R_2 = \emptyset$, as in Figure 3.2b, I expect f^+ to need more neurons to obtain a similar performance. Consider the alternative network g, and let g^+ be the system obtained by setting $g^+_{A_1} = g_{A_1}$ and $g^+_A = \max(g_{A \setminus A_1}, g_{A_1})$. A visual representation of the basic model g^+ is given in Figure 3.1b. Then, I expect g^+ to perform well when $R_1 \cap R_2 = \emptyset$. However, if $R_1 \cap R_2 = R_1$, I expect g^+ to need more neurons to obtain a similar performance.[1] Further, it is difficult to know apriori what will the performance of both f and g when the rectangles are arranged as in Figure 3.2c.

To test our hypothesis, I implemented f and g as feedforward neural networks with one hidden layer with four neurons and *tanh* nonlinearity. I used the *sigmoid* non-linearity for the output layer (from here on, I always assume that the last layer of each neural network presents *sigmoid* non-linearity). f and g were trained with binary cross-entropy loss using Adam optimization [62] for 20k epochs with learning rate 10^{-2} ($\beta_1 = 0.9, \beta_2 = 0.999$). The datasets consisted of 5000 (50/50 train/test split) data points sampled from a uniform distribution over $[0, 1]^2$. The first four columns of Figure 3.3 show the decision boundaries of f^+ and g^+, while the decision boundaries

[1] I do not consider the model with one output for $A \setminus A_1$ and one for A, since it performs poorly in both cases.

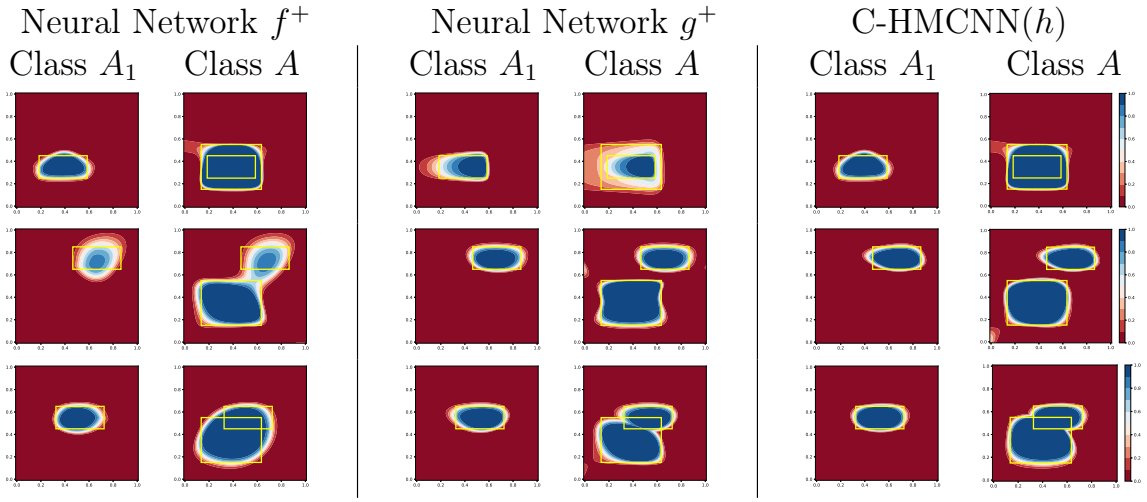


Figure 3.3: In all figures, the smaller yellow rectangle corresponds to R_1, while the bigger yellow one corresponds to R_2. The first row of figures corresponds to $R_1 \cap R_2 = R_1$, the second corresponds to $R_1 \cap R_2 = \emptyset$, and the third corresponds to $R_1 \cap R_2 \notin \{R_1, \emptyset\}$. First four columns: decision boundaries of f^+ and g^+ for the classes A_1 and A. Last two columns: decision boundaries of C-HMCNN(h) for the classes A_1 and A. In each figure, the darker the blue (resp., red), the more confident a model is that the data points in the region belong (do not belong) to the class (see the scale at the end of each row).

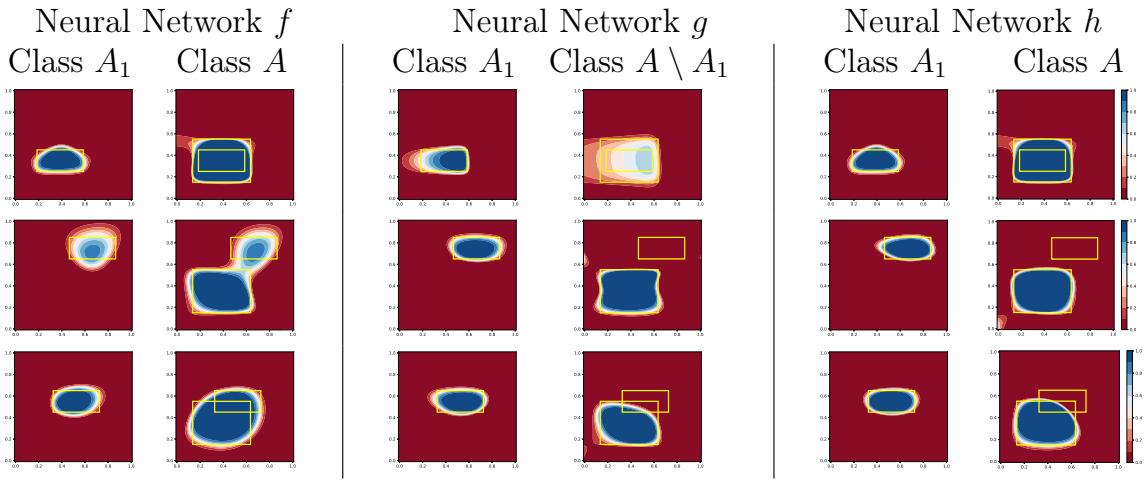


Figure 3.4: First four columns: decision boundaries of f (resp., g) for the classes A_1 and A (resp., A_1 and $A \setminus A_1$). Last two columns: decision boundaries of h for the classes A_1 and A.

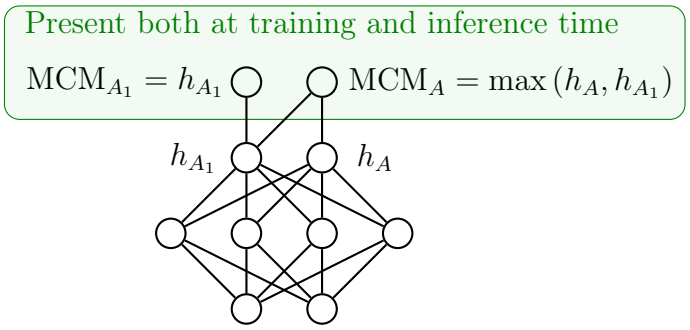


Figure 3.5: Visual representation of the Max Constraint Module for the basic case.

of f and g are reported in Figure 3.4. These figures highlight that f^+ (resp., g^+) approximates the two rectangles better than g^+ (resp., f^+) when $R_1 \cap R_2 = R_1$ (resp., $R_1 \cap R_2 = \emptyset$). In general, when $R_1 \cap R_2 \notin \{R_1, \emptyset\}$, I expect that the behavior of f^+ and g^+ depends on the relative position of R_1 and R_2.

3.2.2 My Solution

Ideally, I would like to build a neural network that is able to have roughly the same performance of f^+ when $R_1 \cap R_2 = R_1$, of g^+ when $R_1 \cap R_2 = \emptyset$, and better than both in any other case. I can achieve the desired behavior with a two steps process.

In the first step, I build a new neural network consisting of two modules: (i) a bottom module h with two outputs in $[0, 1]$ for A_1 and A, and (ii) an upper module, called *max constraint module* (MCM), consisting of a single layer that takes as input the output of the bottom module and imposes the hierarchy constraint. I call the obtained neural network the *coherent hierarchical multi-label classification neural network of* h, denoted C-HMCNN(h). Consider a data point x. Let h_{A_1} and h_A be the outputs of h for the classes A_1 and A, respectively, and let y_{A_1} and y_A be the ground truth for the classes A_1 and A, respectively. The outputs of MCM (which are also the output of C-HMCNN(h)) are:

$$\begin{aligned} \text{MCM}_{A_1} &= h_{A_1}, \\ \text{MCM}_A &= \max(h_A, h_{A_1}). \end{aligned} \tag{3.2}$$

A visual representation of the topology of C-HMCNN(h) is given in Figure 3.5. Notice that the output of C-HMCNN(h) ensures that no hierarchy violation happens, that is, that for any threshold, it cannot be the case that MCM predicts that a data point belongs to A_1 but not to A.

In the second step, in order to exploit the hierarchy constraint during training, I train C-HMCNN(h) with a novel loss function, called *max constraint loss* (MCLoss),

which is defined as

$$\text{MCLoss} = \text{MCLoss}_{A_1} + \text{MCLoss}_A,$$

where:

$$\begin{aligned} \text{MCLoss}_{A_1} &= -y_{A_1} \ln(\text{MCM}_{A_1}) - (1 - y_{A_1}) \ln(1 - \text{MCM}_{A_1}), \\ \text{MCLoss}_A &= -y_A \ln(\max(h_A, h_{A_1} y_{A_1})) - (1 - y_A) \ln(1 - \text{MCM}_A)). \end{aligned} \tag{3.3}$$

MCLoss differs from the standard binary cross-entropy loss $\mathcal{L}$:

$$\mathcal{L} = -y_{A_1} \ln(\text{MCM}_{A_1}) - (1 - y_{A_1}) \ln(1 - \text{MCM}_{A_1}) - y_A \ln(\text{MCM}_A) - (1 - y_A) \ln(1 - \text{MCM}_A),$$

iff $x \notin A_1$ ($y_{A_1} = 0$), $x \in A$ ($y_A = 1$), and $h_{A_1} > h_A$.

The following example highlights the different behavior of MCLoss compared to $\mathcal{L}$.

Example 3.2.1. Assume $h_{A_1} = 0.3$, $h_A = 0.1$, $y_{A_1} = 0$, and $y_A = 1$. Then,

$$\text{MCLoss} = \text{MCLoss}_{A_1} + \text{MCLoss}_A = -\ln(1 - h_{A_1}) - \ln(h_A)\,,$$

and the partial derivatives of MCLoss with respect to h_{A_1} and h_A are

$$\frac{\partial \text{MCLoss}}{\partial h_{A_1}} = -\frac{1}{h_{A_1} - 1} \sim 1.4 \qquad \text{and} \qquad \frac{\partial \text{MCLoss}}{\partial h_A} = -\frac{1}{h_A} = -10\,,$$

and C-HMCNN(h) rightly learns that it needs to decrease h_{A_1} and increase h_A.

On the other hand, if the standard binary cross-entropy $\mathcal{L}$ is used after MCM, then the loss is equal to:

$$\mathcal{L} = -\ln(1 - \text{MCM}_{A_1}) - \ln(\text{MCM}_A) = -\ln(1 - h_{A_1}) - \ln(h_{A_1})\,,$$

and

$$\frac{\partial \mathcal{L}}{\partial h_{A_1}} = -\frac{1}{h_{A_1} - 1} - \frac{1}{h_{A_1}} \sim -1.9 \qquad \text{and} \qquad \frac{\partial \mathcal{L}}{\partial h_A} = 0\,.$$

Hence, if C-HMCNN(h) is trained with $\mathcal{L}$, then it wrongly learns that it needs to increase h_{A_1} and keep h_A. ◁

Consider the example in Figure 3.2. To check that the proposed model behaves as expected, I implemented h as f, and trained C-HMCNN(h) with MCLoss on the same datasets and in the same way as f and g. The last two columns of Figure 3.3 show the decision boundaries of C-HMCNN(h), while the decision boundaries of h can be seen in the last two columns of Figure 3.4. C-HMCNN(h)'s decision boundaries mirror those of f^+ (resp., g^+) when $R_1 \cap R_2 = R_1$ (resp., $R_1 \cap R_2 = \emptyset$). Intuitively, as highlighted by Figure 3.4, C-HMCNN(h) is able to decide whether to learn A:

1. as a whole (top figure),
2. as the union of $A \setminus A_1$ and A_1 (middle figure), and
3. as the union of a subset of A and a subset of A_1 (bottom figure).

C-HMCNN(h) has thus learned when to exploit the prediction on the lower class A_1 to make predictions on the upper class A. In this case, this happens for:

1. 0% of the points in A_1 when $R_1 \cap R_2 = R_1$ as in the top figure,
2. 100% of the points in A_1 when $R_1 \cap R_2 = \emptyset$ as in the middle figure, and
3. 85% of the points in A_1 when R_1 and R_2 are as in the bottom figure.

3.3 General Case

Consider now an arbitrary HMC problem $(\mathcal{P}, \Pi)$ with $\mathcal{P} = (\mathcal{A}, \mathcal{X})$.

Given a class $A \in \mathcal{A}$, let $\mathcal{S}_A$ be the set of all the subclasses of A as given by Π, that is, the set of classes B such that there is a path of length ≥ 0 from B to A in the graph with an edge from A_1 to A for each constraint (3.1) in Π. Thus, for instance, if we have $A_1 \to A_2$ and $A_2 \to A_3$, $\mathcal{S}_{A_1} = \{A_1\}$, $\mathcal{S}_{A_2} = \{A_1, A_2\}$, $\mathcal{S}_{A_3} = \{A_1, A_2, A_3\}$.

Consider a data point $x \in \mathbb{R}^D$ and a model h for $\mathcal{P}$.

The basic idea is to leverage the predictions on the lower classes to make predictions on the upper classes in the hierarchy. Indeed, for a class A, it may be the case that there exists one subclass $A_1 \in \mathcal{S}_A$ with $h_{A_1} > h_A$: in this case, it is desirable to set $h_A = h_{A_1}$, that is, C-HMCNN(h) should delegate the prediction on A to A_1.

Definition 3.3.1 (Delegate)**.** *Let $(\mathcal{P}, \Pi)$ be an HMC problem. Let h be a model for $\mathcal{P}$. Let A and A_1 be two classes with $A_1 \in \mathcal{D}_A$.* C-HMCNN*(h) delegates the prediction on A to A_1 for a data point, if* C-HMCNN*(h)$_A = h_{A_1}$ and $h_{A_1} > h_A$.*

In the general case, I want C-HMCNN(h) to delegate the prediction on A to the subclass A_1 of A having maximum h_{A_1}. This is obtained by defining the output MCM_A of C-HMCNN(h) for a class A to be:

$$\text{MCM}_A = \max_{B \in \mathcal{S}_A}(h_B). \tag{3.4}$$

For each class A, the number of operations performed by MCM_A is independent from the depth of the hierarchy (as we only need to perform the max operation once), making C-HMCNN(h) a scalable model. Thanks to MCM, C-HMCNN(h) is

guaranteed to always output predictions satisfying the hierarchy constraints, as stated by the following theorem, which follows immediately from Eq. (3.4).

Theorem 3.3.2. *Let $(\mathcal{P}, \Pi)$ be an HMC problem. For any model h for $\mathcal{P}$,* C-HMCNN*(h) does not commit any hierarchy violations.*

Proof. For every constraint in Π having form (3.1), $A_1 \in \mathcal{S}_A$, and thus, according to Equation (3.4), $\mathrm{MCM}_A \geq \mathrm{MCM}_{A_1}$. As a result, C-HMCNN$(h)$ does not commit any hierarchy violations. □

As an immediate consequence, C-HMCNN(h) also does not commit any logical violations and is coherent relative to the hierarchy constraints.

Corollary 3.3.3. *Let $(\mathcal{P}, \Pi)$ be an HMC problem. For any model h for $\mathcal{P}$,* C-HMCNN*(h) does not commit any logical violations and is coherent with respect to* Π*.*

The next step is to improve performance by modifying the loss function in order to exploit the constraints. For each class A, MCLoss_A is defined as:

$$\mathrm{MCLoss}_A = -y_A \ln(\max_{B \in \mathcal{S}_A}(y_B h_B)) - (1 - y_A) \ln(1 - \mathrm{MCM}_A)\,,$$

where y_A is the ground truth class for A. The final MCLoss is then given by:

$$\mathrm{MCLoss} = \sum_{A \in \mathcal{A}} \mathrm{MCLoss}_A. \tag{3.5}$$

MCLoss has the fundamental property that the negative gradient descent algorithm behaves as expected, that is, that for each class, it moves in the "right" direction as given by the ground truth. This is formally expressed by the following theorem.

Theorem 3.3.4. *Let $(\mathcal{P}, \Pi)$ be an HMC problem. For any model h for $\mathcal{P}$ and class A, let $\frac{\partial \mathrm{MCLoss}}{\partial h_A}$ be the partial derivative of* MCLoss *with respect to h_A. For each data point, if $y_A = 0$, then $\frac{\partial \mathrm{MCLoss}}{\partial h_A} \geq 0$, and if $y_A = 1$, then $\frac{\partial \mathrm{MCLoss}}{\partial h_A} \leq 0$.*

Proof. Consider a data point and a class A.

$$\frac{\partial \mathrm{MCLoss}}{\partial h_A} = \sum_{B \in \mathcal{A}} \frac{\partial \mathrm{MCLoss}_B}{\partial h_A}.$$

Assume $y_A = 1$. For each class B such that $y_B = 0$:

$$\mathrm{MCLoss}_B = -\ln(1 - \mathrm{MCM}_B), \qquad \frac{\partial \mathrm{MCLoss}_B}{\partial h_A} = \frac{1}{1 - \mathrm{MCM}_B} \frac{\partial \mathrm{MCM}_B}{\partial h_A} = 0,$$

because MCM_B is not a function of h_A (since $y_A = 1$ and $y_B = 0$, $A \notin \mathcal{S}_B$), and hence $\frac{\partial \text{MCM}_B}{\partial h_A} = 0$. For each class B such that $y_B = 1$,

$$\text{MCLoss}_B = -\ln(\max_{C \in \mathcal{S}_B}(y_C h_C)),$$

$$\frac{\partial \text{MCLoss}_B}{\partial h_A} = -\frac{1}{\max_{C \in \mathcal{S}_B}(y_C h_C)} \frac{\partial \max_{C \in \mathcal{S}_B}(y_C h_C)}{\partial h_A} \leq 0,$$

because if $\max_{C \in \mathcal{S}_B}(y_C h_C) = h_A$, then $\frac{\partial \text{MCLoss}_B}{\partial h_A} = -\frac{1}{h_A} \leq 0$, otherwise $\frac{\partial \text{MCLoss}_B}{\partial h_A} = 0$. Since $\frac{\partial \text{MCLoss}}{\partial h_A}$ is given by the sum of quantities that are smaller or equal zero, then $\frac{\partial \text{MCLoss}}{\partial h_A} \leq 0$.

Assume $y_A = 0$. For each class B such that $y_B = 0$:

$$\text{MCLoss}_B = -\ln(1 - \text{MCM}_B), \qquad \frac{\partial \text{MCLoss}_B}{\partial h_A} = \frac{1}{1 - \text{MCM}_B} \frac{\partial \text{MCM}_B}{\partial h_A} \geq 0,$$

because if $\text{MCM}_B = h_A$, then $\frac{\partial \text{MCM}_B}{\partial h_A} = 1$, otherwise $\frac{\partial \text{MCM}_B}{\partial h_A} = 0$. For each class B such that $y_B = 1$:

$$\text{MCLoss}_B = -\ln(\max_{C \in \mathcal{S}_B}(y_C h_C)),$$

$$\frac{\partial \text{MCLoss}_B}{\partial h_A} = -\frac{1}{\max_{C \in \mathcal{S}_B}(y_C h_C)} \frac{\partial \max_{C \in \mathcal{S}_B}(y_C h_C)}{\partial h_A} = 0,$$

because $\max_{C \in \mathcal{S}_B}(y_C h_C) \neq h_A$, since $y_A = 0$. Since $\frac{\partial \text{MCLoss}}{\partial h_A}$ is given by the sum of quantities that are greater than or equal to zero, then $\frac{\partial \text{MCLoss}}{\partial h_A} \geq 0$. □

Example 3.2.1 already pointed out that the standard loss function may not behave as expected. This becomes even more apparent in the general case. Indeed, as highlighted by the following example, the more superclasses a class has, the more likely it is that C-HMCNN(h) trained with the standard binary cross-entropy loss $\mathcal{L}$ will not behave correctly.

Example 3.3.5. Consider an HMC problem with $n+1$ classes $A, A_1, \ldots, A_n$. Assume

1. $A \in \bigcap_{i=1}^{n} \mathcal{S}_{A_i}$,
2. $h_A > \max(h_{A_1}, \ldots, h_{A_n})$, and
3. $y_A = 0$ while $y_{A_1} = \cdots = y_{A_n} = 1$.

Then, the standard binary cross-entropy loss $\mathcal{L}$ is equal to:

$$\mathcal{L} = \mathcal{L}_A + \sum_{i=1}^{n} \mathcal{L}_{A_i}, \qquad \mathcal{L} = -\ln(1 - h_A) - n \ln(h_A),$$

and

$$\frac{\partial \mathcal{L}}{\partial h_A} = \frac{1}{1-h_A} - \frac{n}{h_A}.$$

Since $y_A = 0$, I would like to get $\frac{\partial \mathcal{L}_A}{\partial h_A} \geq 0$. However, this is possible only if $h_A \geq \frac{n}{n+1}$: if $n = 1$, then I need $h_A \geq 0.5$, while if $n = 10$, then I need $h_A \geq 10/11 \sim 0.91$. On the other hand, MCLoss is equal to:

$$\text{MCLoss} = \text{MCLoss}_A + \sum_{i=1}^{n} \text{MCLoss}_{A_i}, \qquad \text{MCLoss} = -\ln(1-h_A) + \sum_{i=1}^{n} \text{MCLoss}_{A_i},$$

and

$$\frac{\partial \text{MCLoss}}{\partial h_A} = \frac{1}{1-h_A}.$$

No matter the value of h_A, $\frac{\partial \text{MCLoss}_A}{\partial h_A} \geq 0$. ◁

3.4 GPU Implementation

So far, for the sake of readability, MCM_A and MCLoss_A have been defined for a specific class A. However, it is possible to compute MCM and MCLoss for all classes in parallel, leveraging GPU architectures.

Let H be an $n \times n$ matrix obtained by stacking n times the n outputs of the bottom module h of C-HMCNN(h). Let M be an $n \times n$ matrix such that, for $i, j \in \{1, \ldots, n\}$, $M_{ij} = 1$ if A_j is a subclass of A_i ($A_j \in \mathcal{S}_{A_i}$), and $M_{ij} = 0$, otherwise. Then,

$$\text{MCM} = \max(M \odot H, \text{dim} = 1)\,,$$

where $\odot$ represents the Hadamard product, and given an arbitrary $p \times q$ matrix Q, $\max(Q, \text{dim} = 1)$ returns a vector of length p whose i-th element is equal to $\max(Q_{i1}, \ldots, Q_{iq})$. For MCLoss, I can use the same mask M to modify the standard binary cross-entropy loss (BCELoss) that can be found in any available library (e.g., PyTorch). In detail, let y be the ground-truth vector, $[h_{A_1}, \ldots, h_{A_n}]$ be the output vector of h, $\bar{h} = y \odot [h_{A_1}, \ldots, h_{A_n}]$, $\bar{H}$ be the $n \times n$ matrix obtained by stacking n times the vector $\bar{h}$. Then,

$$\text{MCLoss} = \text{BCELoss}(((1-y) \odot \text{MCM}) + (y \odot \max(M \odot \bar{H}, \text{dim} = 1)), y).$$

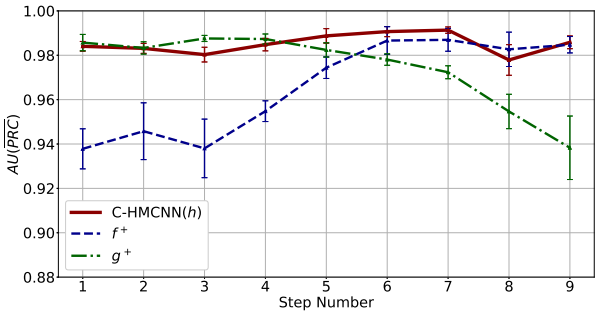


Figure 3.6: Mean $AU(\overline{PRC})$ with standard deviation of C-HMCNN(h), f^+, and g^+ for each step.

3.5 Experimental Analysis

In this section, the experimental results of C-HMCNN(h) are presented, first considering two synthetic experiments, and then considering 20 real-world datasets on which C-HMCNN(h) is compared with current state-of-the-art models for HMC problems. Finally, the ablation studies highlight the positive impact of both MCM and MCLoss on C-HMCNN(h)'s performance.[2]

The metric used to evaluate models is the area under the average precision and recall curve ($AU(\overline{PRC})$). The $AU(\overline{PRC})$ is computed as the area under the average precision recall curve, whose points $(\overline{Prec}, \overline{Rec})$ are computed as:

$$\overline{Prec} = \frac{\sum_{i=1}^{n} \mathrm{TP}_i}{\sum_{i=1}^{n} \mathrm{TP}_i + \sum_{i=1}^{n} \mathrm{FP}_i} \qquad \overline{Rec} = \frac{\sum_{i=1}^{n} \mathrm{TP}_i}{\sum_{i=1}^{n} \mathrm{TP}_i + \sum_{i=1}^{n} \mathrm{FN}_i},$$

where TP_i, FP_i, and FN_i are the number of true positives, false positives, and false negatives for class i, respectively. $AU(\overline{PRC})$ has the advantage of being independent from the threshold used to predict when a datapoint belongs to a particular class (which is often heavily application-dependent) and is the most used in the HMC literature [7, 129, 132].

3.5.1 Synthetic Experiment 1

Consider the generalization of the experiment in Section 3.2 in which R_1 starts outside R_2 (as in Figure 3.2a and as in the second row of Figure 3.3), and then R_1 moves towards the centre of R_2 (as in Figure 3.2b and as in the first row of Figure 3.3) in 9 uniform steps. The last row of Figure 3.3 corresponds to the fifth step, that is, R_1 is halfway. This experiment is meant to show how the performance of C-HMCNN(h),

[2]Link: https://github.com/EGiunchiglia/C-HMCNN/

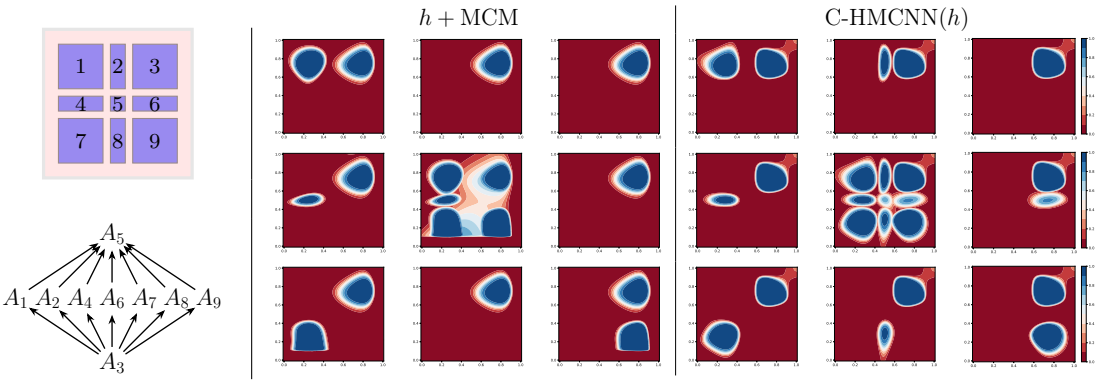


Figure 3.7: First column: rectangles disposition above and hierarchy representation below. Second column: decision boundaries of $h + \text{MCM}$ trained with $\mathcal{L}$ for each class. Last column: decision boundaries of C-HMCNN(h) for each class. In each figure showing the decision boundaries, the darker the blue (resp., red), the more confident a model is that the data points in the region belong (resp., do not belong) to the class (see the scale at the end of each row).

f^+, and g^+ defined as in Section 3.2 varies depending on the relative positions of R_1 and R_2. Here, f, g, and h were implemented and trained as in Section 3.2. For each step, I run the experiment 10 times,[3] and I plot the mean $AU(\overline{PRC})$ together with the standard deviation for C-HMCNN(h), f^+, and g^+ in Figure 3.6.

As expected, Figure 3.6 shows that f^+ performed poorly in the first three steps when $R_1 \cap R_2 = \emptyset$, it then started to perform better at step 4 when $R_1 \cap R_2 \notin \{R_1, \emptyset\}$, and it performed well from step 6 when R_1 overlaps significantly with R_2 (at least 65% of its area). Conversely, g^+ performed well on the first five steps, and its performance started decaying from step 6. C-HMCNN(h) performed well at all steps, as expected, showing robustness with respect to the relative positions of R_1 and R_2. Further, C-HMCNN(h) exhibits much more stable performances than f^+ and g^+ as highlighted by the visibly much smaller standard deviations of C-HMCNN(h).

3.5.2 Synthetic Experiment 2

In order to prove the importance of using MCLoss instead of the standard binary cross-entropy loss $\mathcal{L}$, in this experiment, I compare two models: (i) our model C-HMCNN(h), and (ii) $h + \text{MCM}$, that is h with MCM built on top and trained with $\mathcal{L}$. Consider the nine rectangles arranged as showed on the top left of Figure 3.7 named $R_1, \ldots, R_9$. Assume that

[3] All subfigures in Figure 3.3 correspond to the decision boundaries of f, g, and h in the first of the 10 runs.

1. there are 9 classes $A_1, \ldots, A_9$,

2. a data point belongs to A_i if it belongs to the ith rectangle, and

3. A_5 (resp., A_3) is an ancestor (resp., descendant) of every class, as shown in the hierarchy on the bottom left of Figure 3.7.

Thus, all points in R_3 belong to all classes, and if a data point belongs to a rectangle, then it also belongs to class A_5. The datasets consisted of 5000 (50/50 train/test split) data points sampled from a uniform distribution over $[0, 1]^2$.

Let h be a feed-forward neural network with a single hidden layer with 7 neurons. I trained both h + MCM and C-HMCNN(h) for 20k epochs using Adam optimization with learning rate 10^{-2} ($\beta_1 = 0.9, \beta_2 = 0.999$). As expected, the average $AU(\overline{PRC})$ (and standard deviation) over 10 runs for h + MCM trained with $\mathcal{L}$ is 0.938 (0.038), while h+MCM trained with MCLoss (C-HMCNN(h)) is 0.974 (0.007). Notice that not only h + MCM performs worse, but also, due to the convergence to bad local optima, the standard deviation obtained with h + MCM is 5 times higher than the one of C-HMCNN(h): the (min, median, max) $AU(\overline{PRC})$ for h+MCM are $(0.871, 0.945, 0.990)$, while for C-HMCNN(h) are $(0.964, 0.975, 0.990)$. The difference between C-HMCNN(h) and h + MCM in performance is further highlighted in Figure 3.7, which shows the decision boundaries of the 6th best performing networks.[4] The figure points out how mistakes given by wrong supervisions in lower levels of the hierarchy (see decision boundaries for A_2, A_6, and A_8) might have dramatic consequences in upper levels of the hierarchy (see decision boundaries for A_5). Indeed, focusing on the decision boundaries for A_2, A_6 and A_8, it is possible to see that the neural network is learning to set as positive only those datapoints that belong to A_3, while the datapoints that solely belong to each of these classes (and not to A_3) are confidently classified as negative. This is probably due to the phenomenon of the wrong supervisions explained in Section 3.3. This in turn impacts negatively the performance of the model on the class A_5 because now the model cannot delegate the predictions for those datapoints to A_2, A_6 and A_8 and has to learn the right supervision on its own.

3.5.3 Comparison with the State-of-the-Art

I tested C-HMCNN(h) on 20 real-world datasets commonly used to compare HMC systems (see, e.g., [7, 87, 129, 132]): 16 are functional genomics datasets [19], 2 contain medical images [33], 1 contains images of microalgae [34], and 1 is a text categorization

[4] I picked the 6th best performing networks due to the high variance of the results h + MCM.

Taxonomy	Dataset	D	n	Train	Val	Test
FunCat (FUN)	Cellcycle	77	499	1625	848	1281
FunCat (FUN)	Derisi	63	499	1605	842	1272
FunCat (FUN)	Eisen	79	461	1055	529	835
FunCat (FUN)	Expr	551	499	1636	849	1288
FunCat (FUN)	Gasch1	173	499	1631	846	1281
FunCat (FUN)	Gasch2	52	499	1636	849	1288
FunCat (FUN)	Seq	478	499	1692	876	1332
FunCat (FUN)	Spo	80	499	1597	837	1263
Gene Ontology (GO)	Cellcycle	77	4122	1625	848	1281
Gene Ontology (GO)	Derisi	63	4116	1605	842	1272
Gene Ontology (GO)	Eisen	79	3570	1055	528	835
Gene Ontology (GO)	Expr	551	4128	1636	849	1288
Gene Ontology (GO)	Gasch1	173	4122	1631	846	1281
Gene Ontology (GO)	Gasch2	52	4128	1636	849	1288
Gene Ontology (GO)	Seq	478	4130	1692	876	1332
Gene Ontology (GO)	Spo	80	4166	1597	837	1263
Tree	Diatoms	371	398	1085	464	1054
Tree	Enron	1000	56	692	296	660
Tree	Imclef07a	80	96	7000	3000	1006
Tree	Imclef07d	80	46	7000	3000	1006

Table 3.1: Summary of the 20 real-world datasets. Number of features (D), number of classes (n), and number of data points for each dataset split.

dataset [65].[5] The characteristics of these datasets are summarized in Table 3.1. As we are in the HMC field, each of these datasets comes with the taxonomy according to which the classes are ordered. In Table 3.2 we report for each dataset the number of classes present in each level of the taxonomy. The considered datasets are particularly challenging, because their number of training samples is rather limited, and they have a large variation, both in the number of features (from 52 to 1000) and in the number of classes (from 56 to 4130). I applied the same preprocessing to all the datasets. All the categorical features were transformed using one-hot encoding. The missing values were replaced by their mean in the case of numeric features and by a vector of all zeros in the case of categorical ones. All the features were standardized.

In order to prove the effectiveness of the proposed solution, I decided to pick a very simple model for h. Thus, I built h as a feedforward neural network with two hidden layers and ReLU non-linearity. To prove the robustness of C-HMCNN(h), I

[5]Links: https://dtai.cs.kuleuven.be/clus/hmcdatasets and http://kt.ijs.si/DragiKocev/PhD/resources

	Hierarchical Level												
Dataset	1	2	3	4	5	6	7	8	9	10	11	12	13
CELLCYCLE FUN	18	80	178	142	77	4	-	-	-	-	-	-	-
DERISI FUN	18	80	178	142	77	4	-	-	-	-	-	-	-
EISEN FUN	18	76	165	131	67	4	-	-	-	-	-	-	-
EXPR FUN	18	80	178	142	77	4	-	-	-	-	-	-	-
GASCH1 FUN	18	80	178	142	77	4	-	-	-	-	-	-	-
GASCH2 FUN	18	80	178	142	77	4	-	-	-	-	-	-	-
SEQ FUN	18	80	178	142	77	4	-	-	-	-	-	-	-
SPO FUN	18	80	178	142	77	4	-	-	-	-	-	-	-
CELLCYCLE FUN	33	155	394	597	929	779	631	335	171	63	21	5	9
DERISI FUN	33	155	394	596	927	778	630	334	171	63	21	5	9
EISEN FUN	33	149	360	524	786	679	539	271	141	55	19	5	9
EXPR FUN	33	155	394	599	932	780	631	335	171	63	21	5	9
GASCH1 FUN	33	155	394	597	929	779	631	335	171	63	21	5	9
GASCH2 FUN	33	155	394	599	932	780	631	335	171	63	21	5	9
SEQ FUN	33	155	394	599	932	780	633	335	171	63	21	5	9
SPO FUN	33	155	394	596	927	778	630	334	171	63	21	5	9
DIATOMS	85	310	3	-	-	-	-	-	-	-	-	-	-
ENRON	3	40	13	-	-	-	-	-	-	-	-	-	-
IMCLEF07A	8	25	63	-	-	-	-	-	-	-	-	-	-
IMCLEF07D	4	16	26	-	-	-	-	-	-	-	-	-	-

Table 3.2: Number of classes per level in the hierarchy of each dataset (excluding the "root" level).

kept all the hyperparameters fixed except the hidden dimension and the learning rate used for each dataset, which are given in Appendix A.1 and were optimized over the validation sets. In all experiments, the loss was minimized using Adam optimizer with weight decay 10^{-5}, and patience 20 ($\beta_1 = 0.9$, $\beta_2 = 0.999$). The dropout rate was set to 70% and the batch size to 4. As in the work by [132], I retrained C-HMCNN(h) on both training and validation data for the same number of epochs, as the early stopping procedure determined was optimal in the first pass.

