UNIVERSITÀ DEGLI STUDI DI BARI

ALDO MORO

Department of Computer Science

Master's Degree Thesis in Machine Learning

Neural-Symbolic integration based on SPNs and abduction for classification

Supervisor:
Prof. Nicola Di Mauro

Candidate:
Marco Iannotta

Academic Year 2019/2020

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The study of human behaviour is an important part of computer science, artificial intelligence (AI), neural computation, cognitive science, philosophy, psychology, and other areas. Presupposing that behaviour is generally determined and guided by cognition and mental processing, among the most prominent tools in the modelling of behaviour are computational-logic systems mostly addressing high-level reasoning and thought processes (classical logic, nonmonotonic logic, modal and temporal logic), connectionist models of cognition and the brain mostly addressing lower-level dynamics and emergent processes (feedforward and recurrent networks, symmetric and deep networks, self-organising networks), and models of uncertainty addressing the often vague or probabilistic nature of many aspects of cognitive processing (Bayesian networks, Markov decision processes, Markov logic networks, probabilistic inductive logic programs).

The intuition motivating Neural-Symbolic integration as an active field of research is the following: in neural computing, it is assumed that the mind is an emergent property of the brain, and that computational cognitive modelling can lead to valid theories of cognition and offer an understanding of certain cognitive processes [49]. From this it is in turn assumed that connectionism should be able to offer an appropriate representational language for artificial intelligence as well. In particular, a connectionist computational theory of the mind should be able to replicate the parallelism and kinds of adaptive learning processes seen in neural networks, which are generally accepted as responsible for the necessary robustness and ultimate effectiveness of the system in dealing with commonsense knowledge. As a result, a purely symbolic approach would not be sufficient, as argued by Valiant in [50]. On the other hand, logic is firmly established as a fundamental tool in modelling of thought and behaviour [25] and by many has been viewed generally as the "calculus of computer science". However, when building models that combine learning and reasoning, one has to conciliate the methodologies of distinct areas (namely predominantly statistics and logic), in order to combine the respective advantages and circumvent the shortcomings and limitations.

From a more practical perspective, rational agents are often conceptualised as performing concept acquisition (generally unsepervised and statistical) and concept manipulation (generally supervised and symbolic) as a part of a permanent cycle of perception and action. The question of how to reconcile the statistical nature of learning with the logic nature of reasoning, aiming to build such robust computational models integrating concept acquisition and manipulation, has been identified as a key research challenge and fundamental problem in computer science [51].

In this thesis, we explore the integration between *Sum-Product networks* (instead of classical artificial neural networks usually exploited in Neural-Symbolic models) and *abductive reasoning.* A *Sum-Product Network (SPN)* is a probabilistic model, based on a rooted acyclic directed graph, in which terminal nodes represent univariate probability distributions and non-terminal nodes represent combinations (weighted sums) and products of probability functions. Their main advantage is the possibility of building tractable models from data, i.e., models that can perform several inference tasks in time proportional to the number of links in the graph. They are somewhat similar to neural networks and can address the same kinds of problems. In particular, we exploit *Random-And-Tenzorized SPNs (RAT-SPNs)*, proposed by Peharz et al. [37], which are SPNs with random structures and trained in "classical deep learning manner", i.e. employing automatic differentiation, SGD and GPU support. On the other hand, *abduction* is defined as "a syllogism in which the major premise is evident but the minor premise and therefore the conclusion only probabile". Basically, it involves forming a conclusion from the information that is known, broadly speaking trying to explain a puzzling observation. Abduction is entirely aimed at increase our knowledge, but brings the disadvantage to be extremely subjected to error, since it does not contain in itself its logic validity, but it has to be empirically validated (contrary to deduction). The integration between *RAT-SPNs* and *abduction* has been tested on a particular kind of classification task, that differs from the "standard" one because the known information about training examples is not directly their associated class but an aggregate information deriving from it. A trivial example might be learning to classify digits from 0 to 9 knowing the sum of n pairs. It is clear that facing a task like this one leads to deal with a strong uncertainty component that must be handled by appropriate mechanisms.