In order to evaluate C-HMCNN(h), I compare its performance with the ones of the state-of-the-art models: Clus-Ens [103], HMC-LMLP [15], HMCN-R and HMCN-F [132]. A description of each of these models can be found in Chapter 2. For each dataset, I run C-HMCNN(h), Clus-Ens [103], and HMC-LMLP [15] 10 times, and the average $AU(\overline{PRC})$ is reported in first three columns of Table 3.3.[6] For

[6]As suggested in https://dtai.cs.kuleuven.be/clus/hmcdatasets/, I do not evaluate the performance

Dataset	C-HMCNN(h)	HMC-LMLP	Clus-Ens	Random	HMCN-R	HMCN-F
Cellcycle FUN	**0.255**	0.207	0.227	0.018	0.247	0.252
Derisi FUN	**0.195**	0.182	0.187	0.018	0.189	0.193
Eisen FUN	**0.306**	0.245	0.286	0.020	0.298	0.298
Expr FUN	**0.302**	0.242	0.271	0.018	0.300	0.301
Gasch1 FUN	**0.286**	0.235	0.267	0.018	0.283	0.284
Gasch2 FUN	**0.258**	0.211	0.231	0.018	0.249	0.254
Seq FUN	**0.292**	0.236	0.284	0.017	0.290	0.291
Spo FUN	**0.215**	0.186	0.211	0.018	0.210	0.211
Cellcycle GO	**0.413**	0.361	0.387	0.008	0.395	0.400
Derisi GO	**0.370**	0.343	0.361	0.008	0.368	0.369
Eisen GO	**0.455**	0.406	0.433	0.010	0.435	0.440
Expr GO	0.447	0.373	0.422	0.008	0.450	**0.452**
Gasch1 GO	**0.436**	0.380	0.415	0.008	0.416	0.428
Gasch2 GO	0.414	0.371	0.395	0.008	0.463	**0.465**
Seq GO	0.446	0.370	0.438	0.008	0.443	**0.447**
Spo GO	**0.382**	0.342	0.371	0.008	0.375	0.376
Diatoms	**0.758**	-	0.501	0.005	0.514	0.530
Enron	**0.756**	-	0.696	0.010	0.710	0.724
Imclef07a	**0.956**	-	0.803	0.031	0.904	0.950
Imclef07d	**0.927**	-	0.881	0.065	0.897	0.920
Avg Rank	1.25	5.00	3.93	6.00	2.93	1.90

Table 3.3: Comparison of C-HMCNN(h) with the other state-of-the-art models. The performance of each system is measured as the $AU(\overline{PRC})$ obtained on the test set. The best results are in bold.

simplicity, the standard deviations are omitted, which for C-HMCNN(h) are in the range $[0.5 \times 10^{-3}, 2.6 \times 10^{-3}]$, proving that it is a very stable model. As reported by [87], Clus-Ens and HMC-LMLP are the current state-of-the-art models with publicly available code. These models were run on CPU (as there is no GPU implementation publicly available) with the suggested configuration settings on each dataset.[7] In the fourth column, the performance of a random performance classifier is reported. As described by [102], I computed the $AU(\overline{PRC})$ of such a random baseline by dividing the number of positive examples in the test set by the multiplication of the number of datapoints in the test set and the number of classes. In the last two columns, the results of HMCN-R and HMCN-F are reported. These results are directly taken from [132], since their code is not publicly available. The results of both systems are

of the models on the classes at the root nodes of the hierarchies.

[7] I also ran the code by [81]. However, I obtained very different results from the ones reported in the paper. Similar negative results are also reported by [87].

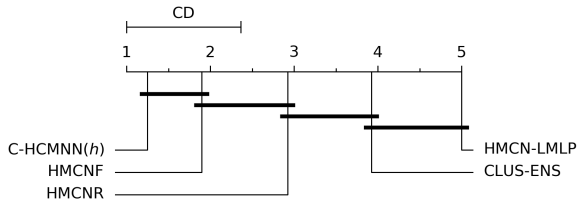


Figure 3.8: Critical diagram for the Nemenyi's statistical test.

reported, because, while HMCN-R has worse results than HMCN-F, the amount of parameters of the latter grows with the number of hierarchical levels. As a consequence, HMCN-R is much lighter in terms of total amount of parameters, and the authors advise that for very large hierarchies, HMCN-R is probably a better choice than HMCN-F considering the trade-off performance vs. computational cost [132]. Note that, as already stated, the number of parameters of C-HMCNN(h) is independent from the number of hierarchical levels.

As reported in Table 3.3, C-HMCNN(h) has the greatest number of wins (it has the best performance on all datasets but 3) and best average ranking (1.25). I also verified the statistical significance of the results following the instructions given in the work by [27].[8] I first executed the Friedman test, obtaining p-value 4.26×10^{-15}. I then performed the post-hoc Nemenyi test, and the resulting critical diagram is shown in Figure 3.8, where the group of methods that do not differ significantly (significance level 0.05) are connected through a horizontal line. The Nemenyi test is powerful enough to conclude that there is a statistical significant difference between the performance of C-HMCNN(h) and all the other models but HMCN-F. Hence, as advised by [27] and [6], I compared C-HMCNN(h) and HMCN-F using the Wilcoxon test. This test, contrarily to the Friedman test and the Nemenyi test, takes into account not only the ranking, but also the differences in performance of the two models. The Wilcoxon test shows that there is a statistical significant difference between the performance of C-HMCNN(h) and HMCN-F with p-value of 6.01×10^{-3}.

Furthermore, to study how the models perform once a threshold is fixed, I compared C-HMCNN(h), Clus-HMC, and HMC-LMLP in terms of F1-score. For each model and dataset, I picked the threshold that maximizes the F1-score. The results are reported in Table 3.4, and show that C-HMCNN(h) is able to outperform HMC-LMLP and Clus-HMC on all datasets. In order to test the statistical significance of the results, I performed the Wilcoxon test, obtaining the p-value 3.1×10^{-5} when comparing C-

[8]For such test, I exclude the baseline model, as its results significantly worse than all the others.

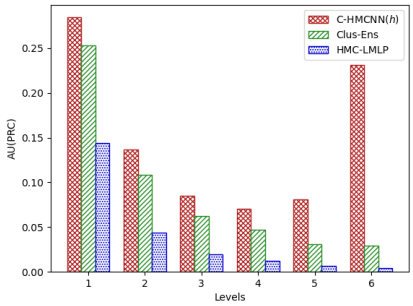
Dataset	C-HMCNN(h)	HMC-LMLP	Clus-Ens
Cellcycle FUN	**0.373**	0.251	0.292
Derisi FUN	**0.337**	0.251	0.265
Eisen FUN	**0.402**	0.262	0.336
Expr FUN	**0.401**	0.249	0.278
Gasch1 FUN	**0.392**	0.253	0.278
Gasch2 FUN	**0.377**	0.250	0.293
Seq FUN	**0.404**	0.242	0.330
Spo FUN	**0.346**	0.248	0.276
Cellcycle GO	**0.507**	0.396	0.420
Derisi GO	**0.481**	0.391	0.405
Eisen GO	**0.529**	0.410	0.422
Expr GO	**0.485**	0.393	0.416
Gasch1 GO	**0.523**	0.396	0.418
Gasch2 GO	**0.510**	0.395	0.417
Seq GO	**0.499**	0.374	0.425
Spo GO	**0.489**	0.391	0.418
Diatoms	**0.819**	-	0.502
Enron	**0.733**	-	0.636
Imclef07a	**0.924**	-	0.727
Imclef07d	**0.887**	-	0.804
Average Ranking	1.0	3.0	2.0

Table 3.4: Comparison of C-HMCNN(h) with the other state-of-the-art models. The performance of each system is measured as the F1-score obtained on the test set. The best results are in bold.

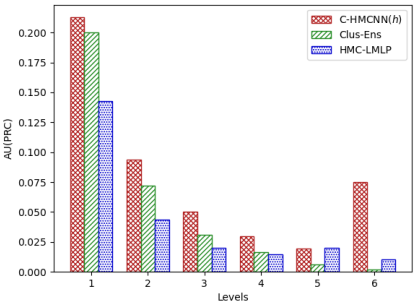
HMCNN(h) and HMC-LMLP, and p-value 1.9×10^{-6} when comparing C-HMCNN(h) and Clus-HMC.

To better understand why C-HMCNN(h) is able to outperform the other models, Figure 3.9 shows the average $AU(\overline{PRC})$ per hierarchy level achieved by C-HMCNN(h), Clus-HMC, and HMC-LMLP in each of the Funcat datasets.[9] The figure shows that C-HMCNN(h) is able to get better results than Clus-HMC and HMC-LMLP at all levels of the hierarchy for all the datasets (with the exception of levels 4 and 5 for the dataset Seq Fun, where Clus-HMC manages to outperform C-HMCNN(h)). Interestingly, the biggest gaps in performance are reported at higher levels of the hierarchy. This is not surprising given that MCLoss penalizes errors more at the higher levels. As an example, if at training time C-HMCNN(h) delegates the prediction

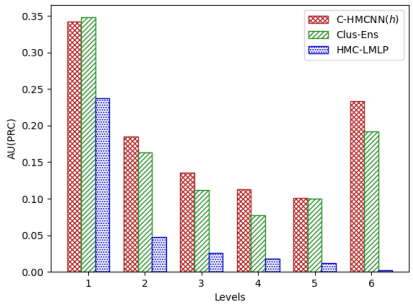
[9]In Figure 3.9, the levels start at 1 because the root is not taken into account for evaluation. I could not measure the performance of HMCN-R and HMCN-F as the code is not available.



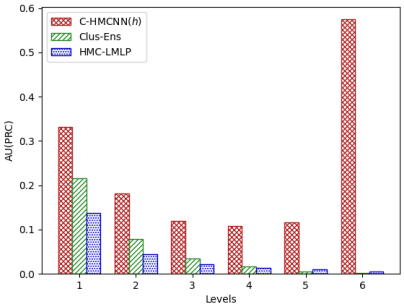
(a) Cellcycle FUN



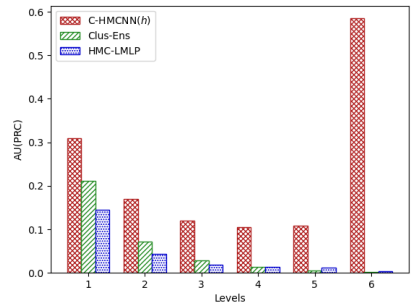
(b) Derisi FUN



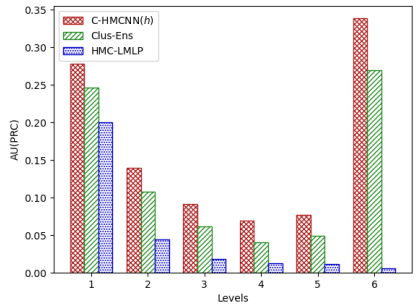
(c) Eisen FUN



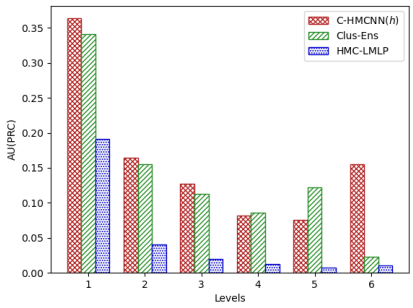
(d) Expr FUN



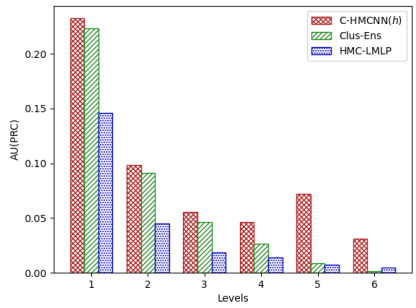
(e) Gasch1 FUN



(f) Gasch2 FUN



(g) Seq FUN



(h) Spo FUN

Figure 3.9: Average $AU(\overline{PRC})$ per hierarchy level.

Dataset	h^+ $AU(\overline{PRC})$	Epochs	h+MCM $AU(\overline{PRC})$	Epochs	C-HMCNN(h) $AU(\overline{PRC})$	Epochs
Cellcycle	0.240	107	0.238	108	**0.255**	106
Derisi	0.190	64	0.188	66	**0.195**	67
Eisen	0.290	112	0.286	107	**0.306**	110
Expr	0.272	39	0.267	19	**0.302**	20
Gasch1	0.265	41	0.262	42	**0.286**	38
Gasch2	0.244	128	0.242	132	**0.258**	131
Seq	0.249	13	0.252	13	**0.292**	13
Spo	0.201	108	0.202	117	**0.215**	115
Average Ranking	2.94		2.06		1.00	

Table 3.5: Impact of MCM and MCM+MCLoss on the performance measured as $AU(\overline{PRC})$ and on the total number of epochs necessary for the networks to converge. The reported results are measured on the validation set of the Funcat datasets.

of class A belonging to level 1 to class B belonging to level 6, then the error made by h_B will be counted 6 times in the loss, while the error made by h_A will not be counted at all. Indeed, if A delegates to B, this means that all the subclasses of A and superclasses of B in the path from B to A are delegating their prediction to B. Thus, a mistake made by h_B will lead to a mistake on the prediction of 6 different classes and this is why its error is counted 6 times in the loss. Thus, MCLoss could also be seen as a rebalancing method for the labels belonging to higher level of the hierarchy. A similar mechanism is completely absent in both HMC-LMLP and Clus-Ens (as seen in Chapter 2 Section 2.2), and this is why their performance suffers more at higher level of the hierarchy.

3.5.4 Ablation Studies

To analyze the impact of both MCM and MCLoss, I compared the performance of C-HMCNN(h) on the FunCat datasets against the performance of:

- h with MCM applied as a post-processing step at inference time (to make the predictions compliant with the constraints as required by the HMC task), and
- h with MCM applied during training time (i.e., our model trained with the standard binary cross-entropy loss instead of the MCLoss).

Recalling the notation used for the basic case, I call the first model h^+, while the second model is called h+MCM to clarify that MCM is applied at training time, but it is not trained with MCLoss.

As it can be seen in Table 3.5, C-HMCNN(h), thanks to the exploitation of both MCM and MCLoss, always outperforms h^+ and h+MCM on all datasets, thus proving the importance of using the proposed layer and loss together. In order to check the statistical significance of the results, I performed the Wilcoxon test twice: once to compare the performance of C-HMCNN(h) against h^+, and once to compare the the performance C-HMCNN(h) against h+MCM. Both the tests returned p-value < 0.05. Further, it is interesting to note that $h + \text{MCM}$ often performs worse than h^+. This was expected, because, as we already showed in Section 3.5.2, using the standard binary cross-entropy loss might cause wrong supervisions, and ultimately convergence to bad local optima.

Finally, in Table 3.5, I also report after how many epochs the algorithm stopped training in average. As it can be seen, C-HMCNN(h), h+MCM and h^+ always require approximately the same number of epochs.

3.6 Overview of the Results

In this chapter, a new model for HMC problems—called C-HMCNN(h)—was presented. Throughout the chapter, we have seen how C-HMCNN(h) is able to (i) leverage the hierarchical information to learn when to delegate the prediction on a superclass to one of its subclasses, (ii) produce predictions coherent by construction, and (iii) outperform current state-of-the-art models on 20 commonly used real-world HMC benchmarks. Further, C-HMCNN(h) has demonstrated to possess other desirable properties like: (i) its number of parameters does not depend on the number of hierarchical levels, and (ii) it can be easily implemented on GPUs using standard libraries. I have thus demonstrated that, given a MC problem with a set of hierarchical constraints, it is possible to build neural networks able to: (i) guarantee that their outputs will always satisfy the constraints, and (ii) earn from the background knowledge specified in the constraints to get better performance.

Chapter 4

Normal Logic Constraints

In this chapter, I extend the language used to express the hierarchical constraints to allow for the specification of more complex logical relations among classes. Indeed, the language for expressing hierarchical constraints is very limited. As an example, consider again the task proposed in [33], (where a radiological image has to be annotated with an IRMA code specifying, among others, the biological system examined) then hierarchical constraints are not expressive enough to model the fact that if a medical image contains the abdomen but neither the middle nor the upper abdomen, then it contains the lower abdomen.

Thus, borrowing concepts from the area of logic programming, I consider general constraints expressed as propositional normal rules [74], that is, expressions of the form (1.2). Such constraints generalize hierarchical constraints (corresponding to the case $n = k = 1$) and naturally arise in many application domains like healthcare. With such an extension, it is now possible to write:

$$abdomen, \neg middleAbdomen, \neg upperAbdomen \rightarrow lowerAbdomen,$$

capturing the above informally stated constraint. I call MC problems with a set of constraints in such an extended syntax *logically constrained multi-label classification* (LCMC) problems.

In this chapter, I thus propose a novel model called *coherent-by-construction network* (CCN(h)), which is the first model able to deal with MC problems with such expressive constraints on the classes. I prove that, given a set $\mathcal{H}$ of initial predictions made by an underlying model h and a set of constraints with stratified negation [2], CCN(h) is able to compute in linear time in the number of constraints the unique minimal set of classes $\mathcal{M}$ that

1. extends $\mathcal{H}$, that is, such that $\mathcal{H} \subseteq \mathcal{M}$, and

2. is coherent with (satisfies) the constraints, that is, such that, given (1.2), $A \in \mathcal{M}$ whenever $\{A_1, \ldots, A_k\} \subseteq \mathcal{M}$ and $\{A_{k+1}, \ldots, A_n\} \cap \mathcal{M} = \emptyset$.

Further, I show that (i) CCN(h) is an extension of C-HMCNN(h), (ii) it can be implemented on GPUs using standard libraries (like C-HMCNN(h)), and (iii) it outperforms the current state-of-the-art models on 18 real-world and publicly available LCMC datasets.

The rest of this chapter is organized as follows. In Section 4.1, all the necessary preliminary concepts are introduced. Then, in Section 4.2, the core ideas behind CCN(h) are illustrated on a simple LCMC problem with just three classes, followed by the presentation of the general solution 4.3. How to implement the proposed model on a GPU is discussed in Section 4.4, while the experimental results are presented in Section 4.5. I conclude with an overview of the results presented in this chapter in Section 4.6.

4.1 Preliminaries

Borrowing notation and concepts from the area of logic programming, I defined *logically constrained multi-label classification* (LCMC) problems as MC problems with a finite set Π of *constraints* or *(normal) rules* r having the form:

$$A_1, \ldots, A_k, \neg A_{k+1}, \ldots, \neg A_n \rightarrow A, \qquad (0 \leq k \leq n), \tag{4.1}$$

where $A, A_1, \ldots, A_n$ are classes. I also assume, w.l.o.g., that $A_i \neq A_j$ for $1 \leq i < j \leq k$ and for $k+1 \leq i < j \leq n$. I call $\mathit{head}(r) = A$ the *head* of r, and $\mathit{body}(r) = \mathit{body}^+(r) \cup \mathit{body}^-(r)$ the *body* of r, where $\mathit{body}^+(r) = \{A_1, \ldots, A_k\}$ and $\mathit{body}^-(r) = \{\neg A_{k+1}, \ldots, \neg A_n\}$. r is said to be *definite* if $n = k$.

Constraint (4.1) imposes that for each data point x and model m, if m predicts the classes $A_1, \ldots, A_k$ and not $A_{k+1}, \ldots, A_n$, then m must also predict A. Given this logical interpretation, it is thus possible to define the concepts of logical violation and coherency, which generalize the corresponding definitions given for the hierarchical case.

Definition 4.1.1. *Let $(\mathcal{P}, \Pi)$ be an LCMC problem. Let m be a model for $\mathcal{P}$. If for a data point and a constraint $r \in \Pi$ of the form (4.1), m predicts $A_1, \ldots, A_k$ and not $A_{k+1}, \ldots, A_n, A$, then m commits a logical violation with respect to r. If m commits no logical violations, then m is coherent with respect to Π.*

The above definition allows to determine whether any model m is coherent with respect to the given constraints. However, the goal is to go beyond coherency and generalize what was done in the HMC setting: whenever convenient, exploit the constraints to compute a value for the classes in the head to ensure coherency and improve performance.

For ease of presentation, assume to have a single constraint r of the form (4.1).

In the special case where r is definite ($n = k$), it is possible to associate with the head A a value that is at least the smallest value associated with the classes in the body, that is, it is possible to set

$$m_A = \min(m_{A_1}, \ldots, m_{A_k}).$$

In this case, the constraint r is always satisfied for any threshold θ. This corresponds to interpreting (4.1) according to the Gödel t-norm T_G [83], which is the only function $T : [0,1]^2 \to [0,1]$ that, for every $a, b, c \in [0,1]$, satisfies the following properties (common to all t-norms):

$$\begin{aligned} T(a,b) &= T(b,a), & T(a,1) &= a, \\ T(a,T(b,c)) &= T(T(a,b),c), & T(a,0) &= 0, \\ & a \leq b \to T(a,c) \leq T(b,c), \end{aligned}$$

and also the following (*idempotency*, characterizing T_G):

$$T(a,a) = a.$$

If r is not definite ($n > k$), given $m_{A_{k+1}}, \ldots, m_{A_n}$, then I need to compute values $\overline{m}_{A_{k+1}}, \ldots, \overline{m}_{A_n}$ such that, for each class $A_i \in \{A_{k+1}, \ldots, A_n\}$ and threshold θ,

1. $\overline{m}_{A_i} = 1$ when $m_{A_i} = 0$, and $\overline{m}_{A_i} = 0$ when $m_{A_i} = 1$,
2. $\overline{m}_{A_i}$ is strictly decreasing and continuous (small changes to the value of m_{A_i} should correspond to small changes in the value of $\overline{m}_{A_i}$), and
3. $\overline{m}_{A_i} = \theta$ when $m_{A_i} = \theta$.

The first two conditions say that the function $\overline{v}$ of $v \in [0,1]$ is a *strict negation* [83],[1] and, together with the third entail

1. if $m_{A_i} > \theta$, then $\overline{m}_{A_i} < \theta$,

[1] A negation is non-strict if it is either non-strictly decreasing or non-continuous. An example of a non-strict negation is the residual negation in the Gödel t-norm according to which I would have $\overline{m}_A = 1$ if $m_A = 0$, and $\overline{m}_A = 0$, otherwise.

2. if $m_{A_i} < \theta$, then $\overline{m}_{A_i} > \theta$.

For any threshold θ there are infinitely many functions $\overline{v}$ satisfying such requirements. A simple solution is to require $\overline{v}$ to be piecewise linear with two segments joining when $\overline{v} = v = \theta$, in which case,

1. $\overline{v}$ is a *strong negation* [83], since $\overline{\overline{v}} = v$, and

2. if $\theta = 0.5$, I obtain $\overline{v} = 1 - v$, that is the *standard negation* in fuzzy logics.

For simplicity, from here on, I assume to have the standard negation, that is, to fix the threshold θ to 0.5 and $\overline{v} = 1 - v$, for each $v \in [0, 1]$. All the definitions and results generalize to the case in which I have an arbitrary strict negation with $\overline{\theta} = \theta$.

Given the above, I can now introduce the concept of constraint violation, generalizing the corresponding definition of hierarchy violation.

Definition 4.1.2. *Let $(\mathcal{P}, \Pi)$ be an LCMC problem. Let m be a model for $\mathcal{P}$. If for a data point and for a constraint (4.1) in Π, m does not satisfy*

$$\min(m_{A_1}, \ldots, m_{A_k}, \overline{m}_{A_{k+1}}, \ldots, \overline{m}_{A_n}) \leq m_A, \tag{4.2}$$

then m commits a constraint violation.

The following theorem easily follows from the previous two definitions.

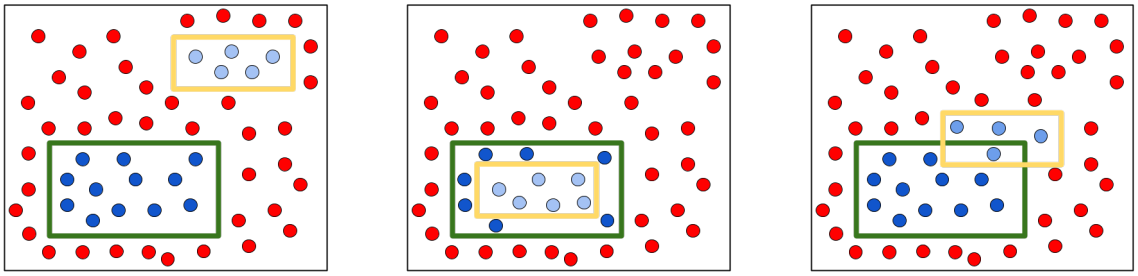
Theorem 4.1.3. *Let $(\mathcal{P}, \Pi)$ be an LCMC problem. Let m be a model for $\mathcal{P}$. If m does not commit constraint violations, then m is coherent with respect to Π.*

Proof. Since m does not commit constraint violations, for each constraint in Π having form (4.1), m satisfies (4.2). Given the threshold θ, if $m_{A_1}, \ldots, m_{A_k}$ (resp. $m_{A_{k+1}}, \ldots, m_{A_n}$) are greater (resp. smaller) than θ, then:

1. according to the definition of negation, $m_{A_{k+1}}, \ldots, m_{A_n}$ are greater than θ, and

2. according to the definition of constraint violation, m_A is also greater than θ.

This entails that m commits no logical violations with respect to any constraint in Π, and that it is coherent with respect to Π. □



(a) (b) (c)

Figure 4.1: Visualization of the basic problem. The yellow rectangle is R_1, and the green rectangle is R_2. The blue datapoints are associated to the class A_2, the light blue datapoints are associated to both the class A_1, and the red datapoints are associated with no classes. Both the blue and light blue datapoints are associated to the class A.

4.2 Basic Case

The main ideas behind our model CCN(h) are now presented through a simple LCMC problem. Assume to have an MC problem with three classes A, A_1, and A_2, and that it is known that A_1 and A_2 are subsets of A, and that A_2 includes the set of data points belonging to A and not to A_1. Then, $A_1 \cup A_2 \subseteq A$ can be imposed with the constraints

$$A_1 \rightarrow A; \qquad A_2 \rightarrow A, \tag{4.3}$$

having the form (1.1), while $A \setminus A_1 \subseteq A_2$ can be expressed as

$$A, \neg A_1 \rightarrow A_2, \tag{4.4}$$

which imposes, to any model m that, for each $x \in \mathbb{R}^D$, if m predicts A and not A_1, then m must also predict A_2.

The final goal is to develop a method that is able to leverage standard neural network approaches for MC problems, while exploiting all the above constraints in order to produce predictions that are guaranteed to satisfy the constraints while improving performance. The basic idea is thus to extend the method presented for HMC problems.

To understand how the three constraints can be exploited to improve performance, assume that $D = 2$, and consider the yellow (R_1) and green (R_2) rectangles in Figure 4.1. Assume that $A = R_1 \cup R_2$, $A_1 = R_1$, and $A_2 = R_2 \setminus R_1$. Let f be a neural network with one output for each class to be learned. Intuitively, when R_1 and R_2 are as in the first row of Figure 4.1a, it is expected to be more difficult for f to learn A than to learn A_1 and A_2. Hence, I would like to exploit the information coming from the

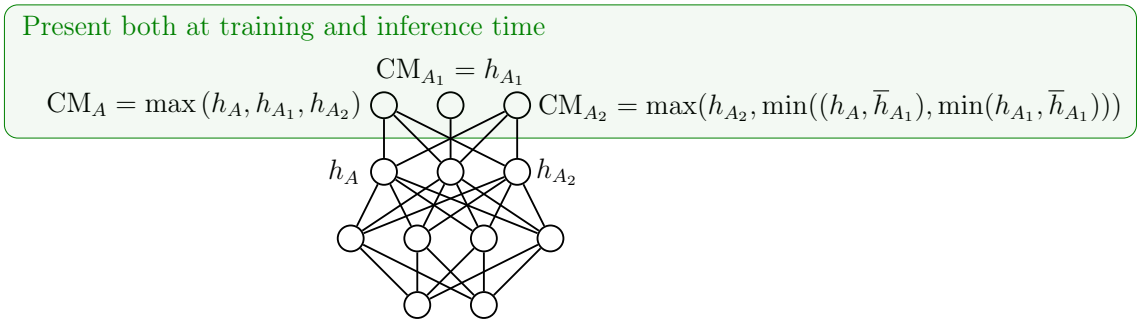


Figure 4.2: Visual representation of CCN(h) for the basic case.

constraints in (4.3) to learn A, given A_1 and A_2. On the other hand, when the two rectangles are arranged as in Figure 4.1b, it is expected to be more difficult for f to learn A_2 than A and A_1. In this case, I would like to exploit the information coming from the constraint (4.4) to learn A_2, given A and A_1. Finally, when R_1 and R_2 are arranged as in Figure 4.1c, learning both A and A_2 will be difficult, and hence I would like to be able to exploit all the constraints in (4.3) and (4.4) to improve performance.

As previously done with C-HMCNN(h), I can achieve such goal in two steps. In the first step, I build a new neural network consisting of two modules: (i) a bottom module h, which can be any neural network with one output in $[0, 1]$ for A, A_1, and A_2, respectively, and (ii) an upper *constraint module* (CM), that takes as input the output of the bottom module and imposes the constraints. I call the obtained neural network *coherent-by-construction network* (CCN(h)).

Consider a data point x. Let h_A, h_{A_1}, and h_{A_2} be the outputs of h for the classes A, A_1, and A_2 respectively. Let y_A, y_{A_1}, and y_{A_2} be the ground truth for the classes A, A_1, and A_2, respectively. Let CM_A, CM_{A_1}, and CM_{A_2} be the outputs of CM (which are the outputs of CCN(h)).

I want CCN(h) to extend the set of classes associated with x by the bottom module h, exploiting, and thus satisfying, the constraints. This is obtained by defining CM_A, CM_{A_1}, and CM_{A_2} to be the smallest values such that

$$\begin{aligned} \mathrm{CM}_A &= \max(h_A, \mathrm{CM}_{A_1}, \mathrm{CM}_{A_2}), \\ \mathrm{CM}_{A_1} &= h_{A_1}, \\ \mathrm{CM}_{A_2} &= \max(h_{A_2}, \min(\mathrm{CM}_A, \overline{\mathrm{CM}}_{A_1})). \end{aligned} \tag{4.5}$$

Indeed, the first equation ensures that (i) x will be associated with the class A whenever h already predicts it, and that (ii) CM_{A_1}, $\mathrm{CM}_{A_2} \leq \mathrm{CM}_A$; thus guaranteeing that (4.3) is satisfied. The other equations have a similar reading. Depending on the values of h_A, h_{A_1}, and h_{A_2}, (4.5) may admit more than one solution, but I later show (see

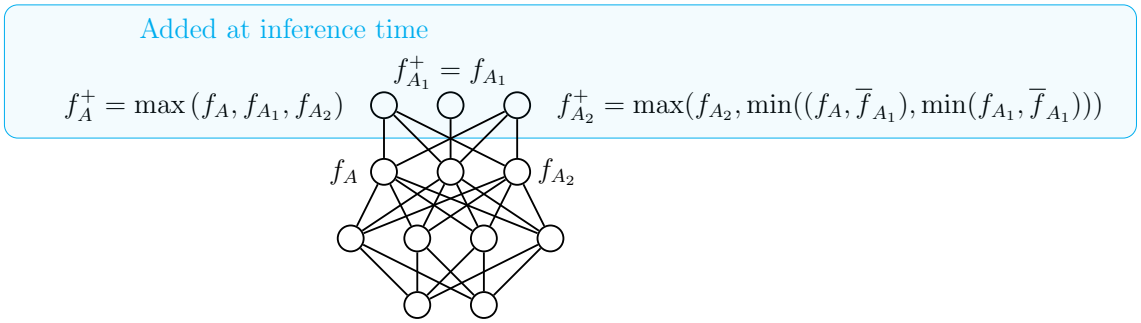


Figure 4.3: Visual representation of f^+ for the basic case.

Example 4.3.12 and Theorem 4.3.13) that none of them has a value for CM_A, CM_{A_1}, and CM_{A_2} smaller than the one defined by

$$\begin{aligned}
\mathrm{CM}_A &= \max(h_A, h_{A_1}, h_{A_2}),\\
\mathrm{CM}_{A_1} &= h_{A_1},\\
\mathrm{CM}_{A_2} &= \max(h_{A_2}, \min(h_A, \overline{h}_{A_1}), \min(h_{A_1}, \overline{h}_{A_1})),
\end{aligned}$$

which I define to be the outputs of CM. A visual representation of the topology of $\mathrm{CCN}(h)$ is given in Figure 4.2.

In the second step, to effectively exploit the constraints during training, $\mathrm{CCN}(h)$ is trained with a new loss function, called *constraint loss* (CLoss), which has two goals:

1. the first one is to give each class the correct supervision (e.g., if $y_A = 1$, then the goal to teach h to increase h_A and not to decrease it), and

2. the second goal is, given a constraint, to teach h to rely on the prediction for the classes in the body to make prediction for the class in the head only when the body is satisfied (e.g., for (4.4), when $y_A = 1$ and $y_{A_1} = 0$).

To achieve the above goals, CLoss is defined as $\mathrm{CLoss} = \mathrm{CLoss}_A + \mathrm{CLoss}_{A_1} + \mathrm{CLoss}_{A_2}$, where:

$$\begin{aligned}
\mathrm{CLoss}_A =& - y_A \ln(\max(h_A, h_{A_1} y_{A_1}, h_{A_2} y_{A_2})) - (1 - y_A) \ln(\overline{\mathrm{CM}}_A),\\
\mathrm{CLoss}_{A_1} =& - y_{A_1} \ln(\mathrm{CM}_{A_1}) - (1 - y_{A_1}) \ln(\overline{\mathrm{CM}}_{A_1}),\\
\mathrm{CLoss}_{A_2} =& - y_{A_2} \ln\big(\max\big(h_{A_2}, \min(h_A y_A, \overline{h}_{A_1}(1 - y_{A_1})), \min(h_{A_1} y_{A_1}, \overline{h}_{A_1}(1 - y_{A_1}))\big)\big)\\
& - (1 - y_{A_2}) \ln\big(1 - \max\big(h_{A_2}, \min(h_A(1 - y_A) + y_A, \overline{h}_{A_1} y_{A_1} + (1 - y_{A_1})),\\
& \min(h_{A_1}(1 - y_{A_1}) + y_{A_1}, \overline{h}_{A_1} y_{A_1} + (1 - y_{A_1}))\big)\big).
\end{aligned}$$

CLoss differs from the standard binary cross entropy loss function $\mathcal{L}$, as highlighted by the following example.

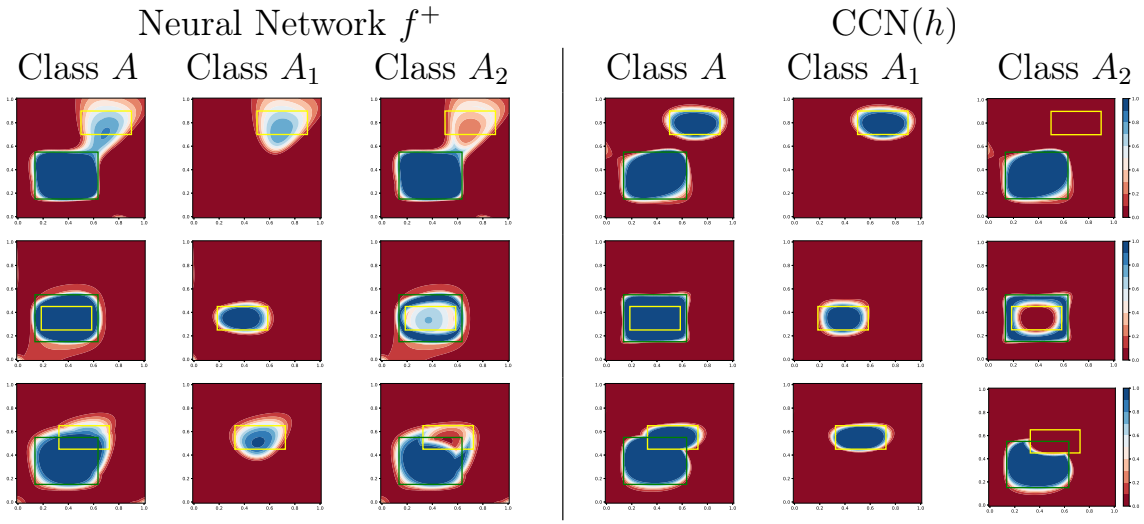


Figure 4.4: First three columns: decision boundaries of f for the classes A, A_1, and A_2. Last three columns: decision boundaries of CCN(h) for the classes A, A_1, and A_2. In each figure, the darker the blue (resp., red), the more confident a model is that the data points in the region is associated (resp. not associated) with the class (see the scale at the end of each row).

Example 4.2.1. Assume that $h_A = 0.6$, $h_{A_1} = 0.2$, $h_{A_2} = 0.3$, $y_A = y_{A_1} = 1$, and $y_{A_2} = 0$. Then,

$$\text{CLoss} = \text{CLoss}_A + \text{CLoss}_{A_1} + \text{CLoss}_{A_2} = -\ln(h_A) - \ln(h_{A_1}) - \ln(h_{A_1}),$$

and

$$\frac{\partial \text{CLoss}}{\partial h_A} = -\frac{1}{h_A} \sim -1.6, \qquad \frac{\partial \text{CLoss}}{\partial h_{A_1}} = -\frac{2}{h_{A_1}} = -10, \qquad \frac{\partial \text{CLoss}}{\partial h_{A_2}} = 0.$$

Thus, CCN(h) rightly learns to increase both h_A and h_{A_1}.

On the other hand, using the standard binary cross-entropy loss $\mathcal{L}$ after CM:

$$\mathcal{L} = -\ln(\text{CM}_A) - \ln(\text{CM}_{A_1}) - \ln(\overline{\text{CM}}_{A_2}) = -\ln(h_A) - \ln(h_{A_1}) - \ln(\overline{h}_A),$$

and thus

$$\frac{\partial \mathcal{L}}{\partial h_A} = -\frac{1}{h_A} + \frac{1}{\overline{h}_A} \sim 0.8, \qquad \frac{\partial \mathcal{L}}{\partial h_{A_1}} = -\frac{1}{h_{A_1}} = -5, \qquad \frac{\partial \mathcal{L}}{\partial h_{A_2}} = 0\,.$$

Hence, if trained with $\mathcal{L}$, CCN(h) would learn to decrease h_A while keeping h_{A_2} despite the fact that $y_A = 1$. ◁

To test the effectiveness of our approach, I consider again the yellow (R_1) and green (R_2) rectangles in Figure 4.1 with $A = R_1 \cup R_2$, $A_1 = R_1$, and $A_2 = R_2 \setminus R_1$. I

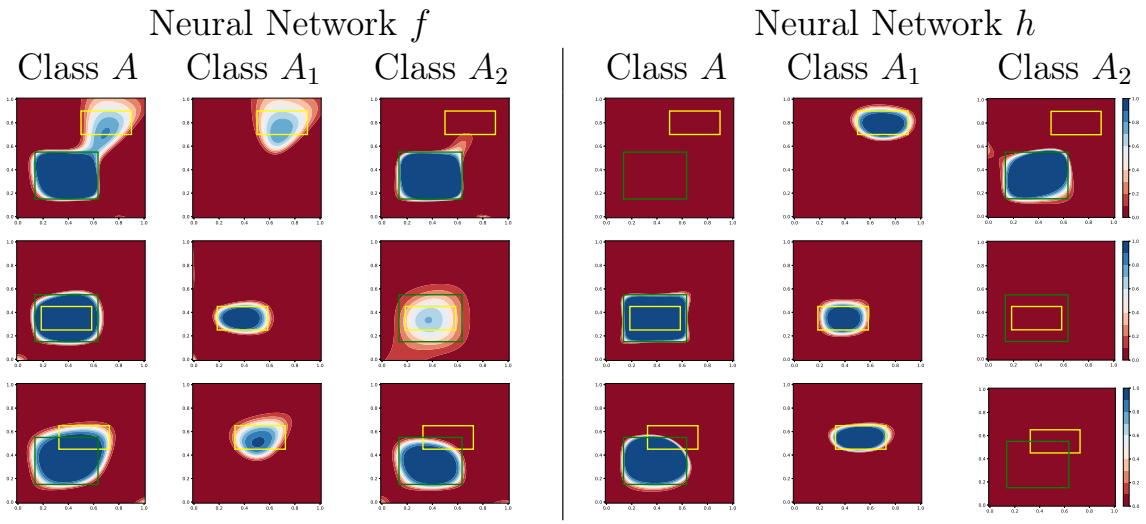


Figure 4.5: First three columns: decision boundaries of f for the classes A, A_1, and A_2. Last three columns: decision boundaries of CCN(h) for the classes A, A_1, and A_2. In each figure, the darker the blue (resp., red), the more confident a model is that the data points in the region belong (resp. do not belong) to the class (see the scale at the end of each row).

implemented f and h as feedforward neural networks with one hidden layer with 4 neurons and *tanh* nonlinearity. I trained f with binary cross-entropy loss, and CCN(h) using CLoss. I trained both networks for 20k epochs using Adam optimization [62], with learning rate 10^{-2} ($\beta_1 = 0.9, \beta_2 = 0.999$) [2]. The datasets consisted of 5000 (50/50 train/test split) data points sampled from a uniform distribution over $[0,1]^2$. In order to obtain predictions that are compliant with the constraints also for the neural network f, I applied an additional post-processing step at inference time, obtaining f^+, whose outputs are defined as follows:

$$\begin{aligned} f^+_A &= \max(f_A, f_{A_1}, f_{A_2}), \\ f^+_{A_1} &= f_{A_1}, \\ f^+_{A_2} &= \max(f_{A_2}, \min(f_A, \overline{f}_{A_1}), \min(f_{A_1}, \overline{f}_{A_1})). \end{aligned} \tag{4.6}$$

The final decision boundaries of f^+ (first three columns) and CCN(h) (last three columns) for all classes are plotted in Figure 4.4, while the decision boundaries of f (first three columns) and h (last three columns) are plotted in Figure 4.5. In these figures, it can be seen that f struggles, as expected, in learning the decision boundaries for the classes A and A_2, and that the application of the constraints as a

[2] In order to decide the learning rate, I further split the train set in train (70% of the datapoints) and validation set (30% of the datapoints) and searched the best learning rate among $\{10^{-1}, 10^{-2}, 10^{-3}\}$. β_1 and β_2 were kept fixed. The model was then trained on the entire train set with learning rate 10^{-2}.

post-processing step, as it happens in f^+, can lead to a decay in performance. On the contrary, it can be seen that CCN(h) is able to easily learn the decision boundaries for all the classes through a smart exploitation of the constraints. Indeed, as it can be seen in the last three columns of Figure 4.5, on the ground of the positions of the rectangles R_1 and R_2, CCN(h) knows which constraints to exploit:

- if R_1 and R_2 are arranged as in the first row, then CCN(h) exploits the constraints $A_1 \rightarrow A$ and $A_2 \rightarrow A$. Thus, the bottom module h does not learn A, which is instead computed from A_1 and A_2,
- if R_1 and R_2 are arranged as in the second row, then CCN(h) exploits the constraint $A, \neg A_1 \rightarrow A_2$. Thus, the bottom module h does not learn A_2, which is instead computed from A_1 and A_2, and
- if R_1 and R_2 are arranged as in the third row, then CCN(h) exploits the constraints $A, \neg A_1 \rightarrow A_2$ and $A_1 \rightarrow A$. Thus, the bottom module h does not learn A_2, and learns A only partially. Then, A_2 is computed from A and A_1, while A exploits A_1 to make predictions on the points belonging to R_2.

4.3 General Case

The general solution is now presented. I consider a general LCMC problem $(\mathcal{P}, \Pi)$ and a model h for $\mathcal{P}$. I first show how CCN(h) computes the set of classes associated to every data point (Section 4.3.1), and why the definition of CM requires some care in order to satisfy some desired properties, stated and motivated at the beginning of the same section. I then present the loss function used to train CCN(h) (Section 4.3.2). I end stating that CCN(h) is a generalization of C-HMCNN(h), that is, that C-HMCNN(h) and CCN(h) have the same behavior when given an HMC problem (Section 4.3.3).

4.3.1 Constraint Module (CM)

The basic idea of CCN(h) is to

1. have an initial set of classes decided by h, and
2. have all the other classes predicted also on the grounds of the constraints in Π.

In the example in the basic case, h decides $\{A_1\}$. Thus, every data point will or will not be assigned to class A_1 solely depending on the value of h_{A_1}. On the contrary, the decision on the classes $\{A, A_2\}$ takes into account not only h_A, h_{A_2} but also the constraints. In particular, CCN(h) may

1. predict A given the values of h_{A_1} and h_{A_2}, or
2. predict A_2 given the values of h_A and h_{A_1}.

The final set of classes $\mathcal{M}$ predicted by CCN(h) will

1. extend the set of classes $\mathcal{H}$ predicted by h (i.e., $\mathcal{H} \subseteq \mathcal{M}$) and be coherent with Π;
2. be such that any class in $\mathcal{M} \setminus \mathcal{H}$ is the head of a constraint r in Π whose body is satisfied by $\mathcal{M}$;
3. include only those classes that are either in $\mathcal{H}$ or are forced to be in $\mathcal{M}$ because of the rules in Π;
4. be unique, that is, there will be no other set of classes $\mathcal{M}'$ satisfying the above requirements.

The first requirement is the obvious one: the constraints in Π must be satisfied, and CCN(h) can only derive more classes in the head of the constraints. Whereas the second requirement is formalized by the concept of supportedness as defined in [2].

Definition 4.3.1. *Let $(\mathcal{P}, \Pi)$ be an LCMC problem. Let h be a model for $\mathcal{P}$. Let $\mathcal{H}$ be the set of classes predicted by h. A set of classes $\mathcal{M}$ is supported relative to $\mathcal{H}$ and Π, if for any class $A \in \mathcal{M}$, $A \in \mathcal{H}$, or there exists a constraint $r \in \Pi$ such that $head(r) = A$, $body^+(r) \subseteq \mathcal{M}$, and for each $\neg B \in body^-(r)$, $B \notin \mathcal{M}$.*

The third requirement is a minimality condition.

Definition 4.3.2. *Let $(\mathcal{P}, \Pi)$ be an LCMC problem. Let h be a model for $\mathcal{P}$. Let $\mathcal{H}$ be the set of classes predicted by h. $\mathcal{M}$ is minimal relative to $\mathcal{H}$ and Π, if there exists no set of classes $\mathcal{M}'$ with $\mathcal{H} \subseteq \mathcal{M}' \subset \mathcal{M}$ that is coherent with Π.*

The four requirements together ensure that the final predictions made by CCN(h) are coherent with Π and that can be uniquely explained on the grounds of the initial predictions made by h and the constraints in Π.

Intuitively, it could be expected that all the above requirements are met if, for each class A, I defined

$$m_A = \max(h_A, m_A^{r_1}, \ldots, m_A^{r_p}), \tag{4.7}$$

where $r_1, \ldots, r_p$ are all the constraints in Π with head A and, for each such constraint r_i of the form (4.1),

$$m_A^{r_i} = \min(m_{A_1}, \ldots, m_{A_k}, \overline{m}_{A_{k+1}}, \ldots, \overline{m}_{A_n}).$$

However, in general, the above equations may lead to not uniquely defined values and not minimal predictions because of circular definitions.

Example 4.3.3. If Π is the set of constraints

$$A_1 \rightarrow A_2; \qquad A_2 \rightarrow A_1, \tag{4.8}$$

then, by Equation (4.7), $m_{A_1} = \max(h_{A_1}, m_{A_2})$ and $m_{A_2} = \max(h_{A_2}, m_{A_1})$. This allows for infinitely many solutions, unless $h_{A_1} = 1$ or $h_{A_2} = 1$. Furthermore, any solution with $m_{A_1} = m_{A_2} > \theta$ when $h_{A_1} < \theta$ and $h_{A_2} < \theta$ leads to a set of predictions that satisfies the constraints but is not minimal. $\lhd$

I later show that such problems, due to circularities involving only positive classes (as the one in the example), can be solved if I consider the minimum of the set of tuples of values satisfying the equations (4.7): in the case of Example 4.3.3, the minimum is $m_{A_1} = m_{A_2} = \max(h_{A_1}, h_{A_2})$.[3]

However, more problems arise when there are circularities involving negated classes.

Example 4.3.4. If Π is the set of constraints

$$\neg A_1 \rightarrow A_2; \qquad \neg A_2 \rightarrow A_1, \tag{4.9}$$

then, by Equation (4.7), $m_{A_1} = \max(h_{A_1}, \overline{m}_{A_2})$ and $m_{A_2} = \max(h_{A_2}, \overline{m}_{A_1})$, and, for example, for $h_{A_1} = h_{A_2} = 0$, it exists no minimum pair of values satisfying the equations. Further, if I set $m_{A_1} = \max(h_{A_1}, \overline{h}_{A_2})$ and $m_{A_2} = \max(h_{A_2}, \overline{h}_{A_1})$, then if for a data point x, I get $h_{A_1} < \theta$ and $h_{A_2} < \theta$, then $m_{A_1} > \theta$ and $m_{A_2} > \theta$, that is, even if h predicts that x belongs to neither A_1 nor A_2, m predicts that x belongs to both A_1 and A_2, and the set of classes $\mathcal{M} = \{A_1, A_2\}$ is not supported relative to $\mathcal{H} = \emptyset$ and the constraints in (4.9). $\lhd$

To avoid the situation described in the above example, whenever the negation on a class is used, it should be referred to an already known value for the class itself. More specifically, first some classes should be computed without the use of negation. Next, some new classes can be computed possibly using the negation of the already computed classes, and this process can be iterated. When this is possible, the set of constraints is stratified [2].

There are several equivalent definitions of stratifiedness. Here, I use the one from [2].

[3]Given a set S of t-tuples of real numbers, $s_1, \ldots, s_t \in S$ is the *minimum of* S if for every $t_1, \ldots, t_t \in S$ and for every $1 \leq i \leq t$, $s_i \leq t_i$. Such a minimum might not exist.

Definition 4.3.5. *A set of constraints* Π *is stratified if there is a partition* $\Pi_1, \Pi_2, \ldots, \Pi_s$ *of* Π*, with* Π_1 *possibly empty, such that, for every* $i \in \{1, \ldots, s\}$*,*

1. *for every class* $A \in \cup_{r \in \Pi_i} body^+(r)$*, all the constraints with head* A *in* Π *belong to* $\cup_{j=1}^{i} \Pi_j$*;*

2. *for every* $\neg A \in \cup_{r \in \Pi_i} body^-(r)$*, all the constraints with head* A *in* Π *belong to* $\cup_{j=1}^{i-1} \Pi_j$*.*

$\Pi_1, \Pi_2, \ldots, \Pi_s$ *is a stratification of* Π*, and each* Π_i *is a stratum.*

I now show that both the check on whether Π is stratified and the computation of a possible stratification can be done on the dependency graph of Π [2].

Definition 4.3.6. *Let* $(\mathcal{P}, \Pi)$ *be an LCMC problem. The dependency graph* G_Π *of* Π *is the directed graph having the set of classes as nodes and with, for each constraint* $r \in \Pi$*,*

1. *a positive edge from each class in* $body^+(r)$ *to* $head(r)$*,*

2. *a negative edge from each class* A *such that* $\neg A \in body^-(r)$ *to* $head(r)$*.*

The following theorem is from [2].

Theorem 4.3.7 (Apt et al. 1988)**.** *Let* $(\mathcal{P}, \Pi)$ *be an LCMC problem.* Π *is stratified iff the dependency graph* G_Π *of* Π *contains no cycles with a negative edge.*

As an easy consequence of the above theorem, every set of constraints containing only definite rules (as, e.g., in the HMC case) is stratified. An example with a stratified and with a non-stratified set of constraints, both containing non definite rules, is the following.

Example 4.3.8. If $\mathcal{A} = \{A, A_1, A_2, A_3, A_4\}$, and Π is the set of constraints in (4.3), (4.4), and $A_3 \rightarrow A_4$, that is, $\Pi = \{A_1 \rightarrow A;\ A_2 \rightarrow A;\ A, \neg A_1 \rightarrow A_2;\ A_3 \rightarrow A_4\}$, then Π is stratified: for example, take $\Pi_1 = \{A_3 \rightarrow A_4\}$, and $\Pi_2 = \Pi \setminus \Pi_1$.[4] On the other hand, any set containing the constraints in (4.9) is not stratified. $\lhd$

For a stratified set of constraints, there can be many stratifications, as shown by the following example.

[4]This set Π of constraints is an example of a semi-positive set of rules [2]. A set Π is *semi-positive* if for every head A of a rule in Π, there is not a rule $r \in \Pi$ with $\neg A \in body(r)$. Every set of definite rules is also semi-positive, and every semi-positve set of rules is stratified.

Example 4.3.9 (Ex. 4.3.8, cont'd)**.** $\Pi'_1 = \{A_3 \to A_4\}$, $\Pi'_2 = \{A_1 \to A\}$, and $\Pi'_3 = \{A_2 \to A;\ A, \neg A_1 \to A_2\}$ is another stratification of the set Π of constraints in Example 4.3.8. $\lhd$

It is well known in the area of logic programming that all the stratifications lead to the same result [2]. Given this, comparing the two stratifications in Examples 4.3.8 and 4.3.9, the latter has two drawbacks:

1. the class A is in the head of constraints belonging to different strata, and
2. it has one more stratum.

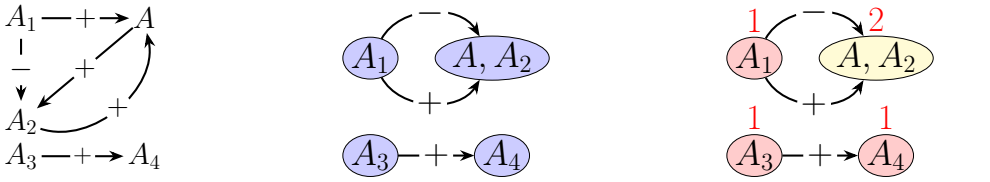
Indeed, for each stratum Π_i, the goal is to compute a value for all the classes in the head of the constraints in Π_i as a single step on GPUs, and thus

1. it is desirable to have all the constraints with the same head A just in one stratum, so that it is possible to compute a value for A in a single step, and
2. it is desirable to have as few strata as possible, to minimize the number of steps.

Thus, assuming Π is stratified,

1. I compute the acyclic component graph [21] of the dependency graph G_Π of Π, that is, the DAG obtained by shrinking each strongly connected component in G_Π into a single vertex (notice that since Π is stratified, negative edges are not involved in any cycle in G_Π),
2. I assign to the classes in each node of the DAG the number 1 plus the maximum number of negative edges connecting a root to the node, and
3. I define:
   (a) $\mathcal{A}_i$ as the set of classes having the number i assigned at the previous step, and
   (b) Π_i as the set of constraints in Π whose head is in $\mathcal{A}_i$.

I call the above procedure CompStrata(Π). Given a stratified set Π of constraints, CompStrata(Π) computes a partition $\mathcal{A}_1, \ldots, \mathcal{A}_s$ of the set $\mathcal{A}$ of classes and also the corresponding stratification $\Pi_1, \ldots, \Pi_s$ of Π with the smallest possible number of strata.



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

Figure 4.6: Given Π as in Example 4.3.10, 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.

Example 4.3.10 (Ex. 4.3.8 cont'd)**.** If $\Pi = \{A_1 \to A;\ A_2 \to A;\ A, \neg A_1 \to A_2;\ A_3 \to A_4\}$, then CompStrata($\Pi$) computes $\mathcal{A}_1 = \{A_1, A_3, A_4\}$, $\mathcal{A}_2 = \{A, A_2\}$, $\Pi_1 = \{A_3 \to A_4\}$, and $\Pi_2 = \Pi \setminus \Pi_1$, as shown in Figure 4.6. ◁

I now prove that CompStrata(Π) indeed computes the stratification of Π having the smallest number of strata.

Theorem 4.3.11. *Let $(\mathcal{P}, \Pi)$ be an LCMC problem with stratified Π. Let $\Pi_1, \ldots, \Pi_s$ be the partition of Π computed by* CompStrata(Π)*. Then,*

1. *$\Pi_1, \ldots, \Pi_s$ is a stratification of Π, and*
2. *there exists no stratification of Π with a smaller number of strata.*

Proof. By induction on the number s of strata.

Assume $s = 1$. Then, the constraints in Π are definite rules, $\Pi_1 = \Pi$, and the statement follows.

Assume $s = n + 1 > 1$, and let $\mathcal{A}_1, \ldots, \mathcal{A}_{n+1}$ be the partition of the set $\mathcal{A}$ of classes computed by CompStrata(Π). By the induction hypothesis, $\Pi_1, \ldots, \Pi_n$ is a stratification of $\bigcup_{i=1}^{n} \Pi_i$ with the smallest number of strata. Thus, by construction, for each class A in $\mathcal{A}_{n+1}$ the path from a root to A in G_Π containing the maximum number of negative edges contains n negative edges. Thus, for each class $A \in \mathcal{A}_{n+1}$, there exists a constraint $r \in \Pi_{n+1}$ such that:

1. $head(r) = A$,
2. there exists a class $B \in \mathcal{A}_n$ such that $\neg B \in body^-(r)$.

Furthermore, $\Pi = \bigcup_{i=1}^{n+1} \Pi_i$ is stratified, since $\bigcup_{i=1}^{n} \Pi_i$ is stratified by the induction hypothesis and, for every constraint $r \in \Pi_{n+1}$,

1. each class in $body^+(r)$ belongs to $\bigcup_{i=1}^{n+1} \mathcal{A}_i$, and

2. each class in $body^-(r)$ belongs to $\bigcup_{i=1}^{n} \mathcal{A}_i$.

Since $\Pi_1, \ldots, \Pi_n$ is a stratification of $\bigcup_{i=1}^{n} \Pi_i$ with the smallest number of strata, then $\Pi_1, \ldots, \Pi_{n+1}$ is a stratification of Π with the smallest number of strata. □

In the sequel, assume Π is stratified, and that $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_s$ and $\Pi_1, \Pi_2, \ldots, \Pi_s$ are the partition of $\mathcal{A}$ and the stratification of Π computed by CompStrata(Π), respectively.

Consider the ith stratum ($1 \leq i \leq s$).

If there is more than one stratum ($s > 1$), the values for the classes in $\mathcal{A}_i$ are iteratively computed on the ground of the values in $\mathcal{A}_1 \cup \ldots \cup \mathcal{A}_{i-1}$. However, inside each single stratum (even the first one), there can be a chain of rules affecting the values of the classes in the stratum. Consider, for example, the set of constraints $\Pi = \{A_1, A_2 \rightarrow A_3; A_3 \rightarrow A\}$. In this case,

1. it is not desirable to first compute the final value for A_3 as $\max(h_{A_3}, \min(h_{A_1}, h_{A_2}))$ and then, in a second step, use it to compute the final value for A (which could be problematic if there is also, e.g., $A \rightarrow A_3$), instead

2. it is desirable to directly compute the final value for A as $\max(h_A, h_{A_3}, \min(h_{A_1}, h_{A_2}))$ as a single operation.

I therefore compute and then use the transitive closure of all the constraints in the same stratum. The additional constraints in the closure are conceptually redundant, but they allow for an improvement in performance, as in the work by [29].

Define Π_i^* to be the set of constraints

1. initially equal to Π_i, and then

2. obtained by recursively adding the constraints obtained from a constraint r already in Π_i^* by substituting a class $A \in body^+(r) \cap \mathcal{A}_i$ with $body(r')$ for any rule r' with $head(r') = A$ (hence, $r' \in \Pi_i$), and finally

3. eliminating the constraints r such that $head(r) \in body^+(r)$, or for which there exists another constraint $r' \in \Pi_i$ with $head(r) = head(r')$ and $body(r') \subset body(r)$.

Π_i^* is guaranteed to be finite, since the set of classes $\mathcal{A}$ is finite, and repetitions in the body of constraints are not allowed. The constraints being eliminated in the third step are redundant.

Then, for each class $A \in \mathcal{A}_i$, the output CM_A of the constraint module CM is defined as

$$\mathrm{CM}_A = \max(h_A, h_A^{r_1}, \ldots, h_A^{r_p}), \tag{4.10}$$

where

1. $r_1, \ldots, r_p$ are all the constraints in Π_i^* with head A, and
2. assuming $r \in \Pi_i^*$ has the form (4.1),

$$h_A^r = \min(v_{A_1}, \ldots, v_{A_k}, \overline{\mathrm{CM}}_{A_{k+1}}, \ldots, \overline{\mathrm{CM}}_{A_n}),$$

with $v_{A_1} = h_{A_1}$ if $A_1 \in \mathcal{A}_i$, and $v_{A_1} = \mathrm{CM}_{A_1}$ if $A_1 \in \cup_{j=1}^{i-1}\mathcal{A}_j$. Analogously for $v_{A_2}, \ldots, v_{A_k}$.

The above definition is well-founded:

1. Π_1^* does not contain negated classes, and thus the definition of CM_A when $A \in \mathcal{A}_1$ relies only on the outputs of the bottom module h, and
2. the definition of CM_A when $A \in \mathcal{A}_i$ $(i > 1)$ uses only outputs of h or of already defined outputs of CM.

Example 4.3.12 (Ex. 4.3.10, cont'd)**.** $\Pi_1 = \Pi_1^* = \{A_3 \rightarrow A_4\}$, while $\Pi_2^* = \Pi_2 \cup \{A_1, \neg A_1 \rightarrow A_2\}$. Thus,

$$\mathrm{CM}_{A_1} = h_{A_1}, \quad \mathrm{CM}_{A_3} = h_{A_3}, \quad \mathrm{CM}_{A_4} = \max(h_{A_3}, h_{A_4}),$$
$$\mathrm{CM}_A = \max(h_A, \mathrm{CM}_{A_1}, h_{A_2}), \mathrm{CM}_{A_2} = \max(h_{A_2}, \min(h_A, \overline{\mathrm{CM}}_{A_1}), \min(\mathrm{CM}_{A_1}, \overline{\mathrm{CM}}_{A_1})).$$

If $h_{A_1} = 0.2$, $h_{A_2} = 0.3$, $h_A = 0.6$ (as in Example 4.2.1), then $\mathrm{CM}_{A_1} = 0.2$, $\mathrm{CM}_{A_2} = 0.6$, $\mathrm{CM}_A = 0.6$.

The constraint $A_1, \neg A_1 \rightarrow A_2 \in \Pi_2^*$, which leads to the inclusion of $\min(\mathrm{CM}_{A_1}, \overline{\mathrm{CM}}_{A_1}) = \min(h_{A_1}, \overline{h}_{A_1})$ in the definition of CM_{A_2}, is necessary in order to guarantee that CM never violates (4.4), that is, that it always holds

$$\min(\mathrm{CM}_A, \overline{\mathrm{CM}}_{A_1}) \leq \mathrm{CM}_{A_2}. \tag{4.11}$$

Indeed assume, $h_A = h_{A_2} = 0.3, h_{A_1} = 0.6$. Then, $\mathrm{CM}_A = \mathrm{CM}_{A_1} = 0.6$, $\mathrm{CM}_{A_2} = 0.4$, and (4.11) is satisfied. If CM_{A_2} was defined as $\mathrm{CM}_{A_2} = \max(h_{A_2}, \min(h_A, \overline{\mathrm{CM}}_{A_1}))$ (omitting $\min(\mathrm{CM}_{A_1}, \overline{\mathrm{CM}}_{A_1})$), then it would have been equal to $\mathrm{CM}_{A_2} = 0.3$, and (4.11) would have been no longer satisfied. $\lhd$

Given a stratified set Π of constraints, $\mathrm{CCN}(h)$ is guaranteed to always satisfy Π.

Theorem 4.3.13. *Let $(\mathcal{P}, \Pi)$ be an LCMC problem. Assume Π is stratified. Let h be a model for $\mathcal{P}$. Then,* $\mathrm{CCN}(h)$ *satisfies Equation (4.7) and thus commits no constraint violations.*

Proof. Let $\mathcal{A}_1, \ldots, \mathcal{A}_s$ be the partition of $\mathcal{A}$ computed by CompStrata(Π). Recall that, for a class A, (4.7) is

$$m_A = \max(h_A, m_A^{r_1}, \ldots, m_A^{r_p}),$$

where $r_1, \ldots, r_p$ are all the constraints in Π with head A and, for each such constraint r_j of form (4.1),

$$m_A^{r_j} = \min(m_{A_1}, \ldots, m_{A_k}, \overline{m}_{A_{k+1}}, \ldots, \overline{m}_{A_n}).$$

Consider a generic class $A \in \mathcal{A}_i$. I show that the definition of CM_A is equal to the expression resulting from the substitution of each m_{A_l} in $m_A^{r_j}$ $(1 \leq j \leq p)$ with

1. CM_{A_l} if $A_l \in \bigcup_{j=1}^{i-1} \mathcal{A}_j$,
2. the right-hand side of Equation (4.10) if $A_l \in \mathcal{A}_i$.

Consider the result of such a substitution in $m_A^{r_j}$. Applying the distributivity of the minimum operation over the maximum operation, I get

$$m_A^{r_j} = \max_{r \in \Pi_i^*(r_j)} (\min(v_{body(r)})),$$

where

1. $\Pi_i^*(r_j)$ is the set of constraints initially equal to $\{r_j\}$ and then obtained by recursively adding the constraints obtained from a constraint $r \in \Pi_i^*(r_j)$ by substituting a class $B \in body^+(r) \cap \mathcal{A}_i$ with $body(r')$ for any constraint r' with $head(r') = B$, and
2. $v_{body(r)}$ is the set

$$\begin{gathered}\{\mathrm{CM}_B : B \in body^+(r), B \in \cup_{j=1}^{i-1} \mathcal{A}_j\} \cup \\ \{h_B : B \in body^+(r), B \in \mathcal{A}_i\} \cup \\ \{\overline{\mathrm{CM}}_B : \neg B \in body^-(r)\}.\end{gathered}$$

Since the set of constraints in Π_i^* with head A is equal to $\bigcup_{j=1}^{p} \Pi_i^*(r_j)$, the statement follows. □

Given the values for the classes in $\bigcup_{j=1}^{i-1} \mathcal{A}_j$, the values of $\mathrm{CCN}(h)$ for the classes in $\mathcal{A}_i$ correspond to the minimum of the set of tuples satisfying (4.7).

Theorem 4.3.14. *Let $(\mathcal{P}, \Pi)$ be an LCMC problem. Assume Π is stratified. Let h be a model for $\mathcal{P}$. Let $\mathcal{A}_1, \ldots, \mathcal{A}_s$ be the partition of $\mathcal{A}$ computed by* CompStrata(Π). *For $1 \leq i \leq s$, let m be a model for $\mathcal{P}$ satisfying Equation (4.7) and such that for every class $B \in \bigcup_{j=1}^{i-1} \mathcal{A}_j$, $m_B = \mathrm{CCN}(h)_B$. For every class $A \in \mathcal{A}_i$, $m_A \geq \mathrm{CCN}(h)_A$.*

Proof. Recall that, for each class A, $\mathrm{CCN}(h)_A = \mathrm{CM}_A$, thus, I use CM_A, since it is shorter.

Consider the partition $\mathcal{A}_1, \ldots, \mathcal{A}_s$ of the set of classes $\mathcal{A}$ and the corresponding stratification $\Pi_1, \ldots, \Pi_s$ of Π. I prove that the outputs of CM for the classes in $\mathcal{A}_i$ are the smallest values satisfying (4.7), given the values CM_B for $B \in \bigcup_{j=1}^{i-1} \mathcal{A}_j$.

If a model satisfies (4.7), then it also satisfies the inequalities (4.2) associated with each constraint (4.1) in Π_i. Thus, I first prove that for any model m satisfying (4.2) for each constraint (4.1) in Π_i, given the values m_B for $B \in \bigcup_{j=1}^{i-1} \mathcal{A}_j$, $m_A \geq \mathrm{CM}_A$. This allows us to conclude that any model m satisfying (4.7) has values bigger than or equal to those of CM.

Let $I(\Pi_i)$ be the set of inequalities (4.2), one for each constraint of the form (4.1) in Π_i union, for each class $A \in \mathcal{A}_i$,

$$h_A \leq m_A. \tag{4.12}$$

I represent $I(\Pi_i)$ as the set of pairs $\langle \mathcal{S}, m_A \rangle$, where $\mathcal{S}$ is the set $\{m_{A_1}, \ldots, \overline{m}_{A_n}\}$ (resp., $\{h_A\}$) in the case of (4.2) (resp., (4.12)).

Define $I^*(\Pi_i)$ to be the set of inequalities obtained from $I(\Pi_i)$ by recursively adding the pairs $\langle \mathcal{S} \cup \mathcal{S}' \setminus A, m_B \rangle$ such that $\langle \mathcal{S}, m_A \rangle \in I^*(\Pi_i)$, $\langle \mathcal{S}', m_B \rangle \in I^*(\Pi_i)$ and $A \in \mathcal{S}'$. $I^*(\Pi_i)$ is finite, and each inequality in $I^*(\Pi_i)$ is entailed by the inequalities in $I(\Pi_i)$. Thus, $I^*(\Pi_i)$ and $I(\Pi_i)$ have the same set of solutions. For each constraint r in Π_i^* of the form (4.1) with head $A \in \mathcal{A}_i$, the inequality

$$\langle \{v_{A_1}, \ldots, v_{A_k}, \overline{m}_{A_{k+1}}, \ldots, \overline{m}_{A_n}\}, A \rangle \tag{4.13}$$

belongs to $I^*(\Pi_i)$, where $v_{A_1} = h_{A_1}$ if $A_1 \in \mathcal{A}_i$, and $v_{A_1} = m_{A_1}$ if $A_1 \in \bigcup_{j=1}^{i-1} \mathcal{A}_j$. Analogously for $v_{A_2}, \ldots, v_{A_k}$.

By definition, CM_A is the smallest value satisfying the inequalities (4.13) in $I^*(\Pi_i)$. Thus, for any model m satisfying $I^*(\Pi_i)$, $m_A \geq \mathrm{CM}_A$. □

I can now state that $\mathrm{CCN}(h)$ has the desired properties mentioned at the beginning of the section.

Theorem 4.3.15. *Let* $(\mathcal{P}, \Pi)$ *be an LCMC problem. Assume* Π *is stratified. Let* h *be a model for* $\mathcal{P}$*. Let* $\mathcal{H}$ *be the set of classes predicted by* h*. Let* $\mathcal{M}$ *be the set of classes predicted by* CCN*(*h*). Then,* $\mathcal{M}$

1. *extends* $\mathcal{H}$ *and is coherent with* Π*,*
2. *is supported relative to* $\mathcal{H}$ *and* Π*,*
3. *is minimal relative to* $\mathcal{H}$ *and* Π*, and*
4. *is the unique set satisfying the previous properties.*

Proof. Each statement of the theorem is proven separately. By hypothesis, $\mathcal{H}$ is the set of classes A such that $h_A > \theta$. $\mathcal{M}$ is the set of classes A such that $\mathrm{CM}_A > \theta$. $\mathcal{A}_1, \ldots, \mathcal{A}_s$ and $\Pi_1, \ldots, \Pi_s$ is the partition of $\mathcal{A}$ and Π computed by CompStrata(Π), respectively.

1. CCN(h) extends the set of classes associated with x by h. I have to prove that $\mathcal{H} \subseteq \mathcal{M}$. $A \in \mathcal{H}$ iff $h_A > \theta$. Since $\mathrm{CM}_A \geq h_A$, if $A \in \mathcal{H}$, then $A \in \mathcal{M}$.

   $\mathcal{M}$ is coherent with Π. I have to prove that for each constraint r of the form (4.1), if $\min(\mathrm{CM}_{A_1}, \ldots, \mathrm{CM}_{A_k}, \overline{\mathrm{CM}}_{A_{k+1}}, \ldots, \overline{\mathrm{CM}}_{A_n}) > \theta$, then $\mathrm{CM}_A > \theta$, which is an easy consequence of the fact that the outputs of CM satisfy (4.7) and thus also (4.2).

2. $\mathcal{M}$ is supported relative to $\mathcal{H}$ and Π. Assume it is not. Then, consider a class $A \in \mathcal{M} \setminus \mathcal{H}$ such that for each constraint $r \in \Pi$ of the form (4.1), there exists an index $j \in \{1, \ldots, n\}$ such that

   (a) both $\mathrm{CM}_{A_j} \leq \theta$ and $j = 1, \ldots, k$, or

   (b) both $\overline{\mathrm{CM}}_{A_j} \leq \theta$ and $j = k+1, \ldots, n$.

   By Theorem 4.3.13, CM_A is the smallest value satisfying Equation (4.7) given the values m_B for $B \in \bigcup_{j=1}^{i-1} \mathcal{A}_j$. Thus, $\mathrm{CM}_A > \theta$ is not possible, and this contradicts $A \in \mathcal{M}$.

3. $\mathcal{M}$ is minimal relative to $\mathcal{H}$ and Π. Let $\mathcal{M}_0 = \mathcal{H}$. Let $\mathcal{M}_{i+1}$ be the closure of $\mathcal{M}_i$ under Π^*_{i+1} $(1 \leq i < s)$. The statement is a consequence of the minimality of each $\mathcal{M}_{i+1}$ and the fact that $\mathcal{M} = \mathcal{M}_s$.

4. $\mathcal{M}$ is the unique set satisfying all the previous properties. As above, let $\mathcal{M}_0 = \mathcal{H}$, and let $\mathcal{M}_{i+1}$ be the closure of $\mathcal{M}_i$ under Π^*_{i+1} $(1 \leq i < s)$. The statement is a consequence of the uniqueness of each $\mathcal{M}_{i+1}$ and the fact that $\mathcal{M} = \mathcal{M}_s$.

□

I now show that, if the given set of constraints is interpreted as a stratified normal logic program, then it is possible to establish a relation with the canonical model semantics of stratified normal logic programs [2], which coincides with the stable model semantics [47].

Definition 4.3.16. *Let Π be a finite set of normal rules. Let $\mathcal{M}$ be a set of classes. The reduct of Π relative to $\mathcal{M}$ is the set $\Pi^{\mathcal{M}}$ of definite rules obtained by:*

1. *dropping the rules r in Π such that for a class $A \in \mathcal{M}$, $\neg A \in body(r)$, and then*
2. *dropping $body^-(r)$ from the remaining rules r.*

$\mathcal{M}$ is a stable model of Π iff $\mathcal{M}$ is the smallest set closed under $\Pi^{\mathcal{M}}$.