The remainder of the thesis is structured as follows. In Chapter 2, we provide a brief introduction about Neural-Symbolic integration, Sum-Product Networks (and specifically RAT-SPNs), abductive reasoning and finally we formally define the classification task we deal with. The proposed model, its architecture and its main features are described in detail in Chapter 3, while Chapter 4 reports information about experiments setting and obtained results. In Chapter 5 we draw conclusions and outline future works.

Keyword: Neural-Symbolic; Probabilistic Circuits; Sum-Product Networks; Logic Programming; Abduction; Classification.

Chapter 2

Background

In this chapter, we provide some background notions about the main concepts related to the thesis work. In particular, more details are given with regard to the two principal components of the proposed model (i.e., *Random And Tensorized SPNs* and *abduction*) and the faced classification task.

2.1 Neural-Symbolic integration

The goals of neural-symbolic computation are to provide a coherent, unifying view for logic and connectionism[1], to contribute to the modelling and understanding of cognition and, thereby, behaviour, and to produce better computational tools for integrated machine learning and reasoning. To this end, logic and network models are studied together as integrated models of computation. Typically, translation algorithms from a symbolic to a connectionist representation and vice-versa are employed to provide either (*i*) a neural implementation of a logic, (*ii*) a logical characterisation of a neural system, or (*ii*) a hybrid learning system that brings together features from connectionism and symbolic artificial intelligence.

From a theoretical perspective, these efforts appear well-founded. According to our current knowledge and understanding, both symbolic/cognitive and sub-symbolic/neural models—especially when focusing on physically-realisable and implementable systems (i.e. physical finite state machines) rather than strictly abstract models of computation, together with the resulting physical

[1] *Connectionism* is an approach in the fields of cognitive science that hopes to explain mental phenomena using artifical neural networks.

and conceptual limitations—seem formally equivalent in a very basic sense: notwithstanding partially differing theoretical arguments, both paradigms are considered in practice equivalent concerning computability [47]. Also from a tractability perspective, for instance in [43], equivalence in practice with respect to classical dimensions of analysis (i.e. interchangeability except for a polynomial overhead) has been established. Finally, [27] provided an *in principle* existence result, showing that there is no substantial difference in representational or problem-solving power between dynamical systems with distributed representations and symbolic systems with non-monotonic reasoning capabilities.

But while these findings provide a solid foundation for attempts at closing the gap between connectionism and logic, many questions nonetheless remain unanswered especially when crossing over from the realm of theoretical research to implementation and application, among others switching from compositional symbols denoting an idealised reality to virtually real-valued vectors obtained from sensors in the real world: although introducing basic connections and mutual dependencies between the symbolic and the subsymbolic paradigm, the levels of analysis are quite coarse and almost all results are only existential in character. For instance, while establishing the in principle equivalence described above, [27] does not provide constructive methods for how to actually obtain the corresponding symbolic counterpart to a sub-symbolic model and vice versa.

Still, over the last decades several attempts have been made at developing a general neural-symbolic framework, usually trying to apply the most popular methods of their respective time—such as currently modular deep networks. Growing attention has been given recently to deep networks where it is hoped that high-level abstract representations will emerge from low-level unprocessed data [15]. Most modern neural-symbolic systems use feedforward and recurrent networks, but seminal work in the area used symmetric networks [40] of the kind applied in deep learning, and recent work starts to address real applications of symmetric neural-symbolic networks [39]. There, in general each level of a neural-symbolic system represents the knowledge evolution of multiple agents over time. Each agent is represented by a network in this level encoding commonsense (nonmonotonic) knowledge and preferences. The networks/agents at different levels can be combined upwards to represent relational knowledge and downwards to create specialisations, following what is known as a network-fibring methodology [13]. Fibring is just one example of how principles from symbolic computation (in this case, recursion) can be used by connectionism to advance the research in this area.

One of the most relevant example of neural-symbolic integration is DeepProbLog [31], a probabilistc logic programming language that incorporates deep learning by means of neural predicates. It supports (*i*) both symbolic and sub-symbolic representations and inference, (*ii*) program induction, (*iii*) probabilistic (logic) programming and (*iv*) (deep) learning from examples. The relevance of this work lies in the fact that it is the first to propose a framework where general-purpose neural networks and expressive probabilistic-logical modeling and reasoning are integrated in a way that exploits the full expressiveness and strengths of both worlds and can be trained end-to-end based on examples.