Example 4.3.17. Let $\Pi = \{A_1 \rightarrow A;\ A_2 \rightarrow A;\ A, \neg A_1 \rightarrow A_2;\ A_3 \rightarrow A_4\}$. Then,

1. the stable model of Π is the empty set,
2. the stable model of $\Pi \cup \{\rightarrow A\}$ is $\{A_2, A\}$, and
3. the stable model of $\Pi \cup \{\rightarrow A; \rightarrow A_4\}$ is $\{A_2, A_4, A\}$. ◁

Theorem 4.3.18. *Let $(\mathcal{P}, \Pi)$ be an LCMC problem with stratified Π. Let h be a model for $\mathcal{P}$. Let $\mathcal{H}$ be the set of classes predicted by h. Let $\mathcal{M}$ be the set of classes predicted by* CCN(h)*. Then, $\mathcal{M}$ is the stable model of the set of constraints $\Pi \cup \{\rightarrow A : A \in \mathcal{H}\}$.*

Proof. First, observe that if Π is stratified, then also $\Pi \cup \{\rightarrow A : A \in \mathcal{H}\}$ is stratified. The theorem is an easy consequence of the fact that in the case of a stratified set of constraints, the stable and the canonical model semantics coincide (see, e.g., [47]), and the latter, for $\Pi \cup \{\rightarrow A : A \in \mathcal{H}\}$, is defined as follows.

Consider the partition $\mathcal{A}_1, \ldots, \mathcal{A}_s$ of the set of classes $\mathcal{A}$ and the corresponding stratification $\Pi_1, \ldots, \Pi_s$ resulting from CompStrata(Π). Define $\mathcal{M}_0 = \mathcal{H}$, and, for each $0 \leq i < s$,

$$\mathcal{M}_{i+1} = T(\mathcal{M}_i, \Pi^{\mathcal{M}_i}_{i+1}),$$

where $T(\mathcal{M}_i, \Pi^{\mathcal{M}_i}_{i+1})$ is the smallest superset of $\mathcal{M}_i$ closed under the reduct $\Pi^{\mathcal{M}_i}_{i+1}$ of Π_{i+1} relative to $\mathcal{M}_i$. $\mathcal{M}_{i+1}$ is unique.

Then, the canonical model $\mathcal{M}$ of $\Pi \cup \{\rightarrow A : A \in \mathcal{H}\}$ is $\mathcal{M} = \mathcal{M}_s$. □

Finally, the definition of the output CM_A of $\mathrm{CCN}(h)$ for class $A \in \mathcal{A}_i$ is based on the computation of Π_i^*, which, as it can be seen in Example 4.3.12, may contain constraints with logically contradictory bodies (i.e., with B and $\neg B$ in the body, for some class B). Indeed, if in the construction of Π_i^* such constraints with contradictory bodies are not included, then the resulting system may exhibit

1. constraint violations (as seen in Example 4.3.12), but
2. still no logical violations, since the set of predicted classes does not change.

4.3.2 Constraint Loss (CLoss)

In the general case, for every data point, the value of the loss function CLoss used to train $\mathrm{CCN}(h)$ is defined as:

$$\mathrm{CLoss} = \sum_{A \in \mathcal{A}} \mathrm{CLoss}_A,$$

CLoss_A being the value of the loss for class A, defined as:

$$\mathrm{CLoss}_A = -y_A \ln(\mathrm{CM}_A^+) - (1 - y_A) \ln(\overline{\mathrm{CM}}_A^-),$$

where:

- y_A is the ground truth for class A,
- CM_A^+ is the value to optimize when $y_A = 1$, and
- CM_A^- is the value to optimize when $y_A = 0$.

CM_A^+ and CM_A^- differ from the output value CM_A of $\mathrm{CCN}(h)$ for class A, that is, from $\mathrm{CCN}(h)_A$. Indeed, as it has been the case in the HMC setting and already discussed in the basic case, for CM_A^+ and CM_A^- the ground-truth should also be taken into account.

Similarly to what has been done for computing CM_A, for a stratified set of constraints, the computation of CM_A^+ and CM_A^- will be done stratum after stratum, starting from the first. I thus assume:

1. that Π is stratified,
2. that $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_s$ and $\Pi_1, \Pi_2, \ldots, \Pi_s$ ($s \geq 1$) are the partitions of $\mathcal{A}$ and Π computed by CompStrata(Π), and
3. that $\Pi_1^*, \Pi_2^*, \ldots, \Pi_s^*$ are the sets of constraints corresponding to $\Pi_1, \Pi_2, \ldots, \Pi_s$ and defined as in the previous subsection.

The values CM_A^+ and CM_A^- associated to a class A in the ith stratum will depend on the set Π_A^* of constraints with head A in Π_i^*, and thus on:

1. the values computed by model h for the class A and for the classes in the ith stratum belonging to the body of a constraint in Π_A^*,
2. the values of the already computed loss for the classes in the lower strata belonging to the body of a constraint in Π_A^*,
3. the ground truth for the classes in the body of the constraints in Π_A^*.

Consider a class A in the ith stratum (i.e., $A \in \mathcal{A}_i$). To each constraint r with head A in Π_i^* I associate two values

1. $h_A^{+,r}$ to be used with $y_A = 1$, and
2. $h_A^{-,r}$ to be used with $y_A = 0$.

Assume $y_A = 1$. Consider a constraint r in Π_i^* with head A of the form (4.1). Then, I want to teach $\mathrm{CCN}(h)$ to possibly exploit the constraint r for predicting A if $y_{A_1} = \ldots = y_{A_k} = 1$ and $y_{A_{k+1}} = \ldots = y_{A_n} = 0$. Thus, I define

$$h_A^{+,r} = \min\big(v_{A_1} y_{A_1}, \ldots, v_{A_k} y_{A_k}, \overline{v}_{A_{k+1}}(1 - y_{A_{k+1}}), \ldots, \overline{v}_{A_n}(1 - y_{A_n})\big),$$

where v_{A_l} is

1. h_{A_l} if $A_l \in \mathcal{A}_i$ (and thus $1 \leq l \leq k$),
2. $\mathrm{CM}_{A_l}^+$ if $A_l \in \cup_{j=1}^{i-1} \mathcal{A}_j$ and $1 \leq l \leq k$,
3. $\mathrm{CM}_{A_l}^-$ if $A_l \in \cup_{j=1}^{i-1} \mathcal{A}_j$ and $k+1 \leq l \leq n$.

The value CM_A^+ associated to class A when $y_A = 1$ is

$$\mathrm{CM}_A^+ = \max(h_A, h_A^{+,r_1}, \ldots, h_A^{+,r_p}),$$

where $r_1, \ldots, r_p$ are all the constraints in Π_i^* with head A.

Assume $y_A = 0$. Consider a constraint r in Π_i^* with head A of the form (4.1). Then, for some class $A_l \in \mathit{body}^+(r)$ $y_{A_l} = 0$, or for some class $A_l \in \mathit{body}^-(r)$ $y_{A_l} = 1$, and I want to teach $\mathrm{CCN}(h)$ to not fire the constraint r. I thus define

$$\begin{aligned} h_A^{-,r} = \min\big(&v_{A_1}(1 - y_{A_1}) + y_{A_1}, \ldots, v_{A_k}(1 - y_{A_k}) + y_{A_k}, \\ &\overline{v}_{A_{k+1}} y_{A_{k+1}} + 1 - y_{A_{k+1}}, \ldots, \overline{v}_{A_n} y_{A_n} + 1 - y_{A_n}\big), \end{aligned}$$

where v_{A_l} now is

1. h_{A_l} if $A_l \in \mathcal{A}_i$ (and thus $1 \leq l \leq k$),

2. $\mathrm{CM}^-_{A_l}$ if $A_l \in \cup_{j=1}^{i-1}\mathcal{A}_j$ and $1 \leq l \leq k$,

3. $\mathrm{CM}^+_{A_l}$ if $A_l \in \cup_{j=1}^{i-1}\mathcal{A}_j$ and $k+1 \leq l \leq n$.

The value CM^-_A associated with the class A when $y_A = 0$ is

$$\mathrm{CM}^-_A = \max(h_A, h_A^{-,r_1}, \ldots, h_A^{-,r_p}),$$

where $r_1, \ldots, r_p$ are all the constraints in Π^*_i with head A.

Example 4.3.19. Consider the simpler version of Example 4.3.12 with $\mathcal{A} = \{A_1, A_2, A\}$, $\Pi = \{A_1 \to A; A_2 \to A; A, \neg A_1 \to A_2\}$, $h_{A_1} = 0.2$, $h_{A_2} = 0.3$, $h_A = 0.6$, as in Example 4.2.1. Then, $\mathcal{A}_1 = \{A_1\}$, $\mathcal{A}_2 = \{A, A_2\}$, $\Pi^*_1 = \emptyset$, $\Pi^*_2 = \Pi \cup \{A_1, \neg A_1 \to A_2\}$ (see also Example 4.3.12). Assume $y_{A_1} = y_A = 1$ and $y_{A_2} = 0$.

If $r_1, \ldots, r_4$ are the constraints listed as above, then

1. $h_A^{+,r_1} = h_{A_1} = 0.2$, $h_A^{+,r_2} = 0$,

2. $\mathrm{CM}^+_{A_1} = h_{A_1} = 0.2$, $\mathrm{CM}^+_A = h_A = 0.6$,

3. $h_{A_2}^{-,r_3} = 1 - h_{A_1} = 0.8$, $h_{A_2}^{-,r_4} = 1 - h_{A_1} = 0.8$, and

4. $\mathrm{CM}^-_{A_2} = 1 - h_{A_1} = 0.8$.

Thus,

$$\mathrm{CLoss} = -\ln(h_{A_1}) - \ln(1 - (1 - h_{A_1})) - \ln(h_A) = -2\ln(h_{A_1}) - \ln(h_A),$$

as already calculated in Example 4.2.1. ◁

As in the hierarchical case, CLoss has the fundamental property that the negative gradient descent algorithm behaves as expected, that is, that for each class, it moves in the "right" direction as given by the ground truth.

Theorem 4.3.20. *Let $(\mathcal{P}, \Pi)$ be an LCMC problem. For any model h for $\mathcal{P}$ and class A, let $\frac{\partial \mathrm{CLoss}}{\partial h_A}$ be the partial derivative of* CLoss *with respect to h_A. For each data point, if $y_A = 0$, then $\frac{\partial \mathrm{CLoss}}{\partial h_A} \geq 0$, and if $y_A = 1$, then $\frac{\partial \mathrm{CLoss}}{\partial h_A} \leq 0$.*

Proof. Consider a data point and a class A.

$$\frac{\partial \mathrm{CLoss}}{\partial h_A} = \sum_{B \in \mathcal{A}} \frac{\partial \mathrm{CLoss}_B}{\partial h_A}.$$

I consider only the case $y_A = 1$ (the case $y_A = 0$ is analogous). Consider a class B.

1. If $y_B = 1$,

$$\text{CLoss}_B = -\ln(\text{CM}_B^+), \qquad \frac{\partial \text{CLoss}_B}{\partial h_A} = -\frac{1}{\text{CM}_B^+}\frac{\partial \text{CM}_B^+}{\partial h_A} \leq 0,$$

because:

- either $\text{CM}_B^+ = h_A$ (which is possible only if $A = B$, or there exists a constraint r with head B such that $h_B^{+,r} = h_A$) and then $\frac{\partial \text{CLoss}_B}{\partial h_A} = -\frac{1}{h_A} \leq 0$,
- or CM_B^+ is a value not dependent on h_A and then $\frac{\partial \text{CLoss}_B}{\partial h_A} = 0$ (since $y_A = 1$, it cannot be the case that there exists a constraint r with head B such that $h_B^{+,r} = \overline{h}_A$).

2. if $y_B = 0$,

$$\text{CLoss}_B = -\ln(\overline{\text{CM}}_B^-), \qquad \frac{\partial \text{CLoss}_B}{\partial h_A} = \frac{1}{\overline{\text{CM}}_B^-}\frac{\partial \text{CM}_B^-}{\partial h_A} \leq 0,$$

because:

- either $\text{CM}_B^- = \overline{h}_A$ (which is possible only if there exists a constraint r with head B such that $h_B^{-,r} = \overline{h}_A$) and then $\frac{\partial \text{CLoss}_B}{\partial h_A} = -\frac{1}{h_A} \leq 0$,
- or CM_B^- is a value not dependent on h_A and then $\frac{\partial \text{CLoss}_B}{\partial h_A} = 0$ (since $y_A = 1$, it cannot be the case that there exists a constraint r with head B such that $h_B^{-,r} = h_A$).

Since $\frac{\partial \text{CLoss}}{\partial h_A}$ is the sum of quantities that are at most zero, then $\frac{\partial \text{CLoss}}{\partial h_A} \leq 0$. □

4.3.3 Relation Between C-HMCNN(h) and CCN(h)

From the definitions of CM_A and CLoss_A, it is clear that they generalize the corresponding definitions given in the hierarchical case. Thus, C-HMCNN(h) and CCN(h) have the same behavior when considering HMC problems.

Theorem 4.3.21. *Let $(\mathcal{P}, \Pi)$ be an HMC problem. Let h be a model for $\mathcal{P}$. For any class A,* C-HMCNN*(h)*$_A$ = CCN*(h)*$_A$.

Proof. A LCMC problem is a pair $(\mathcal{P}, \Pi)$ where each constraint in Π has form (4.1). An HMC problem is a particular case of a LCMC in which each constraint in Π presents a single positive literal in the body and the dependency graph G_Π contains no cycles. I now show that, given an HMC problem, MCM = CM and MCLoss = CLoss.

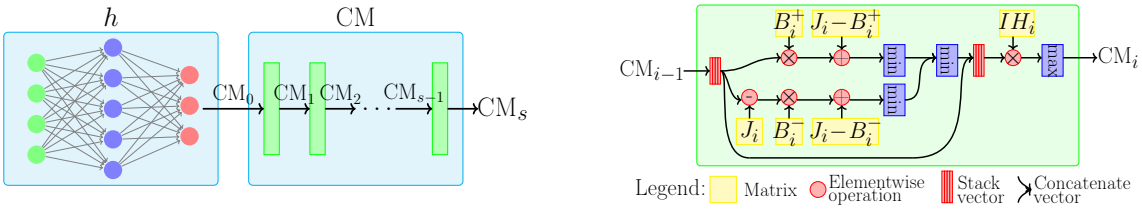


Figure 4.7: Visual representation of CCN(h) (left), and details of the operations associated with each stratum (right).

Notice that, for each $r \in \Pi$, $body^-(r) = \emptyset$. Thus, all the constraints in Π belong to the same stratum. For each class A, the output CM_A is then defined as:

$$\mathrm{CM}_A = \max(h_A, h_A^{r_1}, \ldots, h_A^{r_p})$$

where $r_1, \ldots, r_p$ are all the constraints in Π^* with head A, and $h_A^r = h_{body(r)}$. By construction of Π^*, $\{body(r_1), \ldots, body(r_p)\} = \mathcal{S}_A$, and thus CM = MCM. Similar calculations can be done to show that MCLoss = CLoss. □

4.4 GPU Implementation

In the previous section, both CM_A and CLoss_A have been defined for a specific class A. However, it is possible to compute both CM_A and CLoss_A for all classes in the same stratum in parallel, leveraging GPU architectures. In this section, I show how to compute the values first of the constraint module and then of the loss function on a GPU.

Consider an LCMC problem $(\mathcal{P}, \Pi)$ with stratified Π. I assume to have l classes (i.e., $|\mathcal{A}| = l$), that $\Pi_1, \Pi_2, \ldots, \Pi_s$ is the stratification computed by CompStrata(Π), and that $\Pi_1^*, \Pi_2^*, \ldots, \Pi_s^*$ are the corresponding sets as defined in the previous section.

4.4.1 Constraint Module

The basic idea, starting from the vector CM_0 (which contains the l values resulting from the bottom module h) is to iteratively compute the vector of values CM_i corresponding to the outputs of CM if given the set of constraints $\bigcup_{j=1}^{i} \Pi_j^*$. The final output of CM will correspond to CM_s. Figure 4.7 shows a visual representation of the process and of CCN(h).

For each $i \in \{1, \ldots, s\}$, p_i is the number of constraints in Π_i^* ($p_i = |\Pi_i^*|$), r_{ij} denotes the jth constraint in Π_i^*, and

- B_i^+ is the $p_i \times l$ matrix whose j, k element is 1 if $A_k \in body^+(r_{ij})$, and 0 otherwise;

- B_i^- is the $p_i \times l$ matrix whose j,k element is 1 if $\neg A_k \in \mathit{body}^-(r_{ij})$, and 0 otherwise;

- C_{i-1} is the $p_i \times l$ matrix obtained by stacking p_i times CM_{i-1}.

Stacking p times a vector v of size q returns the $p \times q$ matrix $1_p^T \times v$ whose j,k element is $v[k]$.

Then, the jth value $v_i[j]$ of the vector v_i associated with the body of the constraint r_{ij} is

$$v_i^+ = \min(B_i^+ \odot C_{i-1} + (J_{p_i,l} - B_i^+), \mathrm{dim} = 1),$$
$$v_i^- = \min(B_i^- \odot (J_{p_i,l} - C_{i-1}) + (J_{p_i,l} - B_i^-), \mathrm{dim} = 1),$$
$$v_i[j] = \min(v_i^+[j], v_i^-[j]),$$

where

1. $\odot$ represents the Hadamard product,

2. $J_{p,q}$ is the $p \times q$ matrix of ones, and

3. given an arbitrary $p \times q$ matrix Q, $\min(Q, \mathrm{dim} = 1)$ (resp., $\max(Q, \mathrm{dim} = 1)$) returns a vector of length p whose ith element is equal to $\min(Q_{i1}, \ldots, Q_{iq})$ (resp., $\max(Q_{i1}, \ldots, Q_{iq})$).

Then, the output of the ith layer associated with the ith stratum is given by:

$$\mathrm{CM}_i = \max(IH_i \odot V_i, \mathrm{dim} = 1),$$

where

- H_i is the $l \times p_i$ matrix whose j,k element is 1 if $A_j = \mathit{head}(r_{ik})$, and 0 otherwise,

- IH_i is the $l \times (l + p_i)$ matrix obtained by stacking the $l \times l$ identity matrix and H_i,

- V_i is the $l \times (l + p_i)$ matrix obtained by stacking l times the concatenation of CM_{i-1} and v_i.

Example 4.4.1. Let $\mathcal{A} = \{A_1, A_2, A\}$, $\Pi = \{A_1 \rightarrow A; A_2 \rightarrow A; A, \neg A_1 \rightarrow A_2\}$, $h_{A_1} = 0.2$, $h_{A_2} = 0.3$, $h_A = 0.6$, as in Example 4.3.12. Then, $\mathcal{A}_1 = \{A_1\}$, $\mathcal{A}_2 = \{A, A_2\}$, $\Pi_1^* = \emptyset$, $\Pi_2^* = \Pi \cup \{A_1, \neg A_1 \rightarrow A_2\}$.

Then, $\mathrm{CM}_0 = \begin{bmatrix}0.2 & 0.3 & 0.6\end{bmatrix}$, B_1^+, B_1^-, C_0, H_1 are the empty matrices, IH_1 is the 3×3 identity matrix, while

$$B_2^+ = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\\ 1 & 0 & 0\end{bmatrix} \quad B_2^- = \begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 1 & 0 & 0\\ 1 & 0 & 0\end{bmatrix} \quad C_1 = 1_4^T \times \begin{bmatrix}0.2 & 0.3 & 0.6\end{bmatrix} = \begin{bmatrix}0.2 & 0.3 & 0.6\\ 0.2 & 0.3 & 0.6\\ 0.2 & 0.3 & 0.6\\ 0.2 & 0.3 & 0.6\end{bmatrix}$$

$$H_2 = \begin{bmatrix}0 & 0 & 0 & 0\\ 0 & 0 & 1 & 1\\ 1 & 1 & 0 & 0\end{bmatrix} \quad IH_2 = \begin{bmatrix}1 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & 0 & 1 & 1\\ 0 & 0 & 1 & 1 & 1 & 0 & 0\end{bmatrix}$$

$$V_1 = 1_3^T \times \begin{bmatrix}0.2 & 0.3 & 0.6\end{bmatrix} \quad \mathrm{CM}_1 = \begin{bmatrix}0.2 & 0.3 & 0.6\end{bmatrix}$$

$$v_2^+ = \begin{bmatrix}0.2 & 0.3 & 0.6 & 0.2\end{bmatrix} \quad v_2^- = \begin{bmatrix}1 & 1 & 0.8 & 0.8\end{bmatrix} \quad v_2 = v_2^+$$

$$V_2 = 1_3^T \times \begin{bmatrix}0.2 & 0.3 & 0.6 & 0.2 & 0.3 & 0.6 & 0.2\end{bmatrix} \quad \mathrm{CM}_2 = \begin{bmatrix}0.2 & 0.6 & 0.6\end{bmatrix}$$

and thus $\mathrm{CM}_{A_1} = h_{A_1} = 0.2$, $\mathrm{CM}_{A_2} = h_A = 0.6$, and $\mathrm{CM}_A = h_A = 0.6$, as expected (see Example 4.3.12). ◁

4.4.2 Constraint Loss

I now show how to compute CLoss_A for all classes in parallel, leveraging GPU architectures. Here, I define Y_i the $p_i \times l$ matrix obtained by stacking p_i times the ground-truth vector y, and $v_i^+[j]$ to be the value associated with the body of the jth constraint $r_{ij} \in \Pi_i^*$ when the ground-truth class $y_{head(r_{ij})} = 1$:

$$\begin{gathered} v_i^{+\backslash +} = \min(B_i^+ \odot C_{i-1} \odot Y_i + (J_{p_i,l} - B_i^+), \dim = 1),\\ v_i^{+\backslash -} = \min(B_i^- \odot (J_{p_i,l} - C_{i-1}) \odot (J_{p_i,l} - Y_i) + (J_{p_i,l} - B_i^-), \dim = 1),\\ v_i^+[j] = \min(v_i^{+\backslash +}[j], v_i^{+\backslash -}[j]). \end{gathered}$$

Analogously, $v_i^-[j]$ is the value associated with the body of the jth constraint $r_{ij} \in \Pi_i^*$ when $y_{head(r_{ij})} = 0$:

$$\begin{gathered} v_i^{-\backslash +} = \min(B_i^+ \odot C_{i-1} \odot (J_{p_i,l} - Y_i) + (J_{p_i,l} - B_i^+) + B_i^+ \odot Y_i, \dim = 1),\\ v_i^{-\backslash -} = \min(B_i^- \odot (J_{p_i,l} - C_{i-1}) \odot Y_i + (J_{p_i,l} - B_i^-) + B_i^- \odot (J_{p_i,l} - Y_i), \dim = 1),\\ v_i^-[j] = \min(v_i^{-\backslash +}[j], v_i^{-\backslash -}[j]). \end{gathered}$$

Then, the output of the ith layer associated with the ith stratum is given by:

$$\mathrm{CM}_i^{+\backslash -} = \max\big((IH_i \odot V_i^+) \odot IY_i + (IH_i \odot V_i^-) \odot (J_{l,l+p_i} - IY_i), \dim = 1\big),$$

where:

- $\mathrm{CM}_0^{+\backslash -} = \mathrm{CM}_0$,
- IY_i is the $l \times (l + p_i)$ matrix obtained by stacking the $l \times l$ identity matrix and Y_i^T, the transposed of Y_i,
- V_i^+ is the $l \times (l + p_i)$ matrix obtained by stacking l times the concatenation of $\mathrm{CM}_{i-1}^{+\backslash -}$ and v_i^+, and
- V_i^- is the $l \times (l + p_i)$ matrix obtained by stacking l times the concatenation of $\mathrm{CM}_{i-1}^{+\backslash -}$ and v_i^-.

Once I have computed $\mathrm{CM}_s^{+\backslash -}$, I can compute CLoss by using the standard binary cross-entropy loss (BCELoss), as given in any standard library (e.g., PyTorch):

$$\mathrm{CLoss} = \mathrm{BCELoss}(y, \mathrm{CM}_s^{+\backslash -}).$$

Example 4.4.2 (Ex. 4.4.1, cont'd)**.** Assume that $y_A = y_{A_1} = 1$, and $y_{A_2} = 0$. Then, $y = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$, Y_1 is the empty matrix, $V_1^+ = V_1^- = V_1$, IY_1 is the identity matrix,

$$\mathrm{CM}_1^{+\backslash -} = \begin{bmatrix} 0.2 & 0.3 & 0.6 \end{bmatrix}$$

$$B_2^+ \odot C_1 \odot Y_2 + (J_{4,3} - B_2^+) = \begin{bmatrix} 0.2 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0.6 \\ 0.2 & 1 & 1 \end{bmatrix}$$

$$B_2^- \odot (J_{4,3} - C_1) \odot (J_{4,3} - Y_2) + (J_{4,3} - B_2^-) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

$$v_2^+ = \begin{bmatrix} 0.2 & 0 & 0 & 0 \end{bmatrix}$$

The only value which is not 0 is the first one, corresponding to the first constraint, being the only constraint whose body is satisfied by the ground truth.

$$B_2^+ \odot C_1 \odot (J_{4,3} - Y_2) + (J_{4,3} - B_2^+) + B_2^+ \odot Y_2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0.3 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$B_2^- \odot (J_{4,3} - C_1) \odot Y_2 + (J_{4,3} - B_2^-) + B_2^- \odot (J_{4,3} - Y_2) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0.8 & 1 & 1 \\ 0.8 & 1 & 1 \end{bmatrix}$$

$$v_2^- = \begin{bmatrix} 1 & 0.3 & 0.8 & 0.8 \end{bmatrix}$$

The values which are not 1 correspond to the last four constraints, being the constraints whose body is not satisfied by the ground truth.

$$Y_2 = 1_4^T \times \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$$

$$V_2^+ = 1_3^T \times \begin{bmatrix} 0.2 & 0.3 & 0.6 & 0.2 & 0 & 0 & 0 \end{bmatrix} \qquad V_2^- = 1_3^T \times \begin{bmatrix} 0.2 & 0.3 & 0.6 & 1 & 0.3 & 0.8 & 0.8 \end{bmatrix}$$

$$\begin{array}{c} (IH_2 \odot V_2^+) \odot IY_2 \\ + \\ (IH_2 \odot V_2^-) \odot (J_{3,7} - IY_2) \end{array} = \begin{bmatrix} 0.2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.3 & 0 & 0 & 0 & 0.8 & 0.8 \\ 0 & 0 & 0.6 & 0.2 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathrm{CM}_2^{+\setminus -} = \begin{bmatrix} 0.2 & 0.8 & 0.6 \end{bmatrix}$$

The three values of $\mathrm{CM}_2^{+\setminus -}$ correspond to the expected values of $\mathrm{CM}_{A_1}^+, \mathrm{CM}_{A_2}^-, \mathrm{CM}_A^+$ as computed in Example 4.3.19, and equal to $h_{A_1}, 1 - h_{A_1}, h_A$, respectively. $\lhd$

4.5 Experimental Analysis

In this section, I present the results of the experimental analysis. As for the hierarchical case, I first consider a generalization of the synthetic experiment proposed in the basic case. Then I test CCN(h) on 16 real-world datasets with general constraints, and finally I present the ablation studies.[5]

About the metrics, the analysis of 64 papers on MC problems conducted by [113] and reported by [91], shows that already in 2013 as many as 19 different metrics have been used to evaluate MC models, and still today different papers use different subsets of such metrics. However, as suggested by [91], not all subsets can be used, as the experimental results may appear to favor a specific behavior depending on the subset of measures chosen, thus possibly leading to misleading conclusions. To avoid such undesired results, the authors conducted a correlation analysis of the metrics that led to the individuation of clusters of correlated measures and thus to the proposal of various subsets of metrics, chosen according to the following criteria:

1. first, Hamming loss is highly recommended for inclusion in each subset: it is not correlated with others, and is the most used metric in the literature (55 papers out of the 64 surveyed),

2. next it could be considered employing other measures not correlated with any others like coverage error and ranking loss, and

[5]Link: https://github.com/EGiunchiglia/CCN/

3. finally, a suitable selection should include at least one metric from each cluster of correlated measures. Among them, multi-label accuracy is a good choice because it is among the ones with the highest correlations to other measures.

Other criteria they suggest and use for the selection of the proposed subsets are: popularity in the literature, the choice to include or not AUC-based metrics, and the size of the resulting set of metrics.

Following the above criteria, I used the following six metrics, each taking value in the interval $[0, 1]$ and each annotated with either $\uparrow$ or $\downarrow$ to mean that larger values for that metric stand for better (resp. worse) performance:

1. average precision ($\uparrow$),
2. coverage error ($\downarrow$),
3. Hamming loss ($\downarrow$),
4. multi-label accuracy ($\uparrow$),
5. one-error ($\downarrow$), and
6. ranking loss ($\downarrow$).

The above six metrics are exactly those belonging to the first two subsets of metrics proposed by [91].[6] Notice that, the above list does not include $AU(\overline{PRC})$: already in 2013 and still today, contrarily to the specialized HMC literature, it is generally not used in the MC literature [113, 91, 41].

Finally, notice that, since in the case of HMC problems there is no difference between CCN(h) and C-HMCNN(h) (see Theorem 4.3.21), all the experimental results presented in Chapter 3 also hold for CCN(h).

4.5.1 Synthetic Experiment

Consider the generalization of the experiment presented as basic case, in which R_1 started outside R_2 (as in the first row of Figure 4.4), and then R_1 moved towards the centre of R_2 (as in the second row of Figure 4.4) in 9 uniform steps. As for HMC problems, this experiment is meant to show how the performance of CCN(h) and f^+ defined as in Section 4.2 vary depending on the relative positions of R_1 and R_2. As expected, Figure 4.8 shows that CCN(h) performs better or equally to f^+ at all steps and for all metrics. In particular:

[6] In particular, the first of the proposed subsets includes coverage error, Hamming loss, multi-label accuracy and ranking loss, while the second includes average precision, coverage error, Hamming loss, one-error and ranking loss; see [91] for more details.

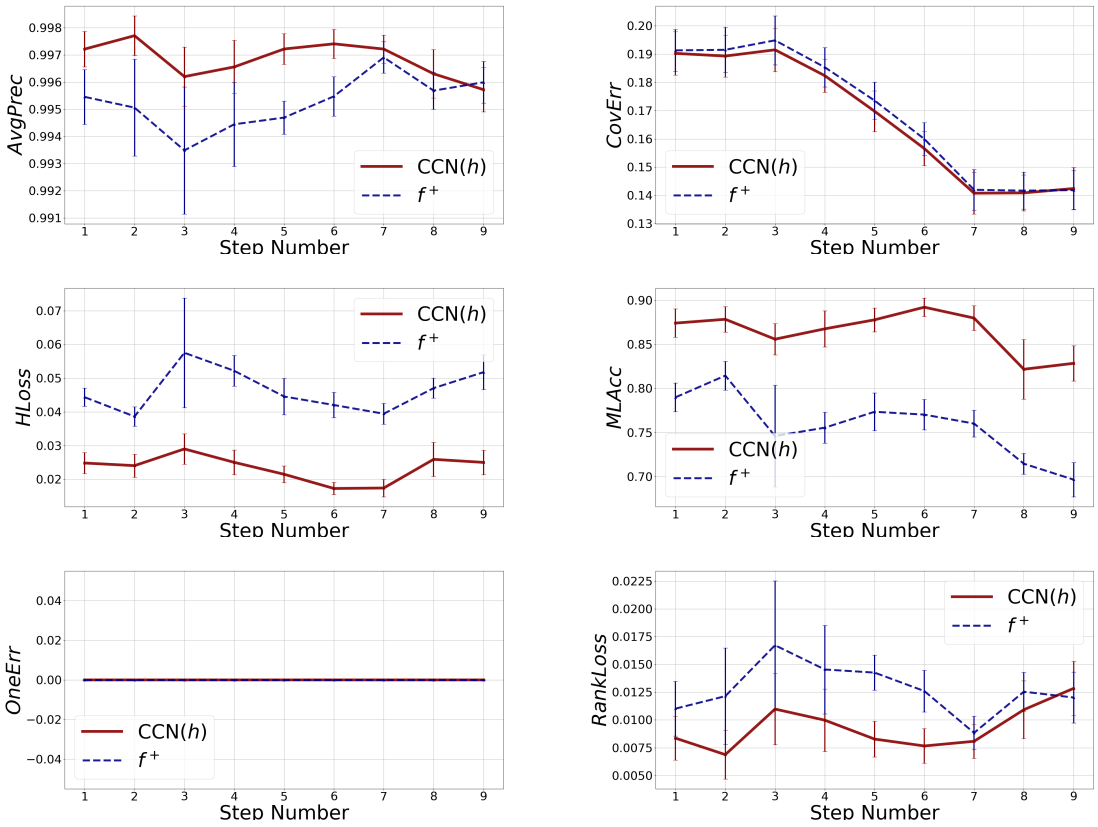


Figure 4.8: Mean average precision, coverage error, hamming loss, multi-label accuracy, one-error, and ranking loss with standard deviation of CCN(h) and f^+ for each step.

- CCN(h) performs consistently better than f^+ at all steps in terms of average precision, coverage error, Hamming loss, multi-label accuracy and ranking loss. Further, as in the HMC case, CCN(h) exhibits much more stable performances than f^+ as highlighted by the visibly much smaller standard deviation bar of CCN(h).

- CCN(h) and f^+ perform identically in terms of one-error. This is due to the fact that one-error measures the fraction of instances whose most confident class is irrelevant, since neither CCN(h) nor f^+ ever make this mistake, they both have one-error equal to zero at all steps.

4.5.2 Comparison with the State-of-the-Art

In order to prove the superiority the proposed model, I adopted the same methodology presented in [41] and consider the three well-established MC models and the state-of-

Dataset	D	L	Train	Val	Test	C	H	B	B^+	B^-	H/L
Arts	462	26	2975	525	1500	344	11	7.1	5.6	1.5	42.3%
Bibtex	1836	159	4148	732	2515	399	32	4.2	4.2	0.0	20.1%
Business	438	30	2975	525	1500	77	7	2.5	2.3	0.2	23.3%
Cal500	68	174	298	53	151	39	7	2.0	1.1	0.9	4.0%
Delicious	500	983	10982	1938	3185	104	61	1.0	1.0	0.0	6.1%
Emotions	72	6	332	59	202	1	1	5.0	0.0	5.0	16.7%
Enron	1001	53	954	169	579	7	5	1.0	1.0	0.0	9.4%
Genbase	1186	27	393	70	199	88	13	2.2	2.0	0.2	48.1%
Image	294	5	1190	210	600	1	1	4.0	0.0	4.0	20.0%
Medical	1449	45	283	50	645	17	9	1.2	1.2	0.0	20.0%
Rcv1subset1	944	101	3570	630	1800	247	16	3.7	2.9	0.8	15.8%
Rcv1subset2	944	101	3570	630	1800	81	15	2.4	1.7	0.7	14.9%
Rcv1subset3	944	101	3570	630	1800	72	16	2.3	1.7	0.6	15.8%
Rcv1subset4	944	101	3570	630	1800	63	14	2.1	1.7	0.4	13.9%
Rcv1subset5	944	101	3570	630	1800	73	11	2.5	2	0.5	10.9%
Science	743	40	2975	525	1500	37	11	2.1	1.7	0.4	27.5%
Scene	294	6	1029	182	1196	1	1	5.0	0.0	5.0	16.7%
Yeast	103	14	1275	225	917	34	11	2.3	1.9	0.4	78.6%

Table 4.1: Summary of the real-world MC datasets. For each dataset, I report from left to right: (i) name, (ii) number of features (D), (iii) number of classes (L), (iv-vi) number of data points for each split, (vii) number of constraints (C), (iix) number of different classes that appear at least once as head of a constraint (H), (ix) average number of classes appearing in the body (B), (x-xi) average number of classes appearing positively (resp., negatively) in the body (B^+), (resp., (B^-)), and (xii) percentage of classes appearing at least once as head of a constraint.

the-art MC model tested in [41], which can be characterized by the order of classes correlations they exploit. Thus, CCN(h) is compared with

1. BR [9], a first order model which considers each class separately, ignoring class correlations, and

2. ECC [97], RAKEL [123] and CAMEL [41], which exploit correlations among two or more classes.

BR, ECC, and RAKEL are the well-established MC models, and CAMEL is the current state-of-the-art MC model [41]. A description of each of these models can be found in Chapter 2 (Section 2.1). Since these models are not guaranteed to output predictions that are coherent with the constraints, I applied CM as additional post-processing steps.

Model	Arts	Bibtex	Business	Cal500	Delicious	Emotions	Enron	Genbase	Image
				Average precision (↑)					
CCN(h)	0.623	0.586	**0.904**	**0.520**	0.347	**0.800**	0.704	0.996	**0.807**
CAMEL	**0.625**	**0.605**	0.899	0.513	**0.398**	0.756	**0.708**	0.990	0.793
ECC	0.544	0.302	0.867	0.401	0.112	0.772	0.643	**1.000**	0.738
BR	0.546	0.308	0.863	0.441	0.140	0.793	0.643	**1.000**	0.726
RAKEL	0.530	0.317	0.856	0.433	0.140	0.798	0.636	**1.000**	0.721
				Coverage error (↓)					
CCN(h)	**0.172**	**0.105**	**0.065**	**0.734**	**0.520**	**0.315**	**0.217**	0.016	**0.187**
CAMEL	0.202	0.157	0.083	0.791	0.613	0.372	0.256	0.010	0.201
ECC	0.223	0.801	0.089	0.853	0.987	0.338	0.285	**0.009**	0.242
BR	0.217	0.802	0.086	0.789	0.989	0.324	0.288	**0.009**	0.245
RAKEL	0.221	0.796	0.085	0.791	0.988	0.317	0.294	**0.009**	0.250
				Hamming loss (↓)					
CCN(h)	**0.054**	**0.013**	**0.023**	**0.136**	**0.018**	**0.197**	**0.046**	**0.001**	**0.172**
CAMEL	0.055	**0.013**	**0.023**	0.138	**0.018**	0.265	0.047	0.003	0.174
ECC	0.081	**0.013**	0.031	0.172	0.019	0.245	0.055	**0.001**	0.218
BR	0.079	**0.013**	0.032	0.162	**0.018**	0.229	0.054	**0.001**	0.232
RAKEL	0.082	**0.013**	0.034	0.165	0.019	0.223	0.055	**0.001**	0.225
				Multi-label accuracy (↑)					
CCN(h)	**0.238**	**0.272**	0.601	0.203	0.905	**0.534**	**0.395**	0.986	**0.488**
CAMEL	0.218	0.193	**0.609**	0.210	0.147	0.354	0.381	0.943	0.456
ECC	0.217	0.250	0.548	0.220	0.114	0.446	0.361	**0.992**	0.387
BR	0.217	0.260	0.538	0.221	0.150	0.465	0.365	**0.992**	0.369
RAKEL	0.215	0.263	0.527	**0.222**	**0.152**	0.485	0.361	**0.992**	0.376
				One-error (↓)					
CCN(h)	0.475	0.376	0.093	**0.113**	0.357	**0.273**	0.235	**0.000**	**0.296**
CAMEL	**0.460**	**0.349**	**0.090**	0.133	**0.315**	0.381	**0.223**	0.020	0.310
ECC	0.568	0.535	0.137	0.378	0.692	0.332	0.309	**0.000**	0.392
BR	0.567	0.517	0.147	0.232	0.546	0.292	0.299	**0.000**	0.425
RAKEL	0.586	0.513	0.159	0.232	0.567	0.292	0.304	**0.000**	0.430
				Ranking loss (↓)					
CCN(h)	**0.115**	**0.058**	**0.030**	**0.173**	**0.110**	**0.161**	**0.076**	0.003	**0.159**
CAMEL	0.136	0.078	0.040	0.189	0.117	0.237	0.086	**0.001**	0.177
ECC	0.158	0.680	0.046	0.257	0.861	0.193	0.107	**0.001**	0.231
BR	0.155	0.670	0.045	0.218	0.820	0.177	0.108	**0.001**	0.234
RAKEL	0.159	0.659	0.044	0.220	0.815	0.169	0.112	**0.001**	0.242

Table 4.2: Comparison of CCN(h) with the other state-of-the-art models. The best results are in bold.

Model	Medical	Rcv1S1	Rcv1S2	Rcv1S3	Rcv1S4	Rcv1S5	Science	Scene	Yeast
					Average precision (↑)				
CCN(h)	**0.866**	**0.642**	**0.666**	**0.647**	**0.675**	0.560	0.603	**0.868**	**0.768**
CAMEL	0.807	0.622	0.647	0.636	0.654	**0.564**	**0.614**	0.824	0.766
ECC	0.823	0.549	0.575	0.585	0.609	0.529	0.502	0.794	0.724
BR	0.823	0.536	0.563	0.572	0.600	0.524	0.500	0.781	0.743
RAKEL	0.811	0.532	0.556	0.562	0.589	0.508	0.493	0.794	0.732
					Coverage error (↓)				
CCN(h)	**0.035**	**0.092**	**0.089**	**0.103**	**0.080**	**0.107**	**0.131**	**0.077**	**0.452**
CAMEL	0.036	0.131	0.115	0.123	0.103	0.130	0.162	0.106	0.457
ECC	0.045	0.185	0.166	0.167	0.169	0.196	0.225	0.127	0.495
BR	0.045	0.194	0.181	0.178	0.184	0.210	0.227	0.128	0.476
RAKEL	0.049	0.201	0.180	0.185	0.195	0.209	0.225	0.123	0.481
					Hamming loss (↓)				
CCN(h)	**0.013**	**0.026**	**0.022**	**0.024**	**0.019**	**0.025**	**0.031**	**0.092**	**0.196**
CAMEL	0.024	0.027	**0.022**	**0.024**	0.021	**0.025**	**0.031**	0.109	**0.196**
ECC	0.019	0.031	0.027	0.028	0.026	0.030	0.049	0.131	0.221
BR	0.019	0.032	0.028	0.029	0.027	0.031	0.051	0.151	0.214
RAKEL	0.019	0.033	0.029	0.030	0.027	0.031	0.051	0.130	0.225
					Multi-label accuracy (↑)				
CCN(h)	**0.589**	**0.296**	**0.310**	**0.303**	**0.324**	**0.275**	**0.255**	**0.607**	**0.480**
CAMEL	0.284	0.204	0.222	0.210	0.257	0.223	0.217	0.528	**0.480**
ECC	0.481	0.264	0.277	0.273	0.297	0.269	0.209	0.478	0.443
BR	0.477	0.263	0.279	0.275	0.289	0.263	0.200	0.438	0.456
RAKEL	0.481	0.263	0.272	0.268	0.290	0.258	0.201	0.481	0.445
					One-error (↓)				
CCN(h)	**0.181**	**0.413**	**0.389**	**0.405**	**0.379**	**0.402**	0.494	**0.224**	0.234
CAMEL	0.285	**0.413**	0.397	0.413	0.399	0.414	**0.472**	0.287	**0.231**
ECC	0.251	0.477	0.462	0.453	0.436	0.451	0.603	0.319	0.300
BR	0.251	0.492	0.474	0.466	0.435	0.466	0.605	0.358	0.259
RAKEL	0.266	0.488	0.481	0.471	0.439	0.474	0.606	0.329	0.270
					Ranking loss (↓)				
CCN(h)	**0.024**	**0.036**	**0.035**	**0.046**	**0.035**	**0.046**	**0.094**	**0.073**	**0.172**
CAMEL	0.026	0.051	0.048	0.050	0.046	0.054	0.117	0.101	0.173
ECC	0.033	0.086	0.078	0.078	0.086	0.093	0.176	0.103	0.208
BR	0.032	0.091	0.088	0.085	0.097	0.101	0.177	0.131	0.190
RAKEL	0.037	0.093	0.088	0.089	0.103	0.102	0.180	0.127	0.200

Table 4.3: Comparison of CCN(h) with the other state-of-the-art models. The best results are in bold.

Metric	Average ranking					Friedman test	Wilcoxon test
	CCN(h)	CAMEL	ECC	BR	RAKEL	p-value	CCN(h) vs. CAMEL
Average precision	**1.50**	2.17	3.53	3.58	4.22	✓✓	
Coverage Error	**1.22**	2.36	3.61	3.92	3.89	✓✓	✓✓
Hamming loss	**1.36**	2.56	3.39	3.61	3.92	✓✓	✓✓
Multi-label accuracy	**1.69**	3.58	3.19	3.27	3.25	✓✓	✓✓
One-error	**1.50**	2.08	3.67	3.56	4.19	✓✓	
Ranking loss	**1.22**	2.19	3.61	3.72	4.17	✓✓	✓✓

Table 4.4: Average ranking for each metric and model, results of the Friedman and Wilcoxon test (the latter deployed to compare the performance of CCN(h) and CAMEL). I use ✓ (resp., ✓✓) to indicate that the test returned p-value < 0.05 (resp., < 0.01).

Being the first paper on LCMC problems, I created 16 real-world LCMC datasets, each obtained by enriching a popular and publicly available MC dataset with constraints extracted using the apriori algorithm [1], which is the standard algorithm used for association rules mining.

I now explain how the apriori algorithm can be used to obtain the constraints given the MC datasets. Given a set of n binary attributes $\mathcal{I} = \{i_1, \ldots, i_n\}$ and a set $\mathcal{T} = \{t_1, t_2, \ldots, t_m\}$ where each $t_j \subseteq \mathcal{I}$ $(1 \leq j \leq m)$, the apriori algorithm is able to find all the association rules which hold true in $\mathcal{T}$. An association rule has form

$$\mathcal{X} \Rightarrow i, \tag{4.14}$$

where $\mathcal{X} \in \mathcal{T}$ and $i \in \mathcal{I}$. The rule states that, for each $t \in \mathcal{T}$, if the set $\mathcal{X} \subseteq t$ then $i \in t$. Given the above, for each multi-label dataset $\mathcal{D} = (X, Y)$ having as classes the set $\mathcal{A}$, I can define the set $\mathcal{I} = \mathcal{A} \cup \{\neg A \mid A \in \mathcal{A}\}$, and the set $\mathcal{T} = \{\{A \mid A \in y\} \cup \{\neg A \mid A \notin y, A \in \mathcal{A}\} \mid y \in Y\}$, and then use the apriori algorithm to find the desired logical constraints.[7] Once I obtained the set of constraints, I then pruned it to obtain a set of stratified normal rules.

The list of the datasets together with a summary of their characteristics is reported in Table 4.1. The various datasets come from different application domains, in particular:

1. Cal500 and Emotions are 2 music classification datasets [124, 122],
2. Genbase and Yeast are 2 functional genomics datasets [35, 39],
3. Image and Scene are 2 image classification datasets [139, 9], and

[7]For computational reasons, I set the maximum number of elements in the body of the rules to be discovered equal to 4.

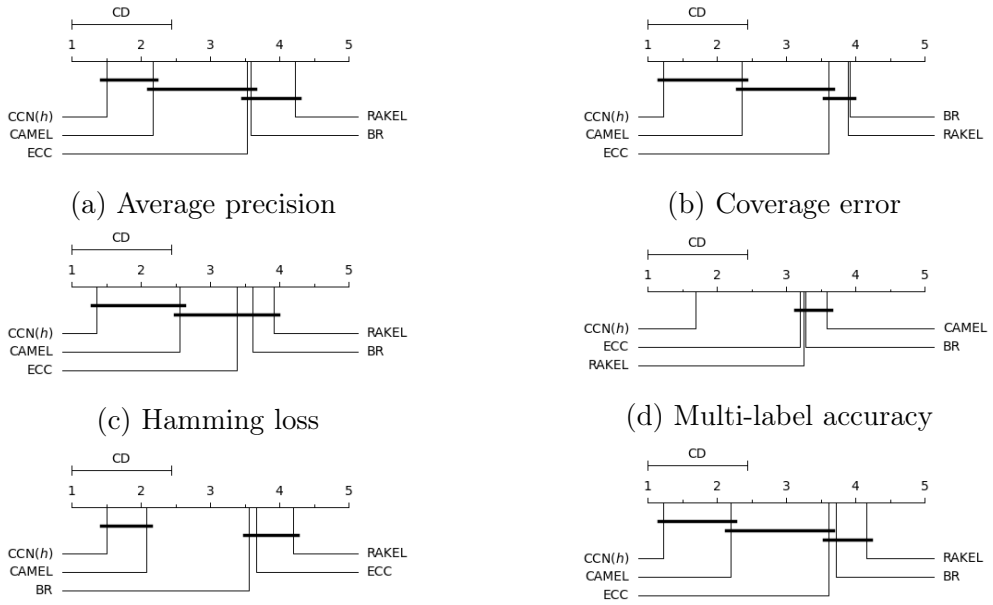
4. the remaining 10 are text classification datasets [92, 96, 114, 121].[8]

Furthermore, as it can be seen from Table 4.1, they differ significantly both in the number of data points/classes (columns D and L) and in the characteristics of the associated sets of constraints. Indeed, there are datasets having just a few (one)/many constraints (column C), involving a few/many classes in the head (column H/L) and in the body, either positively or negatively (columns B, B^+ and B^-).

As in the HMC experiments, I applied the same preprocessing to all the datasets. All the categorical features were transformed using one-hot encoding. The missing values were replaced by their mean in the case of numeric features and by a vector of all zeros in the case of categorical ones. All the features were standardized. As in the HMC experiments, I built h as a feedforward neural network with two hidden layers and ReLU non-linearity. To prove the robustness of CCN(h), I kept all the hyperparameters fixed except the hidden dimension used for each dataset, which is given in Appendix A.2. Such hidden dimensions were optimized over the validation sets. In all experiments, the loss was minimized using Adam optimizer with batch size equal to 4, learning rate equal to 10^{-4}, and patience 20 ($\beta_1 = 0.9, \beta_2 = 0.999$). Since some datasets have very few data points, I set the dropout rate equal to 80% and the weight decay equal to 10^{-4}. As for the HMC case, I retrained CCN(h) on both training and validation data for the same number of epochs, as the early stopping procedure determined was optimal in the first pass. For each dataset, I run the models 10 times, and the average for each of the metrics is reported in Tables 4.2 and 4.3. For ease of presentation, the standard deviations are omitted, which for CCN(h) are in the range $[2.9 \times 10^{-17}, 8.0 \times 10^{-3}]$, proving that it is a very stable model. ECC, BR, and RAKEL were implemented using scikit-multilearn [118] by deploying the logistic regression model as the base classifier. Regarding CAMEL, I used the publicly available authors' implementation,[9] with the hyperparameters suggested by the authors. For all the models, I got results comparable to the ones reported in the work by [41].

As it can be seen in Tables 4.2 and 4.3, CCN(h) has the highest number of wins in all metrics. Indeed, as it can be seen in Table 4.4, CCN(h) has the best average ranking in all metrics. I also verified the statistical significance of the results—as advised by [27]—by performing the Friedman test for each metric. As reported in Table 4.4, the Friedman test returned p-value < 0.01 for each metric; I could thus proceed with the post-hoc Nemenyi tests, and the resulting critical diagrams are reported in Figure 4.9. In each diagram, the groups of methods that do not differ significantly (significance

[8] Link to datasets with constraints: https://github.com/EGiunchiglia/CCN/tree/master/data/
[9] Link: https://github.com/hinanmu/CAMEL



(a) Average precision

(b) Coverage error

(c) Hamming loss

(d) Multi-label accuracy

(e) One-error

(f) Ranking loss

Figure 4.9: Critical diagram for each metric obtained with the post-hoc Nemenyi test.

level 0.05) are connected through an horizontal line. According to the Nemenyi test, CCN(h)'s performance differs significantly to the performance of all the other models but CAMEL in terms of average precision, coverage error, Hamming loss, one-error and ranking loss, while it differs significantly to all the models (including CAMEL) in terms of multi-label accuracy. As in the HMC case, I then followed the guidelines given by [27] and [6] and performed the Wilcoxon test to compare the difference between CAMEL and CCN(h). As reported in the last column of Table 4.4, the performances of CCN(h) and CAMEL differ significantly for all metrics but one-error, thus further confirming the superiority of the proposed model.

4.5.3 Ablation Studies

As in the HMC case, in order to analyze the impact of both CM and CLoss, I compared the performance of CCN(h) against the performance of h^+, that is, h with the enforcement of the constraints done as a post-processing step, and h + CM, that is, h with CM built on top. Both these models were trained using the standard binary cross-entropy loss. The results of the ablation studies for each metric and each dataset are given in Tables 4.5 and 4.6, while the average ranking for each metric can be found in Table 4.7. As it can be seen in Table 4.7, CCN(h) has the highest average ranking

Model	Arts	Bibtex	Business	Cal500	Delicious	Emotions	Enron	Genbase	Image
					Average precision (↑)				
CCN(h)	**0.623**	**0.586**	**0.904**	**0.520**	**0.347**	**0.800**	**0.704**	**0.996**	**0.807**
h^+	**0.623**	0.585	0.903	0.519	**0.347**	0.796	**0.704**	**0.996**	0.800
h+CM	**0.623**	0.580	0.902	0.519	0.344	**0.800**	**0.704**	0.978	**0.807**
					Coverage error (↓)				
CCN(h)	**0.172**	**0.105**	**0.065**	**0.734**	**0.520**	0.315	**0.217**	**0.016**	**0.187**
h^+	**0.172**	0.106	0.073	**0.734**	**0.520**	0.319	**0.217**	0.017	0.194
h+CM	0.173	0.108	0.072	**0.734**	**0.520**	**0.311**	**0.217**	0.020	**0.187**
					Hamming loss (↓)				
CCN(h)	**0.054**	**0.130**	**0.023**	**0.136**	**0.018**	**0.197**	**0.046**	**0.001**	0.172
h^+	**0.054**	**0.130**	0.026	**0.136**	**0.018**	0.201	**0.046**	**0.001**	0.172
h+CM	**0.054**	**0.130**	0.026	**0.136**	**0.018**	0.198	**0.046**	0.003	**0.169**
					Multi-label accuracy (↑)				
CCN(h)	**0.238**	**0.272**	**0.601**	**0.203**	0.095	**0.534**	**0.395**	**0.986**	**0.488**
h^+	**0.238**	**0.272**	**0.601**	**0.203**	**0.096**	0.512	**0.395**	**0.986**	0.475
h+CM	0.237	0.270	0.599	**0.203**	**0.096**	0.519	**0.395**	0.935	0.487
					One-error (↓)				
CCN(h)	**0.475**	**0.376**	**0.093**	**0.113**	**0.357**	**0.273**	**0.235**	**0.000**	**0.296**
h^+	0.476	0.384	**0.093**	**0.113**	**0.357**	0.283	0.238	**0.000**	0.307
h+CM	**0.475**	0.385	**0.093**	**0.113**	0.362	0.276	0.237	0.004	0.297
					Ranking loss (↓)				
CCN(h)	**0.115**	**0.058**	**0.030**	**0.173**	**0.110**	0.161	**0.076**	**0.003**	**0.159**
h^+	0.116	0.059	0.034	**0.173**	**0.110**	0.167	**0.076**	**0.003**	0.169
h+CM	0.116	0.060	0.034	**0.173**	**0.110**	**0.159**	**0.076**	0.006	0.160

Table 4.5: Results of the ablations studies. The best results are in bold.

for all metrics. Furthermore, I check the statistical significance of our results through the Wilcoxon test, whose results are reported in the last two columns of Table 4.7. As it can be seen, CCN(h) performs significantly better than h^+ for all metrics but multi-label accuracy. On the other hand, CCN(h) performs significantly better than $h + \text{CM}$ for all metrics but one-error.

4.6 Overview of the Results and Discussion

In this chapter, I introduced and dealt with LCMC problems: MC problems with hard constraints expressed as normal logic rules. For such class of problems I proposed CCN(h), a novel model with four distinguishing features: (i) its predictions are always coherent with the given constraints, (ii) it can be implemented on GPUs using standard libraries, (iii) it extends C-HMCNN(h), and thus outperforms the state-of-the-art

Model	Medical	Rcv1S1	Rcv1S2	Rcv1S3	Rcv1S4	Rcv1S5	Science	Scene	Yeast
					Average precision (↑)				
CCN(h)	0.866	**0.642**	**0.666**	**0.647**	0.675	**0.560**	**0.603**	**0.868**	**0.768**
h^+	**0.868**	0.639	0.662	0.645	0.663	0.548	**0.603**	0.864	0.765
h+CM	0.864	0.630	0.663	0.640	**0.682**	**0.560**	0.602	0.867	0.767
					Coverage error (↓)				
CCN(h)	**0.035**	**0.092**	**0.089**	0.103	**0.080**	**0.107**	**0.131**	**0.077**	**0.452**
h^+	0.037	0.093	0.090	**0.102**	0.089	0.112	0.135	0.081	0.455
h+CM	0.037	**0.092**	0.090	0.103	0.089	0.115	0.135	0.078	0.453
					Hamming loss (↓)				
CCN(h)	**0.013**	**0.026**	**0.022**	**0.024**	**0.019**	**0.025**	**0.031**	**0.092**	**0.196**
h^+	0.015	0.027	0.023	0.025	0.022	0.027	0.032	**0.092**	**0.196**
h+CM	0.015	0.027	0.023	0.025	0.020	**0.025**	0.032	**0.092**	**0.196**
					Multi-label accuracy (↑)				
CCN(h)	0.589	**0.296**	**0.310**	0.303	**0.324**	0.275	0.255	**0.607**	0.480
h^+	**0.591**	**0.296**	0.309	**0.305**	0.323	**0.277**	**0.257**	0.603	**0.487**
h+CM	0.587	0.283	0.306	0.301	0.321	0.257	0.250	0.604	0.482
					One-error (↓)				
CCN(h)	**0.181**	**0.413**	**0.389**	**0.405**	0.379	**0.402**	**0.494**	**0.224**	0.234
h^+	**0.181**	**0.413**	0.396	**0.405**	0.394	0.420	**0.494**	0.227	0.234
h+CM	0.185	0.434	0.391	0.407	**0.357**	0.407	0.496	0.226	**0.231**
					Ranking loss (↓)				
CCN(h)	**0.024**	**0.036**	**0.035**	**0.046**	**0.035**	**0.046**	**0.094**	**0.073**	**0.172**
h^+	0.026	**0.036**	0.036	**0.046**	0.039	0.049	0.097	0.077	**0.172**
h+CM	0.025	0.037	0.037	0.047	0.039	0.049	0.096	0.074	0.174

Table 4.6: Results of the ablation studies. The best results are in bold.

HMC models on HMC problems, and (iv) it outperforms the state-of-the-art MC models on 16 LCMC problems obtained adding automatically extracted constraints to well-known MC problems.

The above results open the path to many different lines of research. In general, the idea at the base of this chapter is to (i) incorporate the constraints in neural networks models guaranteeing their coherency, and (ii) exploit the constraints to improve performance. Here, I focused on MC problems with hard logical constraints on the output, but indeed the same idea can be applied also to other problems, and it could be interesting to investigate (i) whether it is possible to impose even more expressive constraints (e.g., involving also the input), (ii) how to incorporate both soft and hard constraints, and (iii) how to impose hard constraints on regression, binary classification, and multi-class classification problems.

Metric	Average ranking			Wilcoxon test	
	CCN(h)	h^+	h+CM	CCN(h) vs. h^+	CCN(h) vs. h+CM
Average precision	**1.39**	2.28	2.33	✓✓	✓
Coverage error	**1.39**	2.36	2.25	✓✓	✓
Hamming loss	**1.53**	2.33	2.14	✓✓	✓
Multi-label accuracy	**1.69**	1.72	2.58		✓✓
One-error	**1.44**	2.31	2.25	✓✓	✓
Ranking loss	**1.33**	2.28	2.39	✓✓	✓✓

Table 4.7: Average ranking for each metric and model, and results of the Wilcoxon test: ✓ (resp., ✓✓) is used to indicate that the Wilcoxon test returned p-value < 0.05 (resp., < 0.01).

Chapter 5

ROAD-R: a Case Study

As we have seen in the previous chapters, neural networks can be incredibly powerful at processing low-level inputs, however, they can exhibit unexpected behaviors, contradicting known requirements expressing background knowledge. This can have dramatic consequences, especially in safety-critical scenarios in which the constraints represent requirements that have to be satisfied. A possible solution to this problem is to develop models like CCN(h) which: (i) are able to learn from the requirements, and (ii) are guaranteed to be compliant with the requirements themselves. Unfortunately, the development of such models is massively delayed by the lack of suitable datasets. Indeed, in the datasets proposed in Chapter 4 Section 4.5, the constraints do not represent, from the perspective of the corresponding application domain, requirements that have to be necessarily satisfied.

In order to fill this gap, I thus created the ROad event Awareness Dataset with logical Requirements (ROAD-R), the first publicly available dataset for autonomous driving with requirements expressed as logical constraints of the form

$$l_1 \vee l_2 \vee \cdots \vee l_n \,, \tag{5.1}$$

where $n \geq 1$, and each l_i is either a negative label $\neg A$ or a positive label A. The constraint (5.1) asserts that each prediction must have at least one of the positive or one of the negative labels in $l_1, \ldots, l_n$. Thus, every condition on the labels expressible in propositional logic (and thus also with hierarchical and normal logic constraints, which are strictly less expressive) can be formulated as a set of constraints of the form (5.1).

ROAD-R extends the ROAD dataset [111], which consists of 22 relatively long ($\sim$ 8 minutes each) videos annotated with *road events*. A road event corresponds to a tube, i.e., a sequence of frame-wise bounding boxes linked in time. Each bounding box

Agents	**Actions**	**Locations**
Pedestrian	Move away	AV lane
Car	Move towards	Outgoing lane
Cyclist	Move	Outgoing cycle lane
Motorbike	Brake	Incoming lane
Medium vehicle	Stop	Incoming cycle lane
Large vehicle	Indicating left	Pavement
Bus	Indicating right	Left pavement
Emergency vehicle	Hazards lights on	Right pavement
AV traffic light	Turn left	Junction
Other traffic light	Turn right	Crossing location
	Overtake	Bus stop
	Wait to cross	Parking
	Cross from left	
	Cross from right	
	Crossing	
	Push object	
	Red traffic light	
	Amber traffic light	
	Green traffic light	

Table 5.1: ROAD labels.

is labeled with a subset of the 41 labels specified in Table 5.1. The goal is to predict the set of labels associated to each bounding box. I manually annotated ROAD-R with 243 constraints, each expressing common knowledge and verified to hold for each bounding box. A typical constraint is thus "a traffic light cannot be red and green at the same time", while there are no constraints like "pedestrians should cross at crossings", which should always be satisfied in theory, but which might not be in real-world scenarios.

Given ROAD-R, I considered 6 current state-of-the-art (SOTA) models, and I showed that they are not able to learn the requirements just from the data points, as at least 90% of their predictions violate the constraints. Then, I faced the problem of how to leverage the additional knowledge provided by constraints with the goal of (i) improving their performance, measured by the frame mean average precision (f-mAP) at intersection over union (IoU) thresholds 0.5 and 0.75; see, e.g., [60, 71]), and (ii) guaranteeing that they are compliant with the constraints. To achieve the above two goals, I propose the following new models:

1. **CL models**, i.e., models with a *constrained loss* allowing them to learn from

the requirements during training time,

2. **CO models**, i.e, models with a *constrained output* enforcing the requirements on the output at inference time, and

3. **CLCO models**, i.e., models with both a constrained loss and a constrained output.

As I will explain more in detail later, the loss in the CL models is obtained by adding the semantic based regularization explained in Chapter 2 (Section 2.3), while the post-processing step in the CO models is obtained by modelling the problem of finding a satisfying assignment of the constraints given the initial set of labels as a weighted partial maximum satisfiability problem. Notice that I could not use CCN(h) in this setting, as the constraints are expressed in full propositional logic, and CCN(h) works with stratified propositional normal rules. How to extend CCN(h) in order to deal with arbitrary propositional logic constraints is left for future work. For this work, I consider three different ways to build CL (resp., CO, CLCO) models, and thus I obtain 9×6 different models obtained by equipping the 6 current SOTA models with a constrained loss and/or a constrained output, and I show that it is always possible to

1. improve the performance of each SOTA model, and

2. be compliant with (i.e., strictly satisfy) the constraints.

Overall, the best performing model (for IoU = 0.5 and also IoU = 0.75) is CLCO-RCGRU, i.e., the SOTA model RCGRU equipped with both constrained loss and constrained output: CLCO-RCGRU (i) always satisfies the requirements and (ii) has f-mAP = 31.81 for IoU = 0.5, and f-mAP = 17.27 for IoU = 0.75. RCGRU, (i) produces predictions that violate the constraints at least 92% of the times, and (ii) has f-mAP = 30.78 for IoU = 0.5 and f-mAP = 15.98 for IoU = 0.75.

The rest of this chapter is organized as follows. First, I give an overview of the problem setup (Section 5.1), then, I present ROAD-R (Section 5.2), followed by the evaluation of the SOTA models (Section 5.3) and of the SOTA models incorporating the requirements (Section 5.4) on ROAD-R. I end the chapter with some qualitative examples of violations (Section 5.5), and then an overview of the presented results and outlook (Section 5.6).

Figure 5.1: Example of violation of $\neg\text{RedTL} \vee \neg\text{GreenTL}$.

5.1 Problem Setup

In ROAD, the detection of road events requires the following tasks: (i) identify the bounding boxes in each frame, (ii) associate with each bounding box a set of labels, and (iii) form a tube from the identified bounding boxes with the same labels.

Here, I focus on the second task, i.e., the association of the labels to each bounding box. Thus, I model the problem as a *multi-label classification problem with requirements* $(\mathcal{P}, \Pi)$, where Π is a set of requirements having form (5.1), and $\mathcal{P} = (\mathcal{A}, \mathcal{X})$ is a MC problem in which:

1. $\mathcal{X}$ is a finite set of pairs (x, y) where each x corresponds to a bounding box, and and $y \subseteq \mathcal{A}$ is the *ground truth* of x, and

2. $\mathcal{A}$ is the set of labels that can be associated to each bounding box. Such labels can represent (i) the agent performing the actions in the box, (ii) the actions being performed, and (iii) the locations where the actions take place. A detailed description of each label can be found in Table 5.2.

Notice that the ground truth y associated with a data point x characterizes both the *positive* and the *negative labels* associated with x, defined to be y and $\{\neg A : A \in \mathcal{A} \setminus y\}$, respectively. Further, as in the previous chapters, a *model* m for $\mathcal{P}$ is a function $m(\cdot, \cdot)$ mapping every label A and every datapoint $x \in \mathbb{R}^D$ to $[0, 1]$, and a datapoint x is *predicted* by m to have label A if its *output value* $m(A, x)$ is greater than a user-defined *threshold* $\theta \in [0, 1]$. Given a model m for $\mathcal{P}$, the *prediction of* m *for* x is the set $\{A : A \in \mathcal{A}, m(A, x) > \theta\} \cup \{\neg A : A \in \mathcal{A}, m(A, x) \leq \theta\}$ of positive and negative labels. For ease of presentation, from now on, I use a set-based notation,

Label name	Description	Abbrv.
Pedestrian	A person including children	Ped
Car	A car up to the size of a multi-purpose vehicle	Car
Cyclist	A person is riding a push/electric bicycle	Cyc
Motorbike	Motorbike, dirt bike, scooter with 2/3 wheels	Mobike
Medium vehicle	Vehicle larger than a car, such as van	MedVeh
Large vehicle	Vehicle larger than a van, such as a lorry	LarVeh
Bus	A single or double-decker bus or coach	Bus
Emergency vehicle	Ambulance, police car, fire engine, etc.	EmVeh
AV traffic light	Traffic light related to the AV lane	TL
Other traffic light	Traffic light not related to the AV lane	OthTL
Move away	Agent moving in a direction that increases the distance between Agent and AV	MovAway
Move towards	Agent moving in a direction that decreases the distance between Agent and AV	MovTow
Move	Agent moving perpendicular to the traffic flow or vehicle lane	Mov
Brake	Agent is slowing down, vehicle braking lights are lit	Brake
Stop	Agent stationary but in ready position to move	Stop
Indicating left	Agent indicating left by flashing left indicator light, or using a hand signal	IncatLeft
Indicating right	Agent indicating right by flashing right indicator light, or using a hand signal	IncatRht
Hazard lights on	Hazards lights are flashing on a vehicle	HazLit
Turn left	Agent is turning in left direction	TurLft
Turn right	Agent is turning in right direction	TurRht
Overtake	Agent is moving around a slow-moving user, often switching lanes to overtake	Ovtak
Wait to cross	Agent on a pavement, stationary, facing in the direction of the road	Wait2X
Cross road from left	Agent crossing road, starting from the left and moving towards the right of AV	XingFmLft
Cross road from right	Agent crossing road, starting from the right and moving towards the left of AV	XingFmRht
Crossing	Agent crossing road	Xing
Push object	Agent pushing object, such as trolley or pushchair, wheelchair or bicycle	PushObj
Red traffic light	Traffic light with red light lit	Red
Amber traffic light	Traffic light with amber light lit	Amber
Green traffic light	Traffic light with green light lit	Green
AV lane	Agent in same road lane as AV	VehLane
Outgoing lane	Agent in road lane that should be flowing in the same direction as AV lane	OutgoLane
Outgoing cycle lane	Agent in the cycle lane that should be flowing in the same direction as AV	OutgoCycLane
Incoming lane	Agent in road lane that should be flowing in the opposite direction as AV lane	IncomLane
Incoming cycle lane	Agent in the cycle lane that should be flowing in the opposite direction as AV	IncomCycLane
Pavement	A pavement that is perpendicular to the movement of the AV	Pav
Left pavement	Pavement to the left side of AV	LftPav
Right pavement	Pavement to the right side of AV	RhtPav
Junction	Road linked	Jun
Crossing location	A marked section of road for cross, such as zebra or pelican crossing	XingLoc
Bus stop	A marked bus stop on road, or a section of pavement next to a bus stop sign	BusStop
Parking	A marked parking area on the side of the road	Parking

Table 5.2: Labels with descriptions and abbreviations [111].

and express the constraint (5.1) as the set of its positive and negative labels, i.e., as:

$$\{l_1, l_2, \ldots, l_n\}. \tag{5.2}$$

Consider an MC problem with requirements $(\mathcal{P}, \Pi)$. Each constraint delimits the set of predictions that can be associated with each data point by ruling out those that violate it. A prediction p is *admissible* if each constraint r in Π is *satisfied* by p, i.e., in set-based notation, if $p \cap r \neq \emptyset$. A model m for $\mathcal{P}$ *satisfies* (resp., *violates*) the constraints on a data point x if the prediction of m for x is (resp., is not) admissible.

Example 5.1.1. The requirement that a traffic light cannot be both red and green corresponds to the constraint {¬RedTL, ¬GreenTL}. Any prediction with {RedTL,

GreenTL} is non-admissible. An example of such prediction is shown in Figure 5.1.

Given an MC problem with requirements, it is possible to take advantage of the constraints in two different ways:

- they can be exploited during learning to teach the model the background knowledge that they express, and
- they can be used as post-processing to turn a non-admissible prediction into an admissible one.

Models in the first and second category are said to have a *constrained loss* (CL) and *constrained output* (CO) respectively. Constrained loss models have the advantage that the constraints are deployed during the training phase, and this should result in models (i) with a higher understanding of the problem and a better performance, but still (ii) with no guarantee that no violations will be committed. On the other hand, constrained output models (i) do not exploit the additional knowledge during training, but (ii) are guaranteed to have no violations in the final outputs. These two options are not mutually exclusive (i.e., can be used together), and which one is to be deployed depends also on the extent to which a system is available. For instance, there can be companies that already have their own models (which can be black boxes) and want to make them compliant with a set of requirements without modifying the model itself. On the other hand, the exploitation of the constraints in the learning phase can be an attractive option for those who have a good knowledge of the model and want to further improve it.

5.2 ROAD-R

ROAD-R extends the ROAD dataset[1] [111] by introducing a set Π of 243 constraints that specify the space of admissible outputs. The requirements have been manually specified following three steps:

1. an initial set of constraints Π_1 was manually created,
2. a subset $\Pi_2 \subset \Pi_1$ was retained by eliminating all those constraints that were entailed by the others,
3. the final subset $\Pi \subset \Pi_2$ was retained by keeping only those requirements that were always satisfied by the ground-truth labels of the entire ROAD-R dataset.

Statistics	
$\lvert\mathcal{C}\rvert$	41
$\lvert\Pi\rvert$	243
$\text{avg}_{r\in\Pi}(\lvert r\rvert)$	2.86
$\lvert\{A \in \mathcal{C} : \exists r \in \Pi. A \in r\}\rvert$	41
$\lvert\{A \in \mathcal{C} : \exists r \in \Pi. \neg A \in r\}\rvert$	38
$\min_{A\in\mathcal{C}}(\lvert\{r \in \Pi : \{A, \neg A\} \cap r \neq \emptyset\}\rvert)$	2
$\text{avg}_{A\in\mathcal{C}}(\lvert\{r \in \Pi : \{A, \neg A\} \cap r \neq \emptyset\}\rvert)$	16.95
$\max_{A\in\mathcal{C}}(\lvert\{r \in \Pi : \{A, \neg A\} \cap r \neq \emptyset\}\rvert)$	31

Table 5.3: Constraint statistics. All the constraints have between 2 and 15 positive and negative labels, with an average of 2.86. All (resp., 38 of) the labels appear positively (resp., negatively) in Π. Each label appears either positively or negatively between 2 and 31 times in Π, with an average of 16.95.

Tables 5.3 and 5.4 give a high-level description of the properties of the set Π of constraints. Notice that, of the 243 constraints, there are two in which all the labels are positive (expressing that there must be at least one agent and that every agent but traffic lights has at least one location), and 214 in which all the labels are negative (expressing mutual exclusion between two labels). All the constraints with more than two labels have at most one negative label, as they express a one-to-many relation between actions and agents (like "if something is crossing, then it is a pedestrian or a cyclist").