2.2 Sum-Product Networks (SPNs)

Sum-Product Networks (SPNs) were proposed by Poon and Domingos [42] in 2011 as a modification of Darwiche's [4], [5] arithmetic circuits. Every SPN consists of a directed graph that represents a probability distribution resulting from a hierarchy of distributions combined in the form of mixtures (sum nodes) and factorizations (product nodes), as shown in Figure 2.1. SPNs, like arithmetic circuits, can be built by transforming a probabilistic graphic model [24], such as Bayesian network or a Markov Network, but they can also be learned from data. The main advantage of SPNs is that several inference tasks can be performed in time proportional to the number of links in the graph.

More formally:

Definition 2.2.1. (Sum-Product Network - SPN) An SPN C consists of a rooted acyclic directed graph such that:

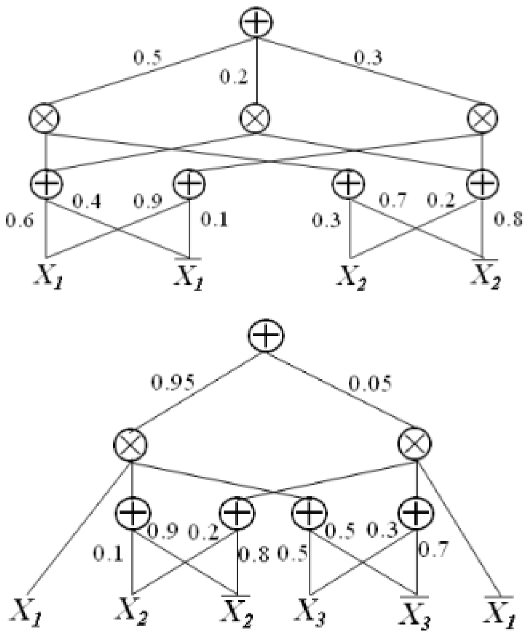
I. every leaf node represents a univariate probability distribution;

II. all the other nodes are either of type sum or product;

III. all the parents of a sum node (if any) are product nodes, and vice versa;

IV. $n_i \rightarrow n_j$ outgoing from a sum node has an associated weight, $w_{ij} \geq 0$.

FIGURE 2.1: Two examples of SPNs.



The probability distribution of each leaf node is defined on a variable V, which can have finite states or be continuous. In the first case, the distribution is usually degenerate, i.e., there is a particular value v^* of V such that $P(v^*) = 1$ and $P(v) = 0$ otherwise (*indicator*). When V is continuous, the probability distribution can be Gaussian [14], [44], piecewise polynomial [33], etc. (SPNs can be further generalized by allowing each terminal node to represent a multivariate probability density - for example, a multivariate Gaussian [9] or a Chow-Liu tree [52]).

An SPN can be built bottom-up beginning with sub-SPNs of one node and joining them with sum and product nodes. All the definitions of SPNs can be established recursively, first for one-node SPNs, and then for sum and product nodes. Similarly, all the properties of SPNs can be proved by structural induction.

If the probability distribution of a leaf node is defined on V, its *scope* is the set V. If n_i is a sum or a product node, its *scope* is the union of the scopes of its children. Let $\mathsf{ch}(\mathsf{n})$ denote the set of children of a node. The *scope* of an SPN, denoted by $\phi(\mathsf{S})$, is the scope of its root, $\phi(\mathsf{n_r})$. The variables in the scope of an SPN are sometimes called *model variables* (in contrast with *latent variables.*

Without going into detail, SPNs parameter learning consists in finding the optimal parameters for an SPN given its graph and a dataset. In generative learning the most common optimality criterion is to maximize the likelihood of the parameters of the model given a dataset, while in discriminative learning the goal is to maximize the conditional likelihood for each value of a variable C, called the class.