Constraints like "pedestrians should cross at crossings", which might not be satisfied in practice, are not included. The list with all the 243 requirements, with their natural language explanations, is in Appendix B.1, Tables B.1, B.2, and B.3. Notice that the 243 requirements restrict the number of admissible prediction to $4985868 \sim 5 \times 10^6$, thus ruling out $(2^{41} - 4985868) \sim 10^{12}$ non-admissible predictions.[2] In principle, the set of admissible predictions can be further reduced by adding other constraints. Indeed, the 243 requirements are not guaranteed to be complete from every possible point of view: as standard in the software development cycle, the requirement specification process deeply involves the stakeholders of the system (see, e.g., [112]). For example, I decided not to include constraints like "it is not possible to both move towards and move away", which were not satisfied by all the data points because of errors in the ground truth labels. In these cases, I decided to dismiss the constraint in order to

[1] ROAD is available at: https://github.com/gurkirt/road-dataset.

[2] The number of admissible predictions has been computed with relsat: https://github.com/roberto-bayardo/relsat/.

n	$\lvert\Pi_n\rvert$	$\text{avg}_{r\in\Pi_n}(\lvert r \cap \overline{\mathcal{C}}\rvert)$	$\text{avg}_{r\in\Pi_n}(\lvert r \cap \mathcal{C}\rvert)$
2	215	1.995	0.005
3	5	1	2
7	1	1	6
8	6	1	7
9	6	1	8
10	1	0	10
12	1	1	11
14	1	0	14
15	7	1	14
Total	243	1.87	0.96

Table 5.4: Constraint statistics. Π_n is the set of constraints r in Π with $|r| = n$, i.e., with n positive and negative labels. $\overline{\mathcal{C}} = \{\neg A : A \in \mathcal{C}\}$. Each row shows the number of rules r with $|r| = n$, and the average number of negative and positive labels in such rules.

maintain (i) consistency between the knowledge provided by the constraints and by the data points, and (ii) backward compatibility.

As an additional consideration, I underline that the effort of manually writing 243 constraints (i) is negligible when compared to the effort of manually annotating the 22 videos, and (ii) can improve such last process, e.g., allowing to prevent errors in the annotation of the data points and/or speeding up the labelling process.

5.3 ROAD-R and SOTA Models

As a first step, I ran 6 SOTA temporal feature learning architectures as part of a 3D-RetinaNet model [111] (with a 2D-ConvNet backbone made of Resnet50 [52]) for event detection and evaluated to which extent constraints are violated. In particular, I considered the following SOTA models:

1. 2D-ConvNet (C2D) [131]: a Resnet50-based architecture with an additional temporal dimension for learning features from videos. The extension from 2D to 3D is done by adding a pooling layer over time to combine the spatial features.

2. Inflated 3D-ConvNet (I3D) [12]: a sequential learning architecture extendable to any SOTA image classification model (2D-ConvNet based), able to learn continuous spatio-temporal features from the sequence of frames.

3. Recurrent Convolutional Network (RCN) [110]: a 3D-ConvNet model that relies on recurrence for learning the spatio-temporal features at each network level. During the feature extraction phase, RCNs exploit both 2D convolutions across the spatial domain and 1D convolutions across the temporal domain.

4. Random Connectivity Long Short-Term Memory (RCLSTM) [58]: an updated version of LSTM in which the neurons are connected in a stochastic manner, rather than fully connected. In our case, the LSTM cell is used as a bottleneck in Resnet50 for learning the features sequentially.

5. Random Connectivity Gated Recurrent Unit (RCGRU) [58]: an alternative version of RCLSTM where the GRU cell is used instead of the LSTM one. GRU makes the process more efficient with fewer parameters than the LSTM.

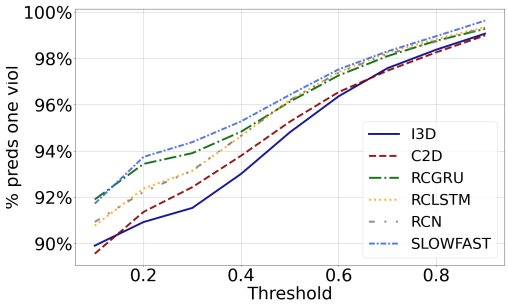
6. SlowFast [40]: a 3D CNN architecture that contains both slow and fast pathways for extracting the sequential features. A Slow pathway computes the spatial semantics at low frame rate while a Fast pathway processes high frame rate for capturing the motion features. Both of the pathways are fused in a single architecture by lateral connections.

I trained 3D-RetinaNet[3] using the same hyperparameter settings for all the models: (i) batch size equal to 4, (ii) sequence length equal to 8, and (iii) image input size equal to 512×682. All the models were initialized with the Kinetics pre-trained weights. An SGD optimizer [68] with step learning rate was used. The initial learning rate was set to 0.0041 for all the models except SlowFast, for which it was set to 0.0021 due to the diverse nature of slow and fast pathways. All the models were trained for 30 epochs and the learning rate was made to drop by a factor of 10 after 18 and 25 epochs. The machine used for the experiments has 64 CPUs (2.2 GHz each) and 4 Titan RTX GPUs having 24 GB of RAM each.

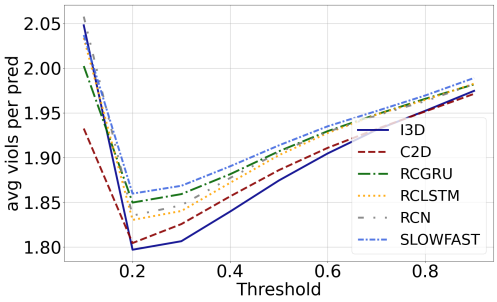
To measure the models' performance, I used the *frame mean average precision* (f-mAP), which is the standard metric used for action detection (see, e.g., [60, 71]), with IoU threshold equal to 0.5 and 0.75, indicated as f-mAP@0.5 and f-mAP@0.75, respectively. The results for the SOTA models at IoU threshold 0.5 and 0.75 are reported in Table 5.5, column "SOTA".

To measure the extent to which each system violates the constraints, I used the following metrics:

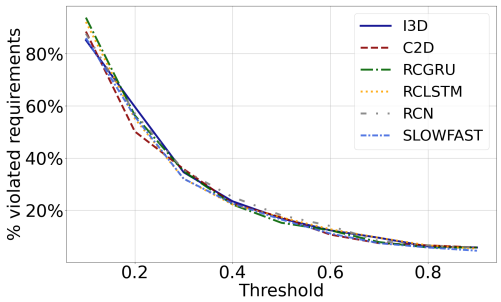
[3] https://github.com/gurkirt/3D-RetinaNets.



(a) Percentage of predictions violating at least one constraint.



(b) Average number of violations committed per prediction.



(c) Percentage of constraints violated at least once.

Figure 5.2: ROAD-R and SOTA models. In the x-axis, there is the threshold $\theta \in [0.1, 0.9]$, step 0.1.

- the percentage of non-admissible predictions,
- the average number of violations committed per prediction, and
- the percentage of constraints violated at least once,

while varying the threshold θ from 0.1 to 0.9 with step 0.1. The results are in Figure 5.2, where (to improve readability) I do not plot the values corresponding to $\theta = 0.0$ and $\theta = 1.0$. For $\theta = 0.0$ (resp., $\theta = 1.0$), all the predictions are positive (resp., negative), and thus the corresponding values are (in order) 100%, 214, and 214/243 (resp., 100%, 2, and 2/243).

Consider the results in Table 5.5, column "SOTA", and in Figure 5.2. First, note that the performances are not an indicator of the ability of the model to satisfy the constraints. Indeed, higher f-mAPs do not correspond to lower trends in the plots of Figure 5.2b. For example, RCGRU performs better than C2D for both IoU = 0.5 and IoU = 0.75, however, its curve is above C2D's in both Figs. 5.2a and 5.2b. Then, note that the percentage of non-admissible predictions is always very high for every model:

at its minimum, for $\theta = 0.1$, more than 90% of the predictions are non-admissible, and this percentage reaches 99% for $\theta = 0.9$ (see Figure 5.2a). In addition, most predictions violate roughly two constraints, as shown by Figure 5.2b. Considering that an autonomous vehicle setting is being considered, such results are critical: one of the constraints that is violated by all the baseline models is $\{\neg\text{RedTL}, \neg\text{GreenTL}\}$, corresponding to predictions according to which there is a traffic light with both the red and the green lights on. Figure 5.1 shows an image where such a prediction is made by C2D. Section 5.5 contains images with all the models making predictions violating $\{\neg\text{RedTL}, \neg\text{GreenTL}\}$ and also other constraints.

5.4 ROAD-R and CL, CO, and CLCO Models

I now show how it is possible to build CL, CO and CLCO models. In particular, I show how to equip the 6 considered SOTA models with constrained a loss and/or a constrained output. As anticipated at the beginning of the chapter, I introduce (i) three different methods to build the constrained loss, (ii) three different methods to obtain the constrained output, and (iii) three combinations of constrained loss and constrained output. Thus, I get 9 models for each SOTA model, for a total of 54. In order to get an overall view of the performance gains produced by each method, I also report the average ranking of the 9 proposed methods and SOTA [27], computed as follows:

1. for each row in Table 5.5, I rank the performances of the 9 CL, CO and CLCO models and of the SOTA model separately: the best performing model gets the rank 1, the second best gets rank 2, etc., and in case of ties, the rank is split (e.g., the assigned rank is 1.5 if two models have the best performance), and

2. for each column, we take the average of the rankings computed in the previous step.

See Table 5.5 for f-mAP@0.5, f-mAP@0.75 and average rankings. In the table, for each row the best results are in bold. The details of the implemented models with constrained loss, constrained output, and both constrained loss and constrained output is given in the three paragraphs below.

Model	SOTA	Constrained Loss (CL)			Constrained Output (CO)			Constrained Loss and Output (CLCO)		
		Product	Gödel	Lukasiewicz	Basic	AP	AP×O	(P, AP×O)	(G, AP×O)	(L, AP×O)
C2D	27.57	27.90 (1)	27.97 (1)	27.57 (10)	27.41 (0.4)	27.67 (0.9)	27.69 (0.9)	27.90 (0.7)	**28.16** (0.8)	27.86 (0.9)
I3D	30.12	30.79 (10)	29.98 (10)	31.03 (10)	29.86 (0.4)	30.15 (0.5)	30.15 (0.5)	30.77 (0.9)	30.02 (0.7)	**31.21** (0.6)
RCGRU	30.78	30.67 (10)	29.57 (1)	31.78 (10)	30.56 (0.4)	30.78 (0.5)	30.78 (0.7)	30.79 (0.6)	29.83 (0.9)	**31.81** (0.5)
RCLSTM	30.49	30.48 (10)	30.88 (1)	31.63 (10)	30.09 (0.4)	30.51 (0.7)	30.51 (0.7)	30.50 (0.5)	30.90 (0.8)	**31.65** (0.9)
RCN	29.64	30.31 (10)	30.24 (10)	**31.02** (10)	29.39 (0.4)	29.80 (0.5)	29.82 (0.5)	30.34 (0.6)	30.28 (0.8)	**31.02** (0.5)
SlowFast	28.79	28.54 (10)	28.91 (1)	28.47 (10)	28.51 (0.5)	28.86 (0.9)	28.86 (0.9)	28.67 (0.9)	**28.98** (0.5)	28.56 (0.7)
Avg. Rank	7.08	5.75	5.50	4.25	9.50	5.75	5.42	4.42	4.50	**2.83**

Model	SOTA	Constrained Loss (CL)			Constrained Output (CO)			Constrained Loss and Output (CLCO)		
		Product	Gödel	Lukasiewicz	Basic	AP	AP×O	(P, AP×O)	(G, AP×O)	(L, AP×O)
C2D	12.66	13.04 (1)	12.90 (1)	12.30 (10)	12.62 (0.4)	12.72 (0.9)	12.72 (0.9)	**13.05** (0.8)	13.01 (0.8)	12.55 (0.9)
I3D	16.29	16.47 (10)	16.19 (10)	16.32 (10)	16.22 (0.5)	16.31 (0.8)	16.31 (0.8)	**16.48** (0.6)	16.20 (0.6)	16.42 (0.6)
RCGRU	15.98	16.57 (10)	15.39 (1)	17.26 (10)	15.97 (0.4)	16.02 (0.5)	16.03 (0.5)	16.63 (0.6)	15.63 (0.9)	**17.27** (0.6)
RCLSTM	16.64	16.14 (10)	15.65 (1)	16.34 (10)	16.38 (0.4)	**16.69** (0.5)	16.66 (0.7)	16.15 (0.7)	15.67 (0.8)	16.39 (0.8)
RCN	15.82	16.57 (10)	16.17 (10)	17.25 (10)	15.73 (0.4)	15.87 (0.8)	15.89 (0.5)	16.56 (0.5)	16.19 (0.9)	**17.26** (0.9)
SlowFast	14.40	14.84 (10)	15.38 (1)	13.80 (10)	14.33 (0.5)	14.47 (0.9)	14.47 (0.9)	14.94 (0.9)	**15.42** (0.5)	13.84 (0.7)
Avg. Rank	6.67	3.83	7.00	5.67	7.83	5.00	4.83	**3.17**	6.00	4.50

Table 5.5: f-mAP@0.5 (top table) and f-mAP@0.75 (bottom table) for the (i) current SOTA models; (ii) CL models, in parentheses the value of α; (iii) CO models, in parenthesis the threshold used to evaluate the admissibility of the predictions; and (iv) CLCO models, in parenthesis the threshold used to evaluate the admissibility of the predictions. **P**, **G**, and **L** stand for the Product, Gödel, and Lukasiewicz t-norm, respectively.

Constrained Loss. To constrain the loss, I take inspiration from the approaches proposed in [32, 31], and I train the models using the standard localization and classification losses, to which I add a regularization term, called Semantic Based Regularization term (for a detailed description of how to define it see Chapter 2 Section 2.3). This last term represents the degree of satisfaction of the constraints in Π and has the form:

$$\mathcal{L}_{\Pi} = \alpha \sum_{i=1}^{|\Pi|} (1 - t(r_i)),$$

where r_i represents the ith constraint in Π, $t(r_i)$ represents the fuzzy logic relaxation of r_i, and α is a hyperparameter ruling the weight of the regularization term (the higher α is, the more relevant the term corresponding to the constraints becomes, up to the limit case in which $\alpha \to \infty$ and the constraints become hard [31]). I considered $\alpha \in \{1, 10, 100\}$ and the three fundamental t-norms:

1. Product t-norm,
2. Gödel t-norm, and
3. Lukasiewicz t-norm

as fuzzy logic relaxations (see e.g., [84]). The best results for f-mAP@0.5 and f-mAP@0.75 while varying α are in Table 5.5, columns Product, Gödel, and Lukasiewicz. As can be seen, SOTA never achieves the best average ranking, even when compared with only the three CL methods. Of these, Lukasiewicz (for IoU = 0.5) and Product (for IoU = 0.75) have the best ranking, though for some model and IoU, the best performances are obtained with Gödel. In only one case (for RCLSTM at IoU = 0.75), the SOTA model performs better than the CL models. Also notice that the best performances are never obtained with $\alpha = 100$, and any significant reduction in the number of predictions violating the constraints is never obtained.

Constrained Output. I now consider the problem of how to correct a prediction p whose admissibility is evaluated at a given threshold θ. A simple solution of the problem is to assign a positive weight w_i (representing the cost of correcting the ith label in p) and then compute the admissible prediction q that minimizes $\sum_{i:p_i \neq q_i} w_i$. Indeed, if p is admissible, q does not differ from p, otherwise some of the positive (resp., negative) labels in p are flipped to negative (resp., positive) in q. However, no polynomial time algorithm able to solve the above problem is currently known.

Theorem 5.4.1. *Let $(\mathcal{P}, \Pi)$ be a MC problem with requirements. Let p be a prediction, and $w_i \in \mathbb{R}^+$ be the cost of correcting the ith label in p. For each positive d, determining the existence of an admissible prediction q such that $\sum_{i:p_i \neq q_i} w_i \leq d$ is an NP-complete problem.*

Proof. The problem of determining an admissible prediction q such that $\sum_{i:p_i \neq q_i} w_i \leq d$ is clearly in NP. The NP-hardness of our problem follows from the fact that it is a generalization of the DISTANCE-SAT problem, defined as the decision problem in which the input is (i) a propositional logic formula f, (ii) a partial interpretation $\mathcal{I}_P$ of f, and (iii) a integer $d \geq 0$, and the question to answer is whether there exists a model $\mathcal{I}$ of f such that $\mathcal{I}$ disagrees with $\mathcal{I}_P$ on at most d variables. Proposition 1 in [4] states that DISTANCE-SAT is NP-complete even assuming that the formula f is in CNF and that the interpretation $\mathcal{I}_P$ is complete. □

From a practical perspective, the problem of finding an admissible prediction q that minimizes the total correction cost of a given prediction p can be solved by considering the weighted partial maximum satisfiability (PMaxSAT) problem of a set of clauses[4] (see, e.g.,[70]) in which

1. each constraint in Π corresponds to a clause marked as hard, and
2. each positive and negative labels in p corresponds to a unit clause marked as soft with weight w_i.

Luckily, for weighted MaxSat problems, very efficient solvers are available. I used MaxHS [53], and running times were in the order of 10^{-3}s at most.

As expected, which labels are flipped depends on the policy used to assign the weights w_i, and in the following experiments, I tested the following three policies:

1. Basic, in which each w_i is equal to 1,
2. AP, in which each w_i is equal to the average precision AP_i of the ith label, and
3. AP×O, in which each $w_i = AP_i \times c_i$, where c_i is equal to (i) the output o_i of the model for the ith label if $o_i > \theta$, and (ii) $(1 - o_i)$, otherwise.

[4]In this setting, a *clause* is a disjunction of literals, and a literal is either a positive label or its negation, representing the corresponding negative label. A clause is *unit* if it consists of a single literal.

The first policy, by assigning all labels the same weight, returns the admissible prediction q with as few as possible labels flipped. The second and the third take into account the reliability of the output o_i of the model for the ith label. Notice that flipping the ith label amounts to changing its output value o_i from a value below the threshold to another value $f(o_i)$ above the threshold or vice versa. In all the experiments, I considered (i) $f(o_i) = \theta + \epsilon$, if $o_i < \theta$, and (ii) $f(o_i) = \theta - \epsilon$, otherwise. In the experiments I used $\epsilon = 10^{-3}$. In this way, assuming $o_i \leq \theta$ (i.e., the label is negatively predicted and then flipped positive), I expect $f(o_i)$ to be lower (but still higher than θ) than the output values of the non-flipped, positively predicted labels. It is done analogously for the case $o_i > \theta$. I tested the three policies with all the thresholds θ from 0.1 to 0.9 with step 0.1, and the resulting f-mAP@0.5 and f-mAP@0.75 for the best threshold are reported in Table 5.5, columns Basic, AP, and AP×O. Comparing the CO's performances with the current SOTA models', it is possible to see that Basic CO models always get worse performances, despite the final output being ensured to have as few as possible labels flipped. On the other hand, flipping the variables taking into account the average precision (columns AP and AP×O) (i) never leads to worse performances than the ones of the SOTA models, and (ii) for IoU = 0.75, correcting the output of RCLSTM with AP and AP×O gives the best and second best performance in the row. Notice that the differences in the performances between AP and AP×O are always negligible, AP×O being better than AP more often. The average rankings are in line with the above statements.

Constrained Loss and Output. Given the results presented in the previous paragraph, of the 9 possible combinations of a constrained loss and a constrained output, I consider only the ones with AP×O as constrained output. The results are shown in Table 5.5, last three columns. Given the constant improvement produced by the constrained output AP×O over the SOTA models discussed in the previous paragraph, the results of the CLCO models are not surprising: post-processing the output with AP×O policy produces again relatively small but almost always constant improvements over the corresponding CL models. The average rankings are in line with the above statements.

Considering the results in Table 5.5 all together, it can be seen that

1. constraining the output alone guarantees the compliance with the constraints, but improvements in the performances are constant but limited,

2. constraining the loss alone does not guarantee the satisfaction of the requirements but can lead to non marginal improvements in the performances,

3. the best performances (the numbers in bold) are always obtained by constraining the output, and thus it is always possible to (i) improve the performance of each SOTA model, and (ii) guarantee to be compliant with the requirements,

4. on average, the best performances are obtained by CLCO models, as witnessed by the average rankings,

5. the best performing model (for IoU $= 0.5$ and IoU $= 0.75$) is CLCO-RCGRU, i.e., RCGRU with Lukasiewicz constrained loss and AP$\times$O constrained output: such model (i) is compliant with the constraints by construction, and (ii) has f-mAP = 31.81 for IoU = 0.5, and f-mAP = 17.27 for IoU = 0.75. RCGRU (without CL and CO) (i) produces predictions that violate the constraints at least 92% of the times, and (ii) has f-mAP = 30.78 for IoU = 0.5, and f-mAP = 15.98 for IoU = 0.75.

5.5 Qualitative Examples of Violations

In this section, I give some qualitative and visual examples of violations. In order to show different situations in which a non-admissible prediction occurs, I consider various requirements, and (for each of them) I display a non-admissible prediction made by each SOTA model. For all examples, I pick the threshold $\theta = 0.5$, as it is the most intuitive and used threshold in multi-label classification problems. In particular, I show the violations for:

1. $\{\neg\text{RedTL}, \neg\text{GreenTL}\}$ in Figure 5.3,

2. $\{\neg\text{TL}, \neg\text{OthTL}\}$ in Figure 5.4 ,

3. $\{\neg\text{Ped}, \neg\text{Cyc}\}$ in Figure 5.5,

4. $\{\neg\text{LeftPav}, \neg\text{RightPav}\}$ in Figure 5.6,

5. {Ped, Car, Cyc, Mobike, MedVeh, LarVeh, Bus, EmVeh, TL, OthTL} in Figure 5.7,

6. {TL, OthTL, VehLane, OutgoLane,OutgoCycLane, Jun IncomLane, IncomCycLane, Pav, LftPav, RhtPav, XingLoc, BusStop, Parking} in Figure 5.8, and

7. {Ped, Car, Cyc, Mobike, MedVeh, LarVeh, Bus, EmVeh, ¬MovAway} in Figure 5.9.

In the above list, there is at least one requirement with (i) all negative labels, (ii) all positive labels, and (iii) at least one positive and one negative label. Furthermore, at least one agent label, one action label, and one location label appear in at least one of the above requirements.

5.6 Overview of the Results and Outlook

In this chapter, I proposed a new dataset, called ROAD-R, for the task of learning with requirements. ROAD-R represents the first dataset with formally specified requirements for autonomous driving. Thanks to ROAD-R, I was able to show that SOTA models most of the times violate the given requirements, and thus, that a requirement specification step is as much needed in deep learning models development as it is in standard software development processes.

Further, I showed how it is possible to exploit the requirements to create models that are compliant with (i.e., strictly satisfy) the requirements while improving their performance. The solutions proposed in this chapter have the benefit that they can be plugged in existing systems, i.e., any model can be retrained using the loss of the CL models, and/or the post-processing step of the CO models can be added on top of any system, even when the system is a black box. This property should facilitate their widespread adoption in both industry and academia. However, the proposed approaches do not have the same ability of CCN(h) of delegating the predictions for one or more labels. One of the reasons behind CCN(h)'s ability (as seen in the previous chapter) is that the constraints are written as rules, which possess a head and a body, and thus the network can delegate the prediction for the label in the head to one of the labels in the body. However, if the constraints are written in CNF, it is no longer clear which labels should delegate and which should be delegated. While it is true that (assuming to allow for rules with negative literals and/or *False* in the head) any clause can always be rewritten as a rule, in order to be able to apply CCN(h) to ROAD-R I have to first answer the following open questions: (i) how can a rule with a negative literal in the head be mapped to the constraint layer? (ii) how can a rule with *False* in the head be mapped in the constraint layer? (iii) given a clause with more than one literal, which literal (including *False*) should be the head of the rule?

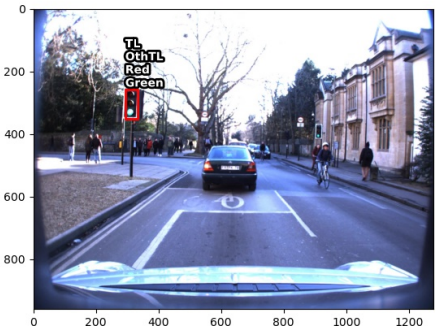
In the future, I will answer the above open questions, which will allow for the extension of CCN(h) to propositional logic constraints, and I will test the resulting extended CCN(h) on ROAD-R.

(a) I3D

(b) C2D

(c) RCGRU

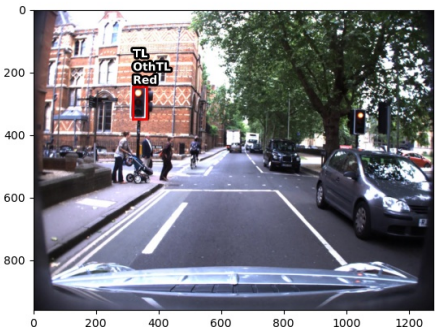
(d) RCLSTM



(e) RCN

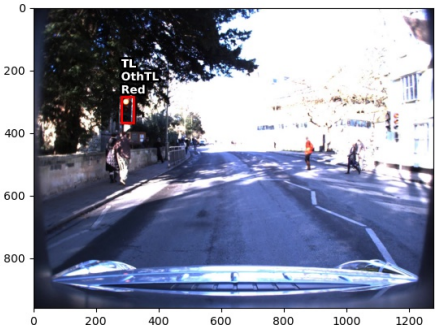
(f) SlowFast

Figure 5.3: Examples of violations of $\{\neg\text{RedTL}, \neg\text{GreenTL}\}$.

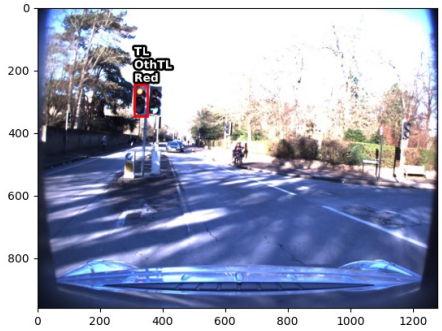


(a) I3D

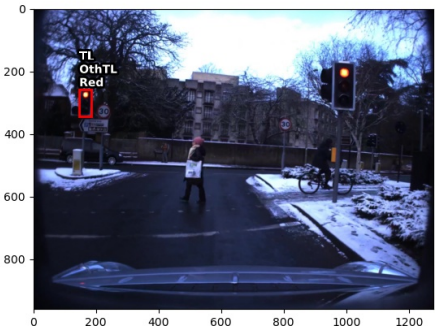
(b) C2D



(c) RCGRU



(d) RCLSTM

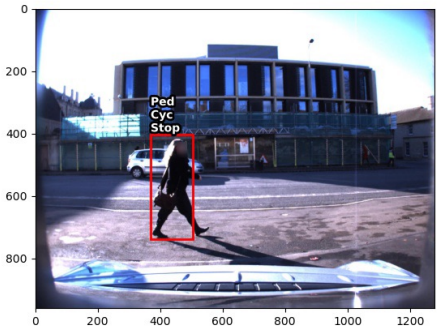


(e) RCN

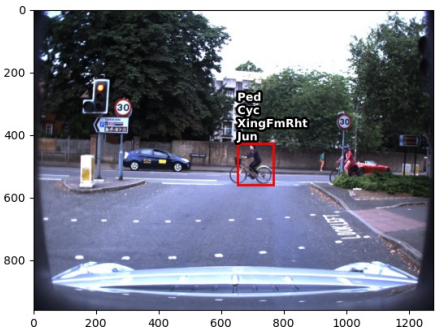
(f) SlowFast

Figure 5.4: Examples of violations of $\{\neg\text{TL}, \neg\text{OthTL}\}$.

(a) I3D



(b) C2D



(c) RCGRU

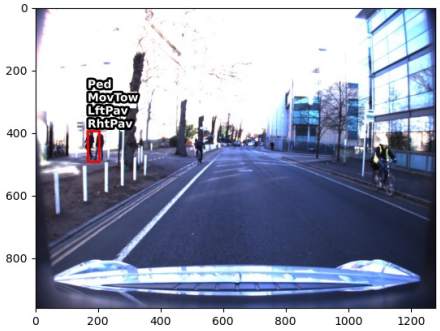
(d) RCLSTM

(e) RCN

(f) SlowFast

Figure 5.5: Examples of violations of $\{\neg\text{Ped}, \neg\text{Cyc}\}$.

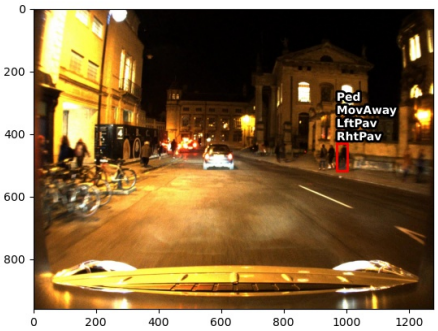
(a) I3D



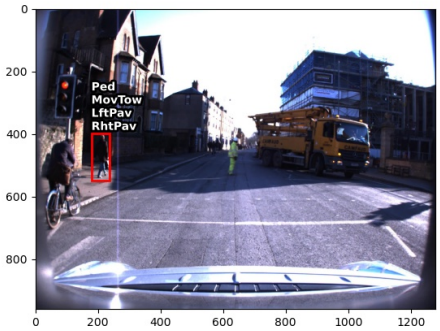
(b) C2D

(c) RCGRU

(d) RCLSTM



(e) RCN



(f) SlowFast

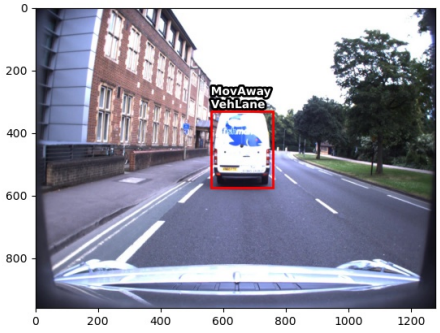
Figure 5.6: Examples of violations of $\{\neg\text{LeftPav}, \neg\text{RightPav}\}$.

(a) I3D

(b) C2D

(c) RCGRU

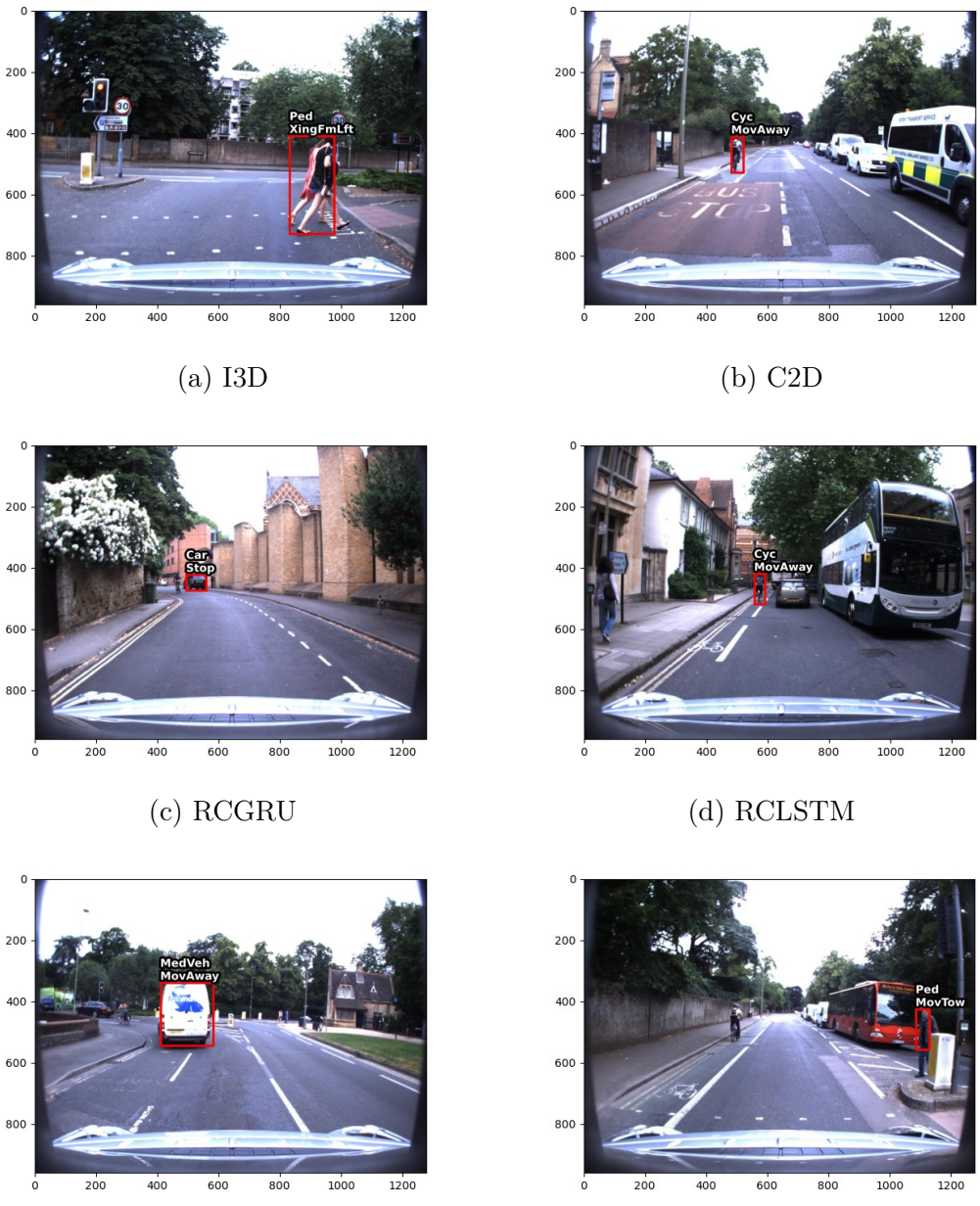
(d) RCLSTM



(e) RCN

(f) RCN

Figure 5.7: Examples of violations of {Ped, Car, Cyc, Mobike, MedVeh, LarVeh, Bus, EmVeh, TL, OthTL}.



(a) I3D

(b) C2D

(c) RCGRU

(d) RCLSTM

(e) RCN

(f) SlowFast

Figure 5.8: Examples of violations of {TL, OthTL, VehLane, OutgoLane, OutgoCycLane, Jun, IncomLane, IncomCycLane, Pav, LftPav, RhtPav, XingLoc, BusStop, Parking}.

(a) I3D

(b) C2D

(c) RCGRU

(d) RCLSTM

(e) RCN

(f) SlowFast

Figure 5.9: Examples of violations of {Ped, Car, Cyc, Mobike, MedVeh, LarVeh, Bus, EmVeh, ¬MovAway}.

Chapter 6

Related Work

In this related work chapter, I conduct a comprehensive and fine-grained analysis of the works developed for the general problem of learning with logical constraints, as defined in Chapter 2 (Section 2.3).

Recall that the problem of learning with logical constraints can be formulated as a couple $(\mathcal{P}, \Pi)$, where $\mathcal{P}$ is a standard machine learning problem, and Π is a finite non-empty set of logical constraints. An example of problem of learning with logical constraints can be built by taking the classification problem associated with CIFAR-100 [66], associating with each class and superclass a separate Boolean output, and then formalizing the knowledge that sharks and trouts are both fish but that sharks are not trouts with the set of logical constraints:

$$\begin{aligned}\Pi = \{&\forall x.(\mathit{Shark}(x) \rightarrow \mathit{Fish}(x)), \quad \forall x.(\mathit{Trout}(x) \rightarrow \mathit{Fish}(x)), \\ &\forall x.(\mathit{Shark}(x) \rightarrow \neg \mathit{Trout}(x))\}.\end{aligned} \tag{6.1}$$

In the above formulas, *Shark*, *Fish*, and *Trout* are predicates, each corresponding to a Boolean output of the network. Notice that the above constraints can be equivalently written in many different ways, exploiting well-known first-order and propositional logic equivalences. Still, in the literature, constraints are often written and handled as *rules*, i.e., formulas like the ones in (6.1), having one of the following two forms

$$\forall x.(F(x) \rightarrow A(x)), \quad \forall x.(F(x) \rightarrow \neg A(x)), \tag{6.2}$$

where $F(x)$ is a conjunction of *literals* (i.e., atomic formulas and negations of atomic formulas, where a formula is *atomic* if it does not contain propositional connectives and/or quantifiers), and A is a predicate associated with one Boolean output. The motivation of introducing rules as a special case of formulas is based on the fact that many models (including the ones presented in this thesis) express background knowledge as rules of the form (6.2), and then, for each data point x, use the value

computed for $F(x)$ to determine the value to be associated with $A(x)$. Furthermore, in the literature, there has been a special focus on *basic rules* in which also F (like A) is a predicate associated with a Boolean output. All the rules in (6.1) are basic.

The analysed papers are thus organised into four macro-categories based on the richness of the language that they use to express the constraints. Furthermore, I take a problem-oriented approach, and for each considered paper, I report which shortcomings the authors aim to address with the inclusion of the logical background knowledge. A summary of the proposed categorization is given in Table 6.2.

The rest of this chapter is organized as follows. Sections 6.1 to 6.4 describe the models that belong to each macro-category, starting from the one with the simplest corresponding language, up to the one with the richest. Thus, I first survey papers in which constraints are basic rules (Section 6.1) and then the ones allowing for general rules (Section 6.2). All the other surveyed papers are then divided into two categories, depending on whether they allow only for universal quantification (Section 6.3) or also for existential quantification (Section 6.4) in the constraints. I end the chapter with some concluding remarks and pointers to other related surveys, in Section 6.5

6.1 Basic Rules

Basic rules are expressions of the form (6.2) where both F and A are predicates corresponding to Boolean outputs. They have been used in the context of multi-label classification (MC) problems with constraints, being the special case of learning with constraints in which all the outputs are Boolean labels. Thus, each output has a corresponding associated predicate, and the rules specify the existing relations between the output labels. All reviewed papers use rules of the first form in (6.2) to express a hierarchical relation between two output labels, and some papers use also rules of the second form in (6.2) to express a mutual exclusion between two output labels. Thus, I further divide the papers depending on their usage of basic rules of the second form in (6.2).

Hierarchical rules. In learning problems with hierarchical rules, knowledge is expressed with basic rules of the first form in (6.2) in which also F (like A) is a Boolean output. Examples of hierarchical rules are the first two rules in (6.1), which are satisfied whenever a data point x predicted to be a shark or a trout is also predicted to be a fish. Interestingly, an entire field has been developed to deal with such rules, and it is the field of hierarchical multi-label classification (HMC). HMC problems

naturally arise in many different domains, such as image classification or functional genomics. HMC models have normally two goals: (i) improve on the SOTA models, and (ii) guarantee the satisfaction of the hierarchical constraints. As expected, many different neural models were developed for HMC problems. To present them, I follow the classic categorization used for general HMC models, which divides them into two groups based on how they exploit the hierarchical knowledge (as already seen in Chapter 2) [109]:

1. *local approaches* exploit the constraints to decompose the problem into smaller classification ones, and then combine the solutions appropriately, while

2. *global approaches* consist of single models able to associate objects with their corresponding classes in the hierarchy as a whole.

Local approaches can be further divided into three subcategories based on the strategy that they deploy to decompose the main task. The most popular strategy is *local classifier per level*, which is given when a method trains a different classifier for each level of the hierarchy. For example, HMC-LMLP [13, 14] is a model consisting of a multi-layer perceptron (MLP) per hierarchical level, and the predictions in one level are used as inputs to the network responsible for the predictions in the next level. This model was later extended in [132], where HMCN is proposed. HMCN is considered a hybrid model, because it is trained with both a local and a global loss. Another model belonging to this category is DEEPre [73], later extended in mlDEEPre [141], which consists of a neural network for each level of the hierarchy and is applied to the challenging problem of enzyme function prediction. On the other hand, if a method trains a classifier for each node of the hierarchy, then it belongs to the category of the *local classifiers per node*. An example model for this category is HMC-MLPN [42], in which one MLP for each node is deployed. Finally, if a method trains a different classifier per parent node in the hierarchy, then the model belongs to the category of the *local classifiers per parent node*. For example, Wu et al. in [134], propose to divide the hierarchy in *splits*, i.e., one level sub-trees including one parent node and all its children nodes, together with the links between the parent node and children nodes, then train a classifier for each split. In order to train each classifier, the authors transform the multi-label classification problem into a standard classification problem by using the label powerset problem transformation method explained in Chapter 2 (Section 2.1.1). A more recent model is DeepGO, proposed in [67], which could be considered a hybrid model, as it creates a network for each of the three main sub-ontologies of the Gene Ontology, however, each label has a

dedicated layer. Global methods, on the other hand, do not have any subclassification, and global neural models are quite recent. The first proposed global model based on neural networks is AWX [81], which is just a feed-forward neural network predicting the leaves of the hierarchy, and then inferring the value for the parent nodes. A more complex model is given by the model proposed in this thesis, C-HMCNN [48], which builds a constraint layer ensuring the satisfaction of the constraints. Such a layer works synergically with a constraint loss to exploit the background knowledge of the hierarchy. Finally, MBM [89] takes an alternative approach by representing the labels by boxes rather than vectors, and thus it is able to capture taxonomic relations among labels.

If we shift the focus on how the satisfaction of the constraints is guaranteed, HMC models can be divided in:

1. approaches that require a post-processing step to enforce the constraints, and

2. approaches that satisfy the constraints by construction.

In the first group, we find models like HMC-LMLP [13, 14], HMCN [132], and MBM [89]. A common policy to enforce the satisfaction of $A_1 \rightarrow A$ in the post-processing step is to force the output for class A_1 to be smaller than or equal to the output for A (see, e.g., [13, 14, 132]). However, other solutions are possible. For example, Borges and Nievola (2012) associate to each data point an initial set of classes which is then extended to include all their ancestors in the hierarchy, while Feng et al. (2018) apply a post processing method based on Bayesian networks in order to guarantee that the results are coherent with the hierarchy constraints. For a detailed overview on the many different policies that can be used to impose the hierarchy constraints as a post-processing step, see the survey [88]. In the second group, we find models like DEEPre [73], mlDEEPre [141], DeepGO [67], AWX [81] and the HMC model presented in this thesis: C-HMCNN [48]. Here it is interesting to note the advantages of C-HMCNN with respect to the other models belonging to this group:

- With respect to DEEPre and mlDEEPre, C-HMCNN has the advantage that its size does not grow with the depth of the hierarchy. Indeed, both of these models create a new neural model for each level of the hierarchy, and the hierarchical constraints are then enforced by allowing the neural network associated to the level $l + 1$ to make a prediction only if the neural network at level l returns a positive prediction.

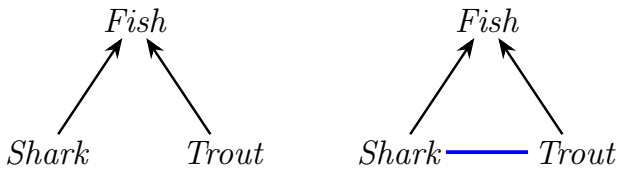


Figure 6.1: Example of a hierarchy DAG (left) and of a hierarchy and exclusion graph (right). Here, "$\rightarrow$" indicates a hierarchical relation, while "—" indicates mutual exclusion.

- With respect to DeepGO, C-HMCNN presents two advantages: (i) its size does not grow with the size of the hierarchy (indeed DeepGO creates a fully connected layer for each label in the hierarchy) and (ii) its gradients always point in the right direction (indeed, to ensure coherent classification, DeepGO also has a layer which selects the maximum value of the class and its children, however, the standard binary cross-entropy loss used).

- With respect to AWX, C-HMCNN has the advantage that it learns every label in the hierarchy, and it uses the hierarchy constraints to teach the underlying network to delegate the predictions when necessary. AWX, on the other hand, makes predictions only for the labels in the leaf nodes of the hierarchy, and uses the hierarchy knowledge to then infer the labels in higher levels of the hierarchy.

Hierarchical and exclusion rules. In computer vision, hierarchical rules are often used together with exclusion constraints expressed as basic rules of the second form in (6.2) in which F (like A) is a Boolean output. An example of a basic rule corresponding to an exclusion constraint is (6.1), which is satisfied whenever a data point x predicted to be a shark cannot be predicted to also be a trout. Hierarchical and mutual exclusion rules can be represented together in a hierarchy and exclusion (HEX) graph, firstly proposed in [29]. An example of a HEX graph is given in Figure 6.1. In the same paper, in addition to proposing HEX graphs, Deng et al. also build a neural model for large scale object classification which is guaranteed to satisfy the rules and is able to deal with incomplete labels. Another task to which HEX graphs are often applied is fine-grained image classification. Fine-grained image classification refers to the problem of classifying images that possess very subtle discriminatory features, e.g., classifying images of different birds by species, or of flowers by categories. Their usage in the field was firstly proposed in [135], where the authors tackle such a problem by adding a set of classes representing broader concepts than the initial ones (all the newly added classes will be thus parent nodes in the hierarchy), and then acquiring

a large number of images from external sources, which are labelled with such newly added classes. The new task is thus to predict both the initial label and the newly annotated labels. The authors show how, due to the newly added data, they achieve better results on this new task (notice that, on the contrary, both C-HMCNN(h) and CCN(h) were able to beat the state-of-the-art models on the original set of classes). An even more interesting study is shown in [17], where the authors show that it is possible to exploit the background knowledge to build a model that is able to get better results than the SOTA models on the initial set of classes. Chang et al.'s results have been recently surpassed by HRN [18], a neural network able to perform hierarchical feature interaction via residual connections.

6.2 General Rules

In this section, I lift the assumption that $F(x)$ in rules of the type (6.2) is a single literal, and I review the papers in which $F(x)$ is allowed to be a conjunction of m literals ($m \geq 1$). I classify such papers depending on whether $F(x)$ can (or cannot) contain functions and/or predicates corresponding to input features. In the first case, given a data point x, the rules constrain the output label A also on the basis of the input x, and thus each rule corresponds to an input-output constraint. In the second case, each rule excludes some output configurations independently from the data point x.

Input-output constraints. Interestingly, this type of constraints was the very first to be studied in this field. This is probably due to the fact that before the spread of machine learning systems, many classifiers were still hand-built, and thus researchers had a large availability of rules of the type "if the input has these features, then the data point belongs to this class". The goal of these models was thus to overcome the flaws of both hand-built classifiers and learned classifiers (which could be seen as almost complementary flaws) by creating hybrid systems. One of the first works able to incorporate such rules in the topology of the neural network was KBANN [108, 120]. Given a set of rules, Shavlik and Towell in [108] map the supporting facts to the input neurons, the intermediate conclusions to the hidden neurons, and the final conclusions to the output neurons. Given such a mapping, clearly KBANN needs the additional assumption that the constraints are acyclic. KBANN was later extended by Fu in [46], where KBCNN is proposed. Given a set of acyclic rules, not only KBCNN is able to map the rules into a neural network like KBANN, but also it is able to

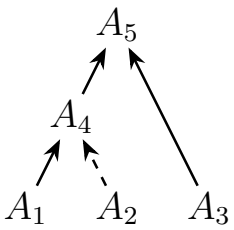
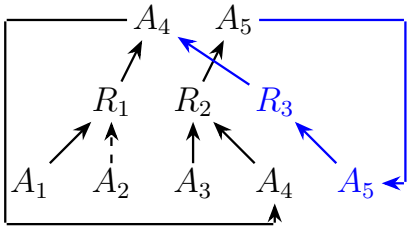
Example Rules	**High Level Algorithm**
$\forall x.(A_1(x) \wedge \neg A_2(x) \rightarrow A_4(x))$ $\forall x.(A_3(x) \wedge A_4(x) \rightarrow A_5(x))$ $\forall x.(A_5(x) \rightarrow A_4(x))$	1. Map rules to the network. 2. Add n and m neurons to the input and hidden layers, respectively. n, m are user-defined parameters. 3. Fully connect the newly added neurons to the adjacent layers. 4. Perturb all the network's weights.
KBANN Mapping	**CIL^{2}P Mapping**
	

Table 6.1: KBANN and CIL^{2}P comparison. Given the sample rules (first row, first column), I provide the high-level algorithm that the two models share to build the networks from the rules (first row, second column), and I show how the rule mapping (Step 1) is done in KBANN (second row, first column) and CIL^{2}P (second row, second column), respectively. I highlight $\forall x.(A_5(x) \rightarrow A_4(x))$, because while CIL^{2}P can map it in the network, KBANN must exclude it in order to maintain the acyclicity of the rules ($\forall x.(A_5(x) \rightarrow A_4(x))$ is indeed absent in KBANN).

map back the neural network to a set of learned rules. Such learned rules have the advantage of being completely transparent to the human user. While both KBANN and KBCNN build a neural network directly from the rules, in [44], the rules are treated as an expert system that is refined through incremental hidden unit generation, which allows the model to learn new rules without corrupting the initial ones (as often happens in KBANN), and to get a good performance even with severely incomplete rule bases (contrarily to KBANN). Another model proposed to solve such problems is Cascade ARTMAP [119], which is able to learn a set of rules that are more accurate and simpler than the ones extracted from KBANN. While all the methods explored so far can only deal with acyclic rules, one of the first methods able to deal with cyclic rules was CIL^{2}P [25]. CIL^{2}P and KBANN share the same high-level learning algorithm, however, they map the rules into neural networks in different ways. Due to such a different mapping, not only d'Avila Garcez and Zaverucha [25] can map cyclic rules, but they are also able to prove that CIL^{2}P computes the stable model of the logic program represented by the neural network, thus making it a parallel system for

logic programming. Notice that CIL^{2}P was later extended in [45] to represent and learn first-order logic formulas. This new system is called CILP++. Since KBANN and CIL^{2}P are probably the most well-known early models able to combine logic and neural networks, I provide a more detailed comparison between them in Table 6.1.

More details on the approaches from the 90s to combine knowledge engineering and machine learning are given in the surveys [107, 86].

Constraints over the output. While all these early models focus on input-output constraints, the more recent works focus on constraints over the output domain. From a high-level perspective, it is possible to further divide these models based on of how they integrate the neural network with the constraints. In the first set of models, learning and reasoning represent two different steps in a pipeline, which is though trained end-to-end. On the other hand, in the second set of models, the constraints are directly integrated in the network structure and/or loss function, thus being "single-stage models".

Pipelines: Probably one of the most famous methods able to constrain neural networks' outputs is DeepProbLog [79]. DeepProbLog extends ProbLog [95] by encapsulating the outputs of the neural network in the form of neural predicates. This can be easily done, because (in ProbLog) atomic expressions are already assigned a probability. Furthermore, since the algebraic extension of ProbLog [61] already supports automatic differentiation, it is possible to backpropagate the gradient from the loss at the output through the neural predicates into the neural networks, thus allowing the neural networks to learn directly from the logical constraints. DeepProbLog can thus be seen as a two-stage pipeline, where the neural networks handle the low-level perception, and then the reasoning is done at a logical level. Very close to DeepProbLog is NeurASP [138], however, differently from DeepProbLog, NeurASP employs reasoning originating from answer set programming, such as defaults, aggregates, and optimization rules. A deeper pipeline, in which each stage models a sub-task of a more complex problem, has been proposed in [101], where the authors present Nuts&Bolts: a framework in which each stage can be a set of different function approximators, and that can be trained end-to-end to minimize a global loss function. Nuts&Bolts is able to exploit the background knowledge by expressing it as rules, then translating them into probabilistic soft logic and incorporating it as one of the function approximators. Notice that, in this case, the rules do not only express constraints over the final outputs of the pipeline, but also over the outputs of the intermediate stages.

Single-stage models: Among these models, it is possible to find the model proposed in this thesis, CCN. Another model belonging to this category is the one proposed byLi and Srikumar in [72]. In this work, the authors also change the structure of neural networks to incorporate the background knowledge expressed by the constraints. To this end, the authors recognize that in a neural network some neurons (called *named neurons*) can be endowed with semantics tied to the task, and that logical rules can be written over such named neurons. Thanks to this particular formulation, Li and Srikumar are able to inject constraints over all the outputs of the neural network, even the intermediate ones (e.g., the outputs of the attention layer). The constraints that they are able to capture are thus over (i) the output domain, (ii) the intermediate outputs, and (iii) the relations between the intermediate outputs and the final outputs. Minervini and Riedel in [85], on the other hand, map the constraints to a loss function that is used to generate adversarial examples that cause a model to violate preexisting background knowledge. In particular, they apply their method to the task of recognizing textual entailment, and they add rules like "if sentence s_1 contradicts sentence s_2, then s_2 contradicts s_1 as well", ultimately showing that their models obtain a much better performance on adversarial datasets.

6.3 Universally Quantified Formulas

In this section, I describe methods that allow for constraints expressed as universally quantified first-order formulas. All the methods belonging to this category either inject the constraints in the loss (loss-based methods) or they constrain the value of the outputs (constrained-output methods).

Loss-based methods. While these methods obviously cannot guarantee the satisfaction of the constraints, they are able to generalize from less data and/or exploit unlabelled data. One of the first papers to propose to exploit background knowledge to solve the problem in data-scarce settings was [115]. In this paper, the authors show how it is possible to train neural networks to detect objects without any direct supervision, by simply injecting in the loss the constraints on the output domain. In their work, they considered both constraints pertaining to dynamics equations and logical constraints. The main drawback of the proposed approach is that the authors had to manually engineer a different loss function in each of the proposed examples. After this work, a number of papers on how to automatically map universally quantified constraints into a loss function were studied. For example, in [137],

semantic loss is proposed. Given a data point and a neural network that outputs the vector of probabilities p, the *semantic loss* is defined such that it is proportional to the negative logarithm of the probability of generating an interpretation that satisfies the constraints, when sampling values according to p. This method has the great quality of being completely syntax-independent (i.e., no matter how the constraints are written, the value of the loss does not change). Another model that incorporates the logical constraints in the loss is LENSR [136]. However, in this model, the formulas are first rewritten in either conjunctive normal form or decision deterministic decomposable negation normal form, and then projected onto a manifold where entailment is related to distance. Once learned, the logic embeddings can then be used to form a logic loss that guides the neural network training by encouraging formula embeddings to be close to satisfying assignments, and far from unsatisfying assignments. Yet another method that translates the constraints in continuous terms to be added to the loss is DL2 [43]. What is particularly interesting about this method is that it allows for queries. For example, a user can inquire the models on which neurons took part in a decision, or to generate adversarial examples violating a given constraint. Furthermore, notice that the language includes Boolean combinations of comparisons between terms, where a term is any real-valued function that may appear as a subexpression of a loss function.

Constrained-output methods. The methods presented here have the goal of guaranteeing the satisfaction of the constraints, and thus they apply the constraints directly on the outputs. MultiPlexNet [55] further extends the language, as its constraints can consist of any quantifier-free linear arithmetic formula over the rationals (thus, involving "$+$", "$\geq$", "$\neg$", "$\wedge$", and "$\vee$"). In MultiPlexNet, the formulas are expressed in disjunctive nomal form (DNF), and then the output layer of an existing neural network is augmented to include a separate transformation for each term in the DNF formula. Thus, the network's output layer can be viewed as a multiplexor in a logical circuit that permits for a branching of logic, and exactly from this property, the model gets the name MultiPlexNet. Another work that limits the output space of the neural network is NESTER [37]. However, in this case, the constraints are not mapped into the last layer of the network (like MultiPlexNet or CNN), but they are enforced by passing the outputs of the neural network to a constraint program, which enforces the constraints. NESTER can enforce hard and soft constraints over both categorical and numerical variables, and the entire architecture can be trained end-to-end by backpropagation.

6.4 Universally and Existentially Quantified Formulas

Our final category contains the models that allow for both universal and existential quantification. I divide the works between those that inject the constraints into the loss (loss-based methods) and those that create specialized neural structures to incorporate the constraints into the topology of the neural network (specialized structure-based methods).

Loss-based methods. One of the first models developed to this end was *semantic-based regularization* (SBR) [30, 32]. Like the above loss-based methods, SBR was developed with the goal of exploiting unlabelled data to train machine learning models. In this work, the authors map the constraints to a differentiable regularization term that can be added to any loss. To this end, they perform the following steps: (i) the first-order logic (FOL) expressions are grounded, (ii) the quantifier-free formulas are mapped to real-valued functions using the t-norms [64], and (iii) each FOL formula containing a universally (resp., existentially) quantified variable x is mapped to a function that returns the average (resp., maximum) of the t-norm generalization when grounding x over its domain. Other methods take inspiration from the distillation method proposed in [54], and have the goal of creating models able to both beat the SOTA models and exploit unlabelled data. In [56], the authors propose an *iterative rule knowledge distillation* procedure that transfers the structured information encoded in the logical rules into the network. In particular, at each iteration, a rule-regularized neural teacher is built, and then a student network is trained to imitate the predictions of the teacher network, while also trying to predict the right label for as many data points as possible. This work was later extended in [57]. This framework iteratively transfers information between the neural network and the structured knowledge, thus resulting in an effective integration of the representation learning capacity of neural networks and the generalization power of the structured knowledge. Different from distillation-based methods, but following a similar reasoning about how unifying two complementary paradigms can yield mutual benefits and surpass SOTA models, an abductive learning (ABL) approach is introduced in [23]. ABL combines a neural network, responsible for interpreting subsymbolic data into primitive logical facts, with a logical component able to reason on these facts. If the generated facts are inconsistent with the background knowledge, ABL generates new pseudo-labels that more likely satisfy the background knowledge and retrains the neural network accordingly.

Specialized structure-based methods. Instead of mapping the constraints into the loss function, Badreddine et al. in [3] map functions and predicates to matrices, and constants to vectors. In this work, the authors propose *real logic*: a logical formalism on a first-order language whereby formulas have a truth value in the interval [0,1] and a semantics defined concretely on the domain of real numbers. They then show how real logic can be encoded in the logic tensor networks (LTNs) proposed by Serafini and d'Avila Garcez in [105]. Donadello et al. in [36] later show how to use LTNs for semantic image interpretation, and Marra et al. in [80] present LYRICS, an extension of LTNs that provides an input language allowing for background knowledge in FOL, where predicate and function symbols are grounded onto any computational graph. All the logics listed above belong to the family of differentiable fuzzy logics (DFL) presented in [126, 127]. DFL is a family of differentiable logics, where the term *differentiable logics* refers to a logic along with a translation scheme from logical expressions to differentiable loss functions. So, differentiable fuzzy logics stand for the case where the logic is a fuzzy logic and the translation scheme applies to logical expressions that include fuzzy operators. In these works, the authors analyze DFL's behavior over different choices of fuzzy logic operators and give some recommendations on the choice of the operators.

6.5 Summary and Outlook

In this chapter, I conducted a comprehensive and fine-grained analysis of deep learning approaches in which background knowledge is expressed and then exploited as logical constraints in first-order logic. I categorized the papers based on the richness of the logical language used to express the constraints. Furthermore, for each approach, I reported which shortcomings the authors wanted to address via the background knowledge.

Coarser surveys on how to include background knowledge expressed in different ways ca be found in [130, 24]. Further, I would like to highlight that all the presented models fall into the broader field of neural-symbolic computing, whose aim is to integrate the abilities of learning and of reasoning about what has been learned, and a comprehensive survey on neural-symbolic models can be found at [26].

Model	Expressivity	Guaranteed Satisfaction	Less Data/ Supervision	Improve on SOTA
HMC-LMLP [13]	basic rules	✗	✗	✓
HEX [29]		✗	✓	✓
HAR-CNN [135]		✓	✗	✓
DEEPre [73]		✓	✗	✓
HMCN [132]		✗	✗	✓
AWX [81]		✓	✗	✓
DeepGO [67]		✓	✗	✓
mlDEEPre [141]		✓	✗	✓
C-HMCNN [48]		✓	✗	✓
MBM [89]		✗	✗	✓
KBANN [108]	general rules	✗	✗	✓
KBCNN [46]		✗	✗	✓
Iterative Neurons Addition [44]		✗	✗	✓
Cascade ARTMAP [119]		✗	✗	✓
CIL^{2}P [25]		✗	✗	✓
Adversarial Regularisation [85]		✗	✗	✓
DeepProbLog [79]		✗	✓	✗
NeurASP [138]		✗	✓	✗
CCN [49]		✓	✗	✓
Nuts&Bolts [101]		✗	✓	✓
Named Neurons [72]		✗	✓	✓
Label-free Supervision [115]	universally quantified formulas	✗	✓	✗
Semantic Loss [137]		✗	✓	✗
LENSR [136]		✗	✗	✓
DL2 [43]		✗	✓	✗
NESTER [37]		✓	✓	✗
MultiPlexNet [55]		✓	✓	✗
SBR [30]	universally and existentially quantified formulas	✗	✓	✗
CILP++ [45]		✗	✗	✓
LTN [105]		✗	✗	✓
Iterative Rule Distillation [56]		✗	✓	✓
Mutual Iterative Rule Distillation [57]		✗	✓	✓
LTN-SII [36]		✗	✗	✓
LYRICS [80]		✗	✓	✗
ABL [23]		✗	✓	✗
DFL [126]		✗	✓	✗

Table 6.2: Summary table of the analyzed works. For each work, I report which problem the authors wanted to tackle through the inclusion of the constraints. I identified three common problems tackled in the literature: (i) build models whose outputs are guaranteed to be compliant with a set of constraints, (ii) exploit the background knowledge to get a better performance in either low-data regimen settings or to exploit unlabelled data, and (iii) beat SOTA models on benchmark datasets (Columns 3 to 5). If a model exploits the background knowledge for one of the three tasks above, then is marked with "✓", and with "✗" otherwise.

Chapter 7

Conclusions

In this thesis, I investigated how to develop DL-based models able to guarantee that their behaviour is compliant with respect to a set of given requirements expressed as hard logical constraints. To this end, I focused on multi-label classification problems and I showed how to create a DL-based models able to (i) guarantee that their outputs satisfy a set of hard constraints over the output space by design, and (ii) learn from the background knowledge specified in the constraints to get better performance. In order to achieve such goal, in the first phase of my project, I focused on HMC problems and I developed a novel model called C-HMCNN(h), which, given a network h for the underlying MC problem, exploits the hierarchy information to produce predictions guaranteed to satisfy the hierarchical constraints and improve performance. C-HMCNN(h) presents four desirable features: (i) its predictions are coherent without any post-processing, (ii) differently from other state-of-the-art models (see, e.g., [132]), its number of parameters is independent from the number of hierarchical levels, (iii) it can be easily implemented on GPUs using standard libraries, and (iv) it outperforms the state-of-the-art models on 20 commonly used real-world HMC benchmarks.

Then, I considered MC problems with constraints expressed as normal logic rules [74], which generalize hierarchical constraints. For this class of problems, I proposed another novel model called CCN(h), which is the first model able to deal with MC problems with such expressive hard constraints. CCN(h) also presents fours desirable features: (i) its predictions are always coherent with the given constraints, (ii) it can be implemented on GPUs using standard libraries, (iii) it extends C-HMCNN(h), and thus outperforms the state-of-the-art HMC models on HMC problems, and (iv) it outperforms the state-of-the-art MC models on 16 LCMC problems obtained adding automatically extracted constraints to well-known MC problems.

Finally, I created the ROad event Awareness Dataset with logical Requirements (ROAD-R), the first publicly available dataset for autonomous driving with require-

ments expressed as propositional logic formulas. ROAD-R extends the ROAD dataset [111], by introducing a set of requirements that specifies the space of admissible outputs. ROAD consists of 22 relatively long ($\sim$8 minutes each) videos annotated with road events, while the annotated set consists of 243 requirements over the output space which were verified to hold for all bounding boxes. Thanks to ROAD-R, I was able to show that current state-of-the-art models are not able to learn the requirements just from the data points, as more than 90% of the times they produce predictions that violate the constraints, and that is possible to exploit the given requirements to create models that (i) have a better performance, and (ii) are guaranteed to be compliant with the requirements themselves.

Regarding the future, I envision that the specification and exploitation of logical constraints in DL-based models will become more and more widespread in the future, especially in safety-critical applications, where requirements are often logically specified, thus combining the advantages of manually engineering safety-critical features and automatically learning all other features. One open challenge for future work thus includes an AI system engineering approach that gives strict guarantees in the form of logical constraints on the system behavior based on such a combined system design. Another open challenge for future research concerns the exploitation of logical constraints in explainable AI. More precisely, logical constraints and reasoning about the functionality of neural networks and their output data can actually be exploited in order to model the abstract reasoning that is underlying the predictions of these neural networks, and it can thus be used in order to naturally create explanations for these predictions. However, only few papers, such as [78], have started to explore this large potential of logical constraints in DL. Notice that, if both the above challenges were to be addressed, then it would be possible to create models able to pass both the certification and the explanation process, and thus it would be possible to create trustworthy models in the sense of [59].

Appendix A

Experimental Analysis Details

A.1 HMC - Experimental Analysis Details

Dataset	Hidden Dimension	Learning Rate	Time per batch (GPU)	Time per batch (CPU)
Cellcycle FUN	500	10^{-4}	2.0	24.5
Derisi FUN	500	10^{-4}	2.0	13.6
Eisen FUN	500	10^{-4}	1.7	14.6
Expr FUN	1000	10^{-4}	1.9	12.6
Gasch1 FUN	1000	10^{-4}	2.0	8.5
Gasch2 FUN	500	10^{-4}	2.8	8.7
Seq FUN	2000	10^{-4}	2.0	16.5
Spo FUN	250	10^{-4}	1.6	13.8
Cellcycle GO	1000	10^{-4}	2.4	715.1
Derisi GO	500	10^{-4}	2.5	668.8
Eisen GO	500	10^{-4}	3.4	571.1
Expr GO	4000	10^{-5}	3.9	751.6
Gasch1 GO	500	10^{-4}	2.5	788.3
Gasch2 GO	500	10^{-4}	2.8	752.2
Seq GO	9000	10^{-5}	2.6	837.8
Spo GO	500	10^{-4}	3.3	720.3
Diatoms	2000	10^{-5}	2.0	71.3
Enron	1000	10^{-5}	3.6	1.9
Imclef07a	1000	10^{-5}	3.4	9.9
Imclef07d	1000	10^{-5}	2.9	1.5

Table A.1: Hidden dimension used for each dataset, learning rate used for each dataset, and average inference time per batch in milliseconds (ms). Average computed over 500 batches for each dataset.

In this section, I provide more details about the conducted experimental analysis for HMC problems. As stated in Chapter 3, across the different experiments, I

kept all hyperparameters fixed with the exception of the hidden dimension and the learning rate, which are reported in the first two columns of Table A.1. The other hyperparameters were determined by searching the best hyperparameters configuration on the Funcat datasets; I then took the configuration that led to the best results on the highest number of datasets. The hyperparameter values taken in consideration were: (i) learning rate: $[10^{-3}, 10^{-4}, 10^{-5}]$, (ii) batch size: $[4, 64, 256]$, (iii) dropout: $[0.6, 0.7]$, and (iv) weight decay: $[10^{-3}, 10^{-5}]$. Concerning the hidden dimension, I took into account all possible dimensions from 250 to 2000 with step equal to 250, and from 2000 to 10000 with step 1000. All experiments were run on an Nvidia Titan Xp with 12 GB memory.

In the last two columns of Table A.1, I compare CCN(h)'s average inference time per batch in milliseconds when run on a CPU and on a GPU. The average is computed over 500 batches for each dataset. As it can be seen from the table, implementing CCN(h) on GPU led to much smaller inference times, especially on the GO datasets. For this experiment, I used an Nvidia Titan Xp with 12 GB memory as GPU and an Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz as CPU. The hyperparameters of all neural networks were kept the same as in the other experiments.

A.2 LCMC - Experimental Analysis Details

In this section, I provide more details about the conducted experimental analysis for LCMC problems. As stated in Chapter 4, across the different experiments, I kept all hyperparameters fixed with the exception of the hidden dimensions, which are reported in Table A.2. The other hyperparameters were determined by searching the best hyperparameters configuration on the validation sets; I then took the configuration that led to the best results on the highest number of datasets. The hyperparameter values taken in consideration were: (i) learning rate: $[10^{-3}, 10^{-4}, 10^{-5}]$, (ii) batch size: $[4, 64, 256]$, (iii) dropout: $[0.6, 0.7, 0.8]$, and (iv) weight decay: $[10^{-3}, 10^{-4}, 10^{-5}]$. Concerning the hidden dimension, I took into account all possible dimensions from 100 to 1000 with step equal to 100, and from 1000 to 5000 with step 500. Again, all experiments were run on an Nvidia Titan Xp with 12 GB memory.

Dataset	Hidden Dimension
Arts	4000
Business	4000
Cal500	100
Emotions	100
Enron	2500
Genbase	5000
Image	1000
Medical	800
Rcv1Subset1	600
Rcv1Subset2	1500
Rcv1Subset3	4000
Rcv1Subset4	1000
Rcv1Subset5	4000
Science	2000
Scene	1500
Yeast	4000

Table A.2: Hidden dimension used for each dataset.

Appendix B

ROAD-R Requirements

B.1 Requirements List

In Tables B.1, B.2, and B.3, I report the list of all the 243 requirements together with their natural language explanations.

Requirements	Natural Language Explanations
{Ped, not PushObj}	If an agent pushes an object then it is a pedestrian
{PushObj, not Ped, MovAway, MovTow, Mov, Stop, TurLft, TurRht, Wait2X, XingFmLft, XingFmRht, Xing}	A pedestrian can only push objects, move away, etc.
{Ped, not XingFmLft, Car, Cyc, Mobike, MedVeh, LarVeh, Bus, EmVeh}	Only pedestrains, cars, cyclists, etc. can cross from left
{Ped, not Wait2X, Cyc}	Only pedestrians and cyclists can wait to cross
{Ped, not Xing, Cyc}	Only pedestrians and cyclists can cross
{Ped, not Stop, Car, Cyc, Mobike, MedVeh, LarVeh, Bus, EmVeh}	Only pedestrians, cars, cyclists, etc can stop
{Ped, not Mov, Car, Cyc, Mobike, MedVeh, LarVeh, Bus, EmVeh}	Only pedestrians, cars, cyclists, etc can move
{Ped, not MovTow, Car, Cyc, Mobike, MedVeh, LarVeh, Bus, EmVeh}	Only pedestrians, cars, cyclists, etc can move towards
{Ped, not MovAway, Car, Cyc, Mobike, MedVeh, LarVeh, Bus, EmVeh}	Only pedestrians, cars, cyclists, etc can move away
{Ovtak, not EmVeh, MovAway, MovTow, Mov, Brake, Stop, IncatLeft, IncatRht, HazLit, TurLft, TurRht, XingFmRht, XingFmLft, Xing}	An emergency vehicle can only overtake, move away etc.
{EmVeh, not HazLit, Car, MedVeh, LarVeh, Bus, Mobike}	Only emergency vehicles, cars etc. can have hazards lights on
{Ovtak, not Bus, MovAway, MovTow, Mov, Brake, Stop, IncatLeft, IncatRht, HazLit, TurLft, TurRht, XingFmRht, XingFmLft, Xing}	A bus can only overtake, move away move towards etc.
{Ovtak, not MedVeh, MovAway, MovTow, Mov, Brake, Stop, IncatLeft, IncatRht, HazLit, TurLft, TurRht, XingFmRht, XingFmLft, Xing}	A medium vehicle can only overtake, move away, move towards etc.
{Ovtak, not LarVeh, MovAway, MovTow, Mov, Brake, Stop, IncatLeft, IncatRht, HazLit, TurLft, TurRht, XingFmRht, XingFmLft, Xing}	A large vehicle can only overtake, move away, move towards etc.
{OthTL, not Green, TL}	Only traffic lights and other traffic lights can give a green signal
{OthTL, not Amber, TL}	Only traffic lights and other traffic lights can give an amber signal
{OthTL, not Red, TL}	Only traffic lights and other traffic lights can give a red signal
{Ovtak, not Mobike, MovAway, MovTow, Mov, Brake, Stop, IncatLeft, IncatRht, HazLit, TurLft, TurRht, XingFmRht, XingFmLft, Xing}	A motorbike can only overtake, move away, move towards etc.
{Xing, not Cyc, MovAway, MovTow, Mov, Brake, Stop, IncatLeft, IncatRht, TurLft, TurRht, Ovtak, Wait2X, XingFmLft, XingFmRht}	A cyclist can only cross, move away, move towards etc.
{Cyc, not Ovtak, MedVeh, LarVeh, Bus, Mobike, EmVeh, Car}	Only cyclists, medium vehicles, large vehicles etc. can overtake
{Cyc, not IncatRht, MedVeh, LarVeh, Bus, Mobike, EmVeh, Car}	Only cyclists, medium vehicles, large vehicles etc. can indicate right
{Cyc, not IncatLeft, MedVeh, LarVeh, Bus, Mobike, EmVeh, Car}	Only cyclists, medium vehicles, large vehicles etc. can indicate left
{Cyc, not Brake, MedVeh, LarVeh, Bus, Mobike, EmVeh, Car}	Only cyclists, medium vehicles, large vehicles etc. can brake
{Ovtak, not Car, MovAway, MovTow, Mov, Brake, Stop, IncatLeft, IncatRht, HazLit, TurLft, TurRht, XingFmRht, XingFmLft, Xing}	A car can only overtake, move away, move towards etc.
{Car, not TurRht, Cyc, Mobike, MedVeh, LarVeh, Bus, EmVeh}	Only cyclists, medium vehicles, large vehicles etc. can turn right
{Car, not TurLft, Cyc, Mobike, MedVeh, LarVeh, Bus, EmVeh}	Only cyclists, medium vehicles, large vehicles etc. can turn left
{VehLane, OutgoLane, OutgoCycLane, IncomLane, IncomCycLane, Pav, LftPav, RhtPav, Jun, XingLoc, BusStop, Parking, TL, OthTL}	Every agent but traffic lights must have a position
{Ped, Car, Cyc, Mobike, MedVeh, LarVeh, Bus, EmVeh, TL, OthTL}	There must be at least an agent
{not Car, not Cyc}	A car cannot be a cyclist
{not Car, not Mobike}	A car cannot be a motorbike
{not Car, not MedVeh}	A car cannot be a medium vehicle
{not Car, not LarVeh}	A car cannot be a large vehicle
{not Car, not Bus}	A car cannot be a bus
{not Cyc, not Mobike}	A cyclist cannot be a motorbike
{not Car, not EmVeh}	A car cannot be a emergency vehicle
{not Car, not TL}	A car cannot be a traffic light
{not Cyc, not MedVeh}	A cyclist cannot be a medium vehicle
{not Car, not OthTL}	A car cannot be a other traffic light
{not Car, not Red}	A car cannot signal red
{not Cyc, not LarVeh}	A cyclist cannot be a large vehicle
{not Car, not Amber}	A car cannot signal amber
{not Car, not Green}	A car cannot signal green
{not Cyc, not Bus}	A cyclist cannot be a bus
{not Mobike, not MedVeh }	A motorbike cannot be a medium vehicle
{not Cyc, not EmVeh}	A cyclist cannot be an emergency vehicle
{not Mobike, not LarVeh}	A motorbike cannot be a large vehicle
{not Cyc, not TL}	A cyclist cannot be a traffic light
{not Cyc, not OthTL}	A cyclist cannot be a other traffic light
{not Mobike, not Bus}	A motorbike cannot be a bus
{not Cyc, not Red}	A cyclist cannot signal red
{not MedVeh, not LarVeh}	A medium vehicle cannot be a large vehicle
{not Mobike, not EmVeh}	A motorbike cannot be an emergency vehicle
{not Cyc, not Amber}	A cyclist cannot signal amber
{not Car, not Wait2X}	A car cannot wait to cross
{not TL, not TurLft}	A traffic light cannot turn left
{not Green, not MovTow}	A green traffic light cannot move towards
{not Red, not Stop}	A red traffic light cannot stop
{not OthTL, not IncatRht}	A other traffic light cannot indicate right
{not TurRht, not TL}	A traffic light cannot turn right
{not Amber, not Brake}	A amber traffic light cannot brake
{not OthTL, not HazLit}	A other traffic light cannot have the hazards light on
{not Red, not IncatLeft}	A red traffic light cannot indicate left
{not Green, not Mov}	A green traffic light cannot move

Table B.1: Requirements table.