SPN allow to perform *probabilistic inference*, i.e. deriving the probability of one or more RVs taking a specific value or a set of values. Exact and efficient probabilistic inference is important when we want to reason and quickly take complex decisions in presence of uncertainty in real-world scenarios. Probabilistic inference can be organized in classes of queries. Informally, a class of queries comprises of all the queries that share the same "structure", and hence present the same computational challenges. More formally, a class of queries is a collection of functions that operate on probability distributions and that can be characterized by the mean of their inputs, outputs and/or by the operations their computation involves [3]. Here, we report several classes of queries which are used to compare and classify probabilistc models.

Definition 2.2.2. (Complete evidence queries - EVI) The class of complete evidence queries $\mathcal{Q}_{\mathsf{EVI}}$ is the set of queries that compute the probability $p_{\mathbf{X}}(\mathbf{x})$ for any complete evidence $\mathbf{x} \in \mathsf{val}(\mathbf{X})$.

Definition 2.2.3. (Marginal queries - MAR) The class of marginal queries $\mathcal{Q}_{\mathsf{MAR}}$ is the set of queries that compute the probability $p_{\mathbf{X}}(\mathbf{y})$ for any partial evidence $\mathbf{y} \in \mathsf{val}(\mathbf{Y})$, where $\mathbf{Y} \subseteq \mathbf{X}$.

Definition 2.2.4. (Conditional queries - CON) The class of conditional queries $\mathcal{Q}_{\mathsf{CON}}$ is the set of queries that compute the probability $p_{\mathbf{X}}(\mathbf{y}|\mathbf{z})$ for any $\mathbf{y} \in \mathsf{val}(\mathbf{Y})$ and $\mathbf{z} \in \mathsf{val}(\mathbf{Z})$, where $\mathbf{Y} \cup \mathbf{Z} = \mathbf{X}$ and $\mathbf{Y} \cap \mathbf{Z} = \varnothing$.

It should be noticed that is possible to easily define the class $\mathcal{Q}_{\mathsf{CON}}$ in terms of $\mathcal{Q}_{\mathsf{EVI}}$ and $\mathcal{Q}_{\mathsf{MAR}}$ by noting that any conditional query $p(x|y)$ can be rewritten as $p(x,y)/p(y)$.

Definition 2.2.5. (Maximum a posteriori queries - MAP) The class of maximum a posteriori queries $\mathcal{Q}_{\mathsf{MAP}}$ is the set of queries that compute:

$$\arg\max_{z \in \mathsf{val}(\mathbf{Z})} p_{\mathbf{X}}(\mathbf{z}|\mathbf{y}) = \arg\max_{z \in \mathsf{val}(\mathbf{Z})} p_{\mathbf{X}}(\mathbf{z}, \mathbf{y})$$

where $\mathbf{Y} \cup \mathbf{Z} = \mathbf{X}$ and $\mathbf{Y} \cap \mathbf{Z} = \varnothing$.

More classes of queries can be defined by "combining" the previous ones. This is the case for marginal MAP queries ($\mathcal{Q}_{\mathsf{MMAP}}$), involving marginalization over one subset of RVs and maximization over another one.

Recently, new advanced probabilistic queries have been introduced in literature. These include computing the probability of an event described as a complex logical sentence (e.g., involving disjunctions) [2]; computing expectations of probabilistic predictive models [22] and information theoretic quantities of a distribution such as its entropy or the Kullback-Leibler divergence between distributions [28]. In the following, we will refer to the class of advanced queries as $\mathcal{Q}_{\mathsf{ADV}}$.

Several well-known structural properties allow us to categorize PCs w.r.t. the classes of queries they make tractable. In the following, we first report the definition of tractability and then such structural properties indicating the class of queries they make tractable if satisfied.

Definition 2.2.6. (Tractability) A class of queries $\mathcal{Q}$ is tractable on a family of probabilistic models $\mathcal{M}$ iff any query $q \in \mathcal{Q}$ on a model $\mathsf{m} \in \mathcal{M}$ can be computed in time $O(poly(|\mathsf{m}|))$, where $|\mathsf{m}|$ is the size of the model.