Requirements	Natural Language Explanations
{not MedVeh, not Bus}	A medium vehicle cannot be a bus
{not Car, not Xing}	A car cannot be crossing
{not Mobike, not OthTL}	A motorbike cannot be an other traffic light
{not Car, not PushObj}	A car cannot push objects
{not MedVeh, not EmVeh}	A medium vehicle cannot be an emergency one
{not Mobike, not Red}	A motorbike cannot be a red traffic light
{not LarVeh, not Bus}	A large vehicle cannot be a bus
{not Brake, not Cyc}	A cyclist cannot brake
{not MedVeh, not TL}	A traffic light cannot be a medium vehicle
{not Mobike, not Amber}	A motorbike cannot be an amber traffic light
{not LarVeh, not EmVeh}	A large vehicle cannot be an emergency vehicle
{not Mobike, not Green}	A motorbike cannot be a green traffic light
{not MedVeh, not OthTL}	A medium vehicle cannot be an other traffic light
{not HazLit, not Cyc}	A cyclist cannot have the hazard lights on
{not MedVeh, not Red}	A medium vehicle cannot be a red traffic light
{not LarVeh, not TL}	A large vehicle cannot be a traffic light
{not Car, not Ped}	A car cannot be a pedestrian
{not Mobike, not TL}	A motorbike cannot be a traffic light
{not Bus, not EmVeh}	A bus cannot be an emergency vehicle
{not MedVeh, not Amber}	A medium vechile cannot be an amber traffic light
{not LarVeh, not OthTL}	A large vehicle cannot be a other traffic light
{not MedVeh, not Green}	A medium vehicle cannot be a green traffic light
{not Bus, not TL}	A bus cannot be a traffic light
{not LarVeh, not Red}	A large vehicle cannot be a red traffic light
{not Bus, not OthTL}	A bus cannot be a other traffic light
{not LarVeh, not Amber}	A large vehicle cannot be a amber traffic light
{not Cyc, not PushObj}	A cyclist cannot push an object
{not EmVeh, not TL}	An emergency vehicle cannot be a traffic light
{not LarVeh, not Green}	A large vehicle cannot be a green traffic light
{not Bus, not Red}	A bus cannot be a red traffic light
{not EmVeh, not OthTL}	An emergency vehicle cannot be a other traffic light
{not Bus, not Amber}	A bus cannot be an amber traffic light
{not EmVeh, not Red}	A emergency vehicle cannot be a red traffic light
{not Mobike, not Wait2X}	A motorbike cannot wait to cross
{not Bus, not Green}	A bus cannot be a green traffic light
{not TL, not OthTL}	A traffic light cannot be a other traffic light
{not EmVeh, not Amber}	A emergency vehicle cannot be a amber traffic light
{not Mobike, not Xing}	A motorbike cannot be crossing
{not Cyc, not Ped}	A cyclist cannot be a pedestrian
{Ped, Cyc, not Xing}	If an agent is crossing it is either a pedestrian or a cyclist
{not Mobike, not PushObj}	A motorbike cannot push objects
{not EmVeh, not Green}	An emergency vehicle cannot be a green traffic light
{not MedVeh, not Wait2X}	A medium vehicle cannot wait to cross
{not TL, not MovAway}	A traffic light cannot move away
{not MedVeh, not Xing}	A medium vehicle cannot be crossing
{not Red, not Amber}	A red traffic light cannot be amber
{not MedVeh, not PushObj}	A medium vehicle cannot push objects
{not MovTow, not TL}	A traffic light cannot move towards
{not OthTL, not MovAway}	A other traffic light cannot move away
{not LarVeh, not Wait2X}	A large vehicle cannot wait to cross
{not TL, not Mov}	A traffic light cannot move
{not Red, not Green}	A red traffic light cannot be green

Requirements	Natural Language Explanations
{not TL, not XingFmRht}	A traffic light cannot be crossing from right
{not Amber, not IncatRht}	An agent cannot indicate right and signal amber
{not Red, not TurLft}	An agent cannot turn left and signal red
{not MovTow, not Mov}	An agent cannot move towards and move
{not TL, not Xing}	A traffic light cannot cross
{not OthTL, not Wait2X}	A other traffic light cannot wait to cross
{not Green, not IncatLeft}	An agent cannot indicate left and signal green
{not Red, not TurRht}	An agent cannot turn right and signal red
{not Amber, not HazLit}	An agent cannot have the hazard lights on and signal amber
{not TL, not PushObj}	A traffic light cannot push objects
{not OthTL, not XingFmLft}	A other traffic light cannot cross from left
{not Green, not IncatRht}	An agent cannot signal green and indicate right
{not Red, not Ovtak}	An agent cannot signal red and overtake
{not Amber, not TurLft}	An agent cannot signal amber and turn left
{not OthTL, not XingFmRht}	A other traffic light cannot cross from right
{not Red, not Wait2X}	An agent cannot signal red and wait to cross
{not Green, not HazLit}	An agent cannot signal green and have the hazard lights on
{not Amber, not TurRht}	An agent cannot signal amber and turn right
{not OthTL, not Xing}	A other traffic light cannot cross
{not Bus, not Ped}	A bus cannot be a pedestrian
{not Red, not XingFmLft}	An agent cannot signal red and cross from left
{not OthTL, not PushObj}	A other traffic light cannot push an object
{not Green, not TurLft}	An agent cannot signal green and turn left
{not Amber, not Ovtak}	An agent cannot signal amber and overtake
{not Mov, not Stop}	An agent cannot move and stop
{not Red, not XingFmRht}	An agent cannot signal red and cross from right
{not Amber, not Wait2X}	An agent cannot signal amber and wait to cross
{not Green, not TurRht}	An agent cannot signal green and turn right
{not Red, not Xing}	An agent cannot signal and cross
{not Amber, not XingFmLft}	An agent cannot signal amber and cross from left
{not Green, not Ovtak}	An agent cannot signal green and overtake
{not Amber, not XingFmRht}	An agent cannot signal amber and cross from right
{not EmVeh, not Ped}	An emergency vehicle cannot be a pedestrian
{not Green, not Wait2X}	An agent cannot signal green and wait to cross
{not Amber, not Xing}	An agent cannot signal amber and cross
{not Green, not XingFmLft}	An agent cannot signal green and cross from left
{not Green, not XingFmRht}	An agent cannot signal green and cross from right
{not Green, not Xing}	An agent cannot signal green and cross
{not TL, not Ped}	A traffic light cannot be a pedestrian
{not IncatLeft, not IncatRht}	An agent cannot indicate left and right
{not OthTL, not Ped}	A other traffic light cannot be a pedestrian
{not Brake, not Wait2X}	An agent cannot brake and wait to cross
{not Red, not Ped}	A pedestrian cannot signal red
{not Xing, not Brake}	An agent cannot cross and brake
{not Wait2X, not IncatLeft}	An agent cannot wait to cross and indicate left
{not Brake, not PushObj}	An agent cannot brake and push an object
{not Amber, not Ped}	A pedestrian cannot signal amber
{not Wait2X, not IncatRht}	An agent cannot wait to cross and indicate right
{not Green, not Ped}	A pedestrian cannot signal green
{not Wait2X, not Ovtak}	An agent cannot wait to cross and overtake
{not Wait2X, not XingFmLft}	An agent cannot wait to cross and cross from left
{not Wait2X, not XingFmRht}	An agent cannot wait to cross and cross from right

Table B.2: Requirements tables.

Requirements	Natural Language Explanations
{not Wait2X, not Xing}	An agent cannot wait to cross and cross
{not Brake, not Ped}	A pedestrian cannot brake
{not Ped, not IncatLeft}	A pedestrian cannot indicate left
{not Ped, not IncatRht}	A pedestrian cannot indicate right
{not HazLit, not Ped}	A pedestrian cannot have the hazard lights on
{not Cyc, not Green}	A cyclist cannot signal green
{not OutgoLane, not OutgoCycLane}	An outgoing lane cannot be an outgoing cycle lane
{not Ped, not Ovtak}	A pedestrian cannot overtake
{not VehLane, not IncomCycLane}	The vehicle lane cannot be an incoming cycle lane
{not OutgoLane, not IncomCycLane}	An incoming cycle lane cannot be a outgoing lane
{not IncomLane, not OutgoCycLane}	An outgoing cycle lane cannot be an incoming lane
{not OutgoLane, not Pav}	An outgoing lane cannot be a pavement
{not IncomCycLane, not OutgoCycLane}	A cycle lane cannot be incoming and outgoing
{not OutgoLane, not RhtPav}	An outgoing lane cannot be a right pavement
{not IncomCycLane, not Pav}	An incoming cycle lane cannot be a pavement
{not XingLoc, not OutgoLane}	A crossing cannot be an outgoing lane
{not VehLane, not Parking}	A vehicle lane cannot be a parking
{not BusStop, not OutgoLane}	A bus stop cannot be a outgoing lane
{not OutgoLane, not Parking}	An outgoing lane cannot be a parking
{not BusStop, not OutgoCycLane}	A bus stop cannot be a outgoing cycle lane
{not Parking, not OutgoCycLane}	A parking cannot be a outgoing cycle lane
{not XingLoc, not IncomCycLane}	A crossing cannot be an incoming cycle lane
{not LftPav, not RhtPav}	A pavement is either on the left or on the right
{not IncomLane, not Parking}	An incoming lane cannot be a parking
{not IncomCycLane, not BusStop}	An incoming cycle lane cannot be a bus stop
{not IncomCycLane, not Parking}	An incoming cycle lane cannot be a parking
{not Pav, not Parking}	A pavement cannot be a parking
{not LftPav, not Parking}	A left pavement cannot be a parking
{not RhtPav, not Parking}	A right pavements cannot be a parking
{not Jun, not Parking}	A junction cannot be a parking
{not XingLoc, not BusStop}	A crossing cannot be a bus stop
{not XingLoc, not Parking}	A crossing cannot be a parking
{not BusStop, not Parking}	A bus stop cannot be a parking
{not Mobike, not Ped }	A motorbike cannot be a pedestrian
{not MovTow, not OthTL}	A other traffic light cannot move towards
{not TL, not Brake}	A traffic light cannot brake
{not Red, not MovAway}	A red traffic light cannot move away
{not Amber, not Green}	An amber traffic light cannot be green
{not LarVeh, not Xing}	A large vehicle cannot cross
{not OthTL, not Mov}	A other traffic light cannot move
{not Stop, not TL}	A traffic light cannot stop
{not LarVeh, not PushObj}	A large vehicle cannot push objects
{not Red, not MovTow}	A red traffic light cannot move towards
{not Amber, not MovAway}	A amber traffic light cannot move away
{not Bus, not Wait2X}	A bus cannot wait to cross
{not TL, not IncatLeft}	A traffic light cannot idicate left
{not OthTL, not Brake}	A other traffic light cannot brake
{not Red, not Mov}	A red traffic light cannot move
{not TL, not IncatRht}	A traffic light cannot indicate right
{not Stop, not OthTL}	A other traffic light cannot stop
{not Amber, not MovTow}	A amber traffic light cannot move towards
{not Green, not MovAway}	A green traffic light cannot move away
{not TL, not HazLit}	A traffic light cannot have the hazard lights on
{not Red, not Brake}	A red traffic light cannot brake
{not Bus, not Xing}	A bus cannot be crossing
{not OthTL, not IncatLeft}	A other traffic light cannot indicate right
{not MedVeh, not Ped}	A medium vehicle cannot be a pedestrian
{not Amber, not Mov}	A amber traffic light cannot move
{not Bus, not PushObj}	A bus cannot push objects
{not EmVeh, not Wait2X}	A emergency vehicle cannot wait to cross

Requirements	Natural Language Explanations
{not TL, not Ovtak}	A traffic light cannot overtake
{not Amber, not Stop}	A amber traffic light cannot stop
{not EmVeh, not Xing}	An emergency vehicle cannot be crossing
{not OthTL, not TurLft}	A other traffic light cannot turn left
{not Red, not IncatRht}	A red traffic light cannot indicate right
{not TL, not Wait2X}	A traffic light cannot wait to cross
{not Green, not Brake}	A green traffic light cannot be green and break
{not MovAway, not Mov}	An agent cannot move perpendicularly to and away from the AV
{not EmVeh, not PushObj}	An emergency vehicle cannot push an object
{not TurRht, not OthTL}	A other traffic light cannot turn right
{not Amber, not IncatLeft}	An amber traffic light cannot indicate left
{not TL, not XingFmLft}	An traffic light cannot be crossing from the left
{not Red, not HazLit}	A red traffic light cannot have the hazard lights on
{not Green, not Stop}	A green traffic light cannot stop
{not LarVeh, not Ped}	A large vehicle cannot be a pedestrian
{not OthTL, not Ovtak}	A other traffic light cannot overtake

Table B.3: Requirements table.

Bibliography

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of VLDB*, 1994.

[2] Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Towards a theory of declarative knowledge. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann, 1988.

[3] Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor networks. *Artificial Intelligence*, 303, 2022.

[4] Olivier Bailleux and Pierre Marquis. Some computational aspects of distance-sat. *Journal of Automated Reasoning*, 37(4), 2006.

[5] Zafer Barutcuoglu, Robert E. Schapire, and Olga G. Troyanskaya. Hierarchical multi-label prediction of gene function. *Bioinformatics*, 22(7), 2006.

[6] Alessio Benavoli, Giorgio Corani, and Francesca Mangili. Should we really use post-hoc tests based on mean-ranks? *Journal of Machine Learning Research*, 17 (5), 2016.

[7] Wei Bi and James T. Kwok. Multi-label classification on tree- and DAG-structured hierarchies. In *Proceedings of ICML*, 2011.

[8] Helyane Bronoski Borges and Júlio C. Nievola. Multi-label hierarchical classification using a competitive neural network for protein function prediction. In *Proceedings of IJCNN*, 2012.

[9] Matthew R. Boutell, Jiebo Luo, Xipeng Shen, and Christopher M. Brown. Learning multi-label scene classification. *Pattern Recognition*, 37(9), 2004.

[10] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2), 1996.

[11] Leo Breiman. Random forests. *Machine Learning*, 45(1), 2001.

[12] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? A new model and the kinetics dataset. In *Proceedings of CVPR*, 2017.

[13] Ricardo Cerri, Rodrigo C. Barros, and André Carlos Ponce de Leon Ferreira de Carvalho. Hierarchical multi-label classification for protein function prediction: A local approach based on neural networks. In *Proceedings of ISDA*, 2011.

[14] Ricardo Cerri, Rodrigo C. Barros, and André Carlos Ponce de Leon Ferreira de Carvalho. Hierarchical multi-label classification using local neural networks. *Journal of Computer and System Sciences*, 80(1), 2014.

[15] Ricardo Cerri, Rodrigo C. Barros, André Carlos Ponce de Leon Ferreira de Carvalho, and Yaochu Jin. Reduction strategies for hierarchical multi-label classification in protein function prediction. *BMC Bioinformatics*, 17, 2016.

[16] Nicolò Cesa-Bianchi, Claudio Gentile, and Luca Zaniboni. Incremental algorithms for hierarchical classification. *Journal of Machine Learning Research*, 7, 2006.

[17] Dongliang Chang, Kaiyue Pang, Yixiao Zheng, Zhanyu Ma, Yi-Zhe Song, and Jun Guo. Your "flamingo" is my "bird": Fine-grained, or not. In *Proceedings of CVPR*, 2021.

[18] Jingzhou Chen, Peng Wang, Jian Liu, and Yuntao Qian. Label relation graphs enhanced hierarchical residual network for hierarchical multi-granularity classification. 2022. *arXiv:2201.03194*.

[19] Amanda Clare. *Machine Learning and Data Mining for Yeast Functional Genomics*. PhD thesis, University of Wales, 2003.

[20] Amanda Clare and Ross D. King. Knowledge discovery in multi-label phenotype data. In *Proc. of PKDD*, 2001.

[21] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.

[22] Nicolas Coudray, Paolo Santiago Ocampo, Theodore Sakellaropoulos, Navneet Narula, Matija Snuderl, David Fenyö, Andre L. Moreira, Narges Razavian, and Aristotelis Tsirigos. Classification and mutation prediction from non–small cell lung cancer histopathology images using deep learning. *Nature Medicine*, 24 (10), 2018. URL `https://doi.org/10.1038/s41591-018-0177-5`.

[23] Wang-Zhou Dai, Qiuling Xu, Yang Yu, and Zhi-Hua Zhou. Bridging machine learning and logical reasoning by abductive learning. In *Proceedings of NeurIPS*, 2019.

[24] Tirtharaj Dash, Sharad Chitlangia, Aditya Ahuja, and Ashwin Srinivasan. A review of some techniques for inclusion of domain-knowledge into deep neural networks. *Scientific Reports*, 12, 2022.

[25] Artur d'Avila Garcez and Gerson Zaverucha. The connectionist inductive learning and logic programming system. *Applied Intelligence*, 11(1), 1999.

[26] Artur d'Avila Garcez, Marco Gori, Luís Lamb, Luciano Serafini, Michael Spranger, and Son Tran. Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. *Journal of Applied Logics*, 6(4), 2019.

[27] Janez Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7, 2006.

[28] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. ImageNet: A large-scale hierarchical image database. In *Proceedings of CVPR*, 2009.

[29] Jia Deng, Nan Ding, Yangqing Jia, Andrea Frome, Kevin Murphy, Samy Bengio, Yuan Li, Hartmut Neven, and Hartwig Adam. Large-scale object classification using label relation graphs. In *Proceedings of ECCV*, 2014.

[30] Michelangelo Diligenti, Marco Gori, Marco Maggini, and Leonardo Rigutini. Bridging logic and kernel machines. *Machine Learning*, 86, 2012.

[31] Michelangelo Diligenti, Marco Gori, and Claudio Sacca. Semantic-based regularization for learning and inference. *Artificial Intelligence*, 244, 2017.

[32] Michelangelo Diligenti, Soumali Roychowdhury, and Marco Gori. Integrating prior knowledge into deep learning. In *Proceedings of ICMLA*, 2017.

[33] Ivica Dimitrovski, Dragi Kocev, Suzana Loskovska, and Sašo Džeroski. Hierchical annotation of medical images. In *Proceedings of IS*, 2008.

[34] Ivica Dimitrovski, Dragi Kocev, Suzana Loskovska, and Saso Dzeroski. Hierarchical classification of diatom images using ensembles of predictive clustering trees. *Ecological Informatics*, 7(1), 2012.

[35] Sotiris Diplaris, Grigorios Tsoumakas, Pericles A. Mitkas, and Ioannis Vlahavas. Protein classification with multiple algorithms. In Panayiotis Bozanis and Elias N. Houstis, editors, *Advances in Informatics*, 2005.

[36] Ivan Donadello, Luciano Serafini, and Artur D'Avila Garcez. Logic tensor networks for semantic image interpretation. In *Proceedings of IJCAI*, 2017.

[37] Paolo Dragone, Stefano Teso, and Andrea Passerini. Neuro-symbolic constraint programming for structured prediction. In *Proceedings of IJCLR-NeSy*, 2021.

[38] Harris Drucker. Improving regressors using boosting techniques. In *Proc. of (ICML.*

[39] André Elisseeff and Jason Weston. A kernel method for multi-labelled classification. In *Proceedings of NeurIPS*, 2001.

[40] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. SlowFast networks for video recognition. In *Proceedings of ICCV*, 2019.

[41] Lei Feng, Bo An, and Shuo He. Collaboration based multi-label learning. In *Proceedings of AAAI*, 2019.

[42] Shou Feng, Ping Fu, and Wenbin Zheng. A hierarchical multi-label classification method based on neural networks for gene function prediction. *Biotechnology and Biotechnological Equipment*, 32, 2018.

[43] Marc Fischer, Mislav Balunovic, Dana Drachsler-Cohen, Timon Gehr, Ce Zhang, and Martin Vechev. DL2: training and querying neural networks with logic. In *Proceedings of ICML*, 2019.

[44] Justin Fletcher and Zoran Obradovic. Combining prior symbolic knowledge and constructive neural network learning. *Connection Science*, 5, 1993.

[45] Manoel França, Gerson Zaverucha, and Artur d'Avila Garcez. Fast relational learning using bottom clause propositionalization with artificial neural networks. *Machine Learning*, 94, 2014.

[46] Li Min Fu. Knowledge-based connectionism for revising domain theories. *IEEE Transactions on Systems, Man, and Cybernetics*, 23, 1993.

[47] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of Logic Programming*, 1988.

[48] Eleonora Giunchiglia and Thomas Lukasiewicz. Coherent hierarchical multi-label classification networks. In *Proceedings of NeurIPS*, 2020.

[49] Eleonora Giunchiglia and Thomas Lukasiewicz. Multi-label classification neural networks with hard logical constraints. *Journal of Artificial Intelligence Research*, 72, 2021.

[50] Eleonora Giunchiglia, Mihaela Catalina Stoian, Salman Khan, Fabio Cuzzolin, and Thomas Lukasiewicz. ROAD-R: The autonomous driving dataset for learning with requirements. *Currently Under Submission*, 2022.

[51] Eleonora Giunchiglia, Mihaela Catalina Stoian, and Thomas Lukasiewicz. Deep learning with logical constraints. In *Proceedings of IJCAI*, 2022.

[52] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of CVPR*, 2016.

[53] Randy Hickey and Fahiem Bacchus. Speeding up assumption-based SAT. In *Proceedings of SAT*, 2019.

[54] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

[55] Nicholas Hoernle, Rafael-Michael Karampatsis, Vaishak Belle, and Kobi Gal. MultiplexNet: Towards fully satisfied logical constraints in neural networks. In *Proceedings of AAAI*, 2022.

[56] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. Harnessing deep neural networks with logic rules. In *Proceedings of ACL*, 2016.

[57] Zhiting Hu, Zichao Yang, Ruslan Salakhutdinov, and Eric Xing. Deep neural networks with massive learned knowledge. In *Proceedings of EMNLP*, 2016.

[58] Yuxiu Hua, Zhifeng Zhao, Zhiming Liu, Xianfu Chen, Rongpeng Li, and Honggang Zhang. Traffic prediction based on random connectivity in deep learning with long short-term memory. In *Proceedings of VTC-Fall*, 2018.

[59] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review*, 37, 2020.

[60] Vicky Kalogeiton, Philippe Weinzaepfel, Vittorio Ferrari, and Cordelia Schmid. Action tubelet detector for spatio-temporal action localization. In *Proceedings of ICCV*, 2017.

[61] Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt. An algebraic Prolog for reasoning about possible worlds. In *Proceedings of AAAI*, 2011.

[62] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of ICLR*, 2015.

[63] Diederik P. Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Proc. of NeurIPS*, 2014.

[64] Erich Peter Klement, Radko Mesiar, and Endre Pap. *Triangular Norms*. Springer, 2000.

[65] Bryan Klimt and Yiming Yang. The Enron Corpus: A new dataset for email classification research. In *Proceedings of ECML*, 2004.

[66] Alex Krizhevsky. Learning multiple layers of features from tiny images — Chapter 3. *Technical Report*, 2009.

[67] Maxat Kulmanov, Mohammad Asif Khan, and Robert Hoehndorf. DeepGO: Predicting protein functions from sequence and interactions using a deep ontology-aware classifier. *Bioinformatics*, 34(4), 2018.

[68] Yann LeCun, Léon Bottou, Genevieve Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade*. 2012.

[69] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5, 2004.

[70] Chu Min Li and Felip Manyà. Maxsat, hard and soft constraints. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*. 2009.

[71] Dong Li, Zhaofan Qiu, Qi Dai, Ting Yao, and Tao Mei. Recurrent tubelet proposal and recognition networks for action detection. In *Proceedings of ECCV*, 2018.

[72] Tao Li and Vivek Srikumar. Augmenting neural networks with first-order logic. In *Proceedings of ACL*, 2019.

[73] Yu Li, Sheng Wang, Ramzan Umarov, Bingqing Xie, Ming Fan, Lihua Li, and Xin Gao. DEEPre: Sequence-based enzyme EC number prediction by deep learning. *Bioinformatics*, 34(5), 2018.

[74] John W. Lloyd. *Foundations of Logic Programming, 2nd Edition*. Springer, 1987.

[75] Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward relu neural networks. *CoRR*, abs/1706.07351, 2017. URL `http://arxiv.org/abs/1706.07351`.

[76] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of NeurIPS*. 2017.

[77] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. Deepgauge: multi-granularity testing criteria for deep learning systems. In *Proceedings of ACM/IEEE ASE*. ACM, 2018.

[78] Bodhisattwa Prasad Majumder, Oana-Maria Camburu, Thomas Lukasiewicz, and Julian J. McAuley. Rationale-inspired natural language explanations with commonsense. 2021. *arXiv:2106.13876*.

[79] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. DeepProbLog: Neural probabilistic logic programming. In *Proceedings of NeurIPS*, 2018.

[80] Giuseppe Marra, Francesco Giannini, Michelangelo Diligenti, and Marco Gori. LYRICS: A general interface layer to integrate logic inference and deep learning. In *Proceedings of ECML-PKDD*, 2019.

[81] Luca Masera and Enrico Blanzieri. AWX: An integrated approach to hierarchical-multilabel classification. In *Proceedings of ECML-PKDD*, 2018.

[82] Michael McGough. How bad is Sacramento's air, exactly? Google results appear at odds with reality, some say. *Sacramento Bee*, 2018. URL `https://www.sacbee.com/news/california/fires/article216227775.html`.

[83] George Metcalfe. Fundamentals of fuzzy logics. https://www.logic.at/tbilisi05/Metcalfe-notes.pdf, Sep 2005.

[84] George Metcalfe. Fundamentals of fuzzy logics. https://www.logic.at/tbilisi05/Metcalfe-notes.pdf, 2005.

[85] Pasquale Minervini and Sebastian Riedel. Adversarially regularising neural NLI models to integrate logical background knowledge. In *Proceedings of CoNLL*, 2018.

[86] Raymond Mooney and Jude Shavlik. A recap of early work on theory and knowledge refinement. In *Proceedings of AAAI-MAKE*, 2021.

[87] Felipe Kenji Nakano, Mathias Lietaert, and Celine Vens. Machine learning for discovering missing or wrong protein function annotations — A comparison using updated benchmark datasets. *BMC Bioinformatics*, 20(1), 2019.

[88] Guillaume Obozinski, Gert R. G. Lanckriet, Charles E. Grant, Michael I. Jordan, and William Stafford Noble. Consistent probabilistic outputs for protein function prediction. *Genome Biology*, 9, 2008.

[89] Dhruvesh Patel, Jay-Yoon Lee Pavitra Dangati, Michael Boratko, and Andrew McCallum. Modeling label space interactions in multi-label classification using box embeddings. In *Proceedings of ICLR*, 2022.

[90] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: automated whitebox testing of deep learning systems. *Communications ACM*, 62(11), 2019.

[91] Rafael B. Pereira, Alexandre Plastino, Bianca Zadrozny, and Luiz H.C. Merschmann. Correlation analysis of performance measures for multi-label classification. *Information Processing & Management*, 54(3), 2018.

[92] John P. Pestian, Christopher Brew, Paweł Matykiewicz, D. J. Hovermale, Neil Johnson, K. Bretonnel Cohen, and Włodzisław Duch. A shared task involving multi-label classification of clinical free text. In *Proceedings of Workshop on BioNLP*, 2007.

[93] Luca Pulina and Armando Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *Proceedings of CAV*, 2010.

[94] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. 1993.

[95] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. ProbLog: A probabilistic Prolog and its application in link discovery. In *Proceedings of IJCAI*, 2007.

[96] Jesse Read, Bernhard Pfahringer, and Geoffrey Holmes. Multi-label classification using ensembles of pruned sets. In *Proceedings of IEEE ICDM*, 2008.

[97] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. In Wray Buntine, Marko Grobelnik, Dunja Mladenić, and John Shawe-Taylor, editors, *Machine Learning and Knowledge Discovery in Databases*, 2009.

[98] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of ACM SIGKDD*, 2016.

[99] Juho Rousu, Craig Saunders, Sándor Szedmák, and John Shawe-Taylor. Kernel-based learning of hierarchical multilabel classification models. *Journal of Machine Learning Research*, 7, 2006.

[100] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5), 2019.

[101] Mrinmaya Sachan, Kumar Avinava Dubey, Tom Mitchell, Dan Roth, and Eric Xing. Learning pipelines with limited data and domain knowledge: A study in parsing physics problems. In *Proceedings of NeurIPS*, 2018.

[102] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLoS ONE*, 10(3), 2015.

[103] Leander Schietgat, Celine Vens, Jan Struyf, Hendrik Blockeel, Dragi Kocev, and Saso Dzeroski. Predicting gene function using hierarchical multi-label decision tree ensembles. *BMC Bioinformatics*, 11, 2010.

[104] Andrew W. Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander W. R. Nelson, Alex Bridgland, Hugo Penedones, Stig Petersen, Karen Simonyan, Steve Crossan, Pushmeet Kohli, David T. Jones, David Silver, Koray Kavukcuoglu, and Demis Hassabis. Improved protein structure prediction using potentials from deep learning. *Nature*, 577, 2020.

[105] Luciano Serafini and Artur d'Avila Garcez. Logic tensor networks: Deep learning and logical reasoning from data and knowledge. In *Proceedings of NeSy-HLAI*, 2016.

[106] Luciano Serafini and Artur S. d'Avila Garcez. Logic tensor networks: Deep learning and logical reasoning from data and knowledge. *CoRR*, abs/1606.04422, 2016.

[107] Jude Shavlik. Combining symbolic and neural learning. *Machine Learning*, 14, 1994.

[108] Jude Shavlik and Geoffrey Towell. Combining explanation-based learning and artificial neural networks. In *Proceedings of the International Workshop on Machine Learning*, 1989.

[109] Carlos Nascimento Jr. Silla and Alex Alves Freitas. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22(1-2), 2011.

[110] Gurkirt Singh and Fabio Cuzzolin. Recurrent convolutions for causal 3D CNNs. In *Proceedings of ICCV Workshops*, 2019.

[111] Gurkirt Singh, Stephen Akrigg, Manuele Di Maio, Valentina Fontana, Reza Javanmard Alitappeh, Suman Saha, Kossar Jeddi Saravi, Farzad Yousefi, Jacob Culley, Tom Nicholson, Jordan Omokeowa, Salman Khan, Stanislao Grazioso, Andrew Bradley, Giuseppe Di Gironimo, and Fabio Cuzzolin. ROAD: The road event awareness dataset for autonomous driving. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[112] Ian Sommerville. *Software Engineering*. 2011.

[113] N. Spolaôr, E. A. Cherman, J. Metz, and M. C. Monard. A systematic review on experimental multi-label learning. Technical Report 362, Institute of Mathematics and Computational Sciences, University of São Paulo, 2013.

[114] A. N. Srivastava and B. Zane-Ulman. Discovering recurring anomalies in text reports regarding complex space systems. In *Proceedings of IEEE Aerospace Conference*, 2005.

[115] Russell Stewart and Stefano Ermon. Label-free supervision of neural networks with physics and domain knowledge. In *Proceedings of AAAI*, 2017.

[116] Jonathan M. Stokes et al. A deep learning approach to antibiotic discovery. *Cell*, 180(4), 2020.

[117] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proc. of CVPR*, 2016.

[118] Piotr Szymanski and Tomasz Kajdanowicz. A scikit-based python environment for performing multi-label classification. *CoRR*, abs/1702.01460, 2017. URL `http://arxiv.org/abs/1702.01460`.

[119] Ah-Hwee Tan. Cascade ARTMAP: Integrating neural computation and symbolic knowledge processing. *IEEE Transactions on Neural Networks*, 8(2), 1997.

[120] Geoffrey Towell and Jude Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1), 1994.

[121] Grigorios Tsoumakas and Ioannis Vlahavas. Random k-labelsets: An ensemble method for multilabel classification. In *Proceedings of ECML*, 2007.

[122] Grigorios Tsoumakas, I. Katakis, and I. Vlahavas. Effective and efficient multilabel classification in domains with large number of labels. In *Proceedings of ECML/PKDD — Workshop on Mining Multidimensional Data*, 2008.

[123] Grigorios Tsoumakas, A. Dimou, Eleftherios Spyromitros, V. Mezaris, I. Kompatsiaris, and I. Vlahavas. Correlation-based pruning of stacked binary relevance models for multi-label learning. In *Proceedings of International Workshop on Learning from Multi-label Data*, 2009.

[124] D. Turnbull, L. Barrington, D. Torres, and G. Lanckriet. Semantic annotation and retrieval of music and sound effects. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(2), 2008.

[125] Giorgio Valentini. True path rule hierarchical ensembles for genome-wide gene function prediction. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8(3), 2011.

[126] Emile van Krieken, Erman Acar, and Frank van Harmelen. Analyzing differentiable fuzzy implications. In *Proceedings of KR*, 2020.

[127] Emile van Krieken, Erman Acar, and Frank van Harmelen. Analyzing differentiable fuzzy logic operators. *Artificial Intelligence*, 302, 2022.

[128] Kush R. Varshney and Homa Alemzadeh. On the safety of machine learning: Cyber-physical systems, decision sciences, and data products. *Big data*, 5(3), 2016.

[129] Celine Vens, Jan Struyf, Leander Schietgat, Saso Dzeroski, and Hendrik Blockeel. Decision trees for hierarchical multi-label classification. *Machine Learning*, 73 (2), 2008.

[130] Laura von Rueden, Sebastian Mayer, Katharina Beckh, Bogdan Georgiev, Sven Giesselbach, Raoul Heese, Birgit Kirsch, Michal Walczak, Julius Pfrommer, Annika Pick, Rajkumar Ramamurthy, Jochen Garcke, Christian Bauckhage, and Jannis Schuecker. Informed machine learning — A taxonomy and survey of integrating prior knowledge into learning systems. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[131] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of CVPR*, 2018.

[132] Jonatas Wehrmann, Ricardo Cerri, and Rodrigo C. Barros. Hierarchical multi-label classification networks. In *Proceedings of ICML*, 2018.

[133] Rebecca Wexler. When a computer program keeps you in jail: How computers are harming criminal justice. *New York Times*, 2017. URL `https://www.nytimes.com/2017/06/13/opinion/how-computers-are-harming-\criminal-justice.html`.

[134] Feihong Wu, Jun Zhang, and Vasant G. Honavar. Learning classifiers using hierarchically structured class taxonomies. In *Proc. of SARA 2005*, volume 3607, 2005.

[135] Saining Xie, Tianbao Yang, Xiaoyu Wang, and Yuanqing Lin. Hyper-class augmented and regularized deep learning for fine-grained image classification. In *Proceedings of CVPR*, 2015.

[136] Yaqi Xie, Ziwei Xu, Kuldeep Meel, Mohan Kankanhalli, and Harold Soh. Embedding symbolic knowledge into deep networks. In *Proceedings of NeurIPS*, 2019.

[137] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *Proceedings of ICML*, 2018.

[138] Zhun Yang, Adam Ishay, and Joohyung Lee. NeurASP: Embracing neural networks into answer set programming. In *Proceedings of IJCAI*, 2020.

[139] Min-Ling Zhang and Zhi-Hua Zhou. ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognition*, 40(7), 2007.

[140] Min-Ling Zhang and Zhi-Hua Zhou. ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognit.*, 40(7), 2007.

[141] Zhenzhen Zou, Shuye Tian, Xin Gao, and Yu Li. mlDEEPre: Multi-functional enzyme function prediction with hierarchical multi-label deep learning. *Frontiers in Genetics*, 9, 2019.