Definition 2.2.7. (Smoothness, *aka* Completeness) A probabilistic circuit is ***smooth*** if for each sum node S it hols that:

$$\forall \mathsf{C}_1, \mathsf{C}_2 \in \mathsf{ch}(\mathsf{S}) : \phi(\mathsf{C}_1) = \phi(\mathsf{C}_2) \tag{2.1}$$

Definition 2.2.8. (Decomposability) A probabilistic circuit is ***decomposable*** if for each product node P it hols that:

$$\forall \mathsf{C}_1, \mathsf{C}_2 \in \mathsf{ch}(\mathsf{P}), \mathsf{C}_1 \neq \mathsf{C}_2 : \phi(\mathsf{C}_1) \cap \phi(\mathsf{C}_2) = \varnothing \tag{2.2}$$

To allow for efficient inference, PCs should satify the structural constraints 2.1 and 2.2. In that way, all nodes in an PC recursively define a distribution over their respective scopes: the leaves are distributions by definition, sum nodes are mixtures of their child distributions, and products are factorized distributions, assuming (conditional) independence among the scopes of their children. Smooth and decomposable PCs enable the tractable computation of any marginal query (MAR inference) and thus also of any conditional one (CON inference).

Definition 2.2.9. (Determinism, *aka* Selectivity) A probabilistic circuit is ***deterministic*** if for every sum node S and input $\mathbf{x}$, at most one of the children of S has a non-zero output, i.e. children distribuions have disjoint support.

A deterministic sum defines a mixture model whose components have disjoint support, enabling tractable MAP queries. A decomposable and deterministic PC ensures MAP tractable queries.

Definition 2.2.10. (Vtree) A vtree $\mathcal{V}$ of a set of variables $\mathbf{X}$ is an n-ary tree where every leaf node uniquely denotes a RV and every internal node denotes the set of RVs equal to the union of the RVs denoted by its children, so that the root of $\mathcal{V}$ denotes $\mathbf{X}$. In other words, a vtree $\mathcal{V}$ over $\mathbf{X}$ represent a hierarchical decomposition of the RVs in $\mathbf{X}$.

Definition 2.2.11. (Structured-Decomposability) A probabilistic circuit is normalized for a vtree $\mathcal{V}$ if the scope of every product node P decomposes over its children as its corresponding node, i.e. the one denoting the scope of P. A circuit is ***structured decomposable*** if it is normalized for some vtree. The circuit is then decomposable.

A trivial test for structured decomposability is ensuring that for every possible pair of product nodes $\langle \mathsf{P}_1, \mathsf{P}_2 \rangle$ in a PC, either $\phi(\mathsf{P}_1) \subseteq \phi(\mathsf{P}_2)$ or $\phi(\mathsf{P}_2) \subseteq \phi(\mathsf{P}_1)$.

By enforcing structured decomposability, several classes of advanced probabilistic queries, which we denoted as $\mathcal{Q}_{\mathsf{ADV}}$, become computable exactly and efficiently.

2.2.1 Random And Tensorized SPNs (RAT-SPNs)

Sum-product networks are an excellent architecture that allows to efficiently evaluate likelihoods, as well arbitrary marginalization and conditioning tasks. Nevertheless, SPNs have not been fully explored as serious deep learning models, likely due to their special structural requirements, which complicate learning. Peharz et al. [37] make a drastic simplification and use random SPN structures which are trained in a "classical deep learning manner", i.e. employing automatic differentiation, SGD and GPU support. The resulting models, called *Random And Tensorized SPNs* (RAT-SPNs), yeld prediction results comparable to deep neural networks, while stille being interpretable as generative model and maintaining well-calibrated uncertainties. This property makes them highly robust under missing input features and enables them to naturally detect outliers and peculiar samples.

In order to construct Random And Tensorized SPNs, a region graph ([42], [14], [35]) is used as an abstract representation of the network structure. Given a set of RVSs $\mathbf{X}$, a *region* $\mathbf{R}$ is defined as any non-empty sub-set of $\mathbf{X}$. Given any region $\mathbf{R}$, a *K-partition* P of $\mathbf{R}$ is a collection of K non-empty, non overlapping subsets $\mathbf{R}_1$,...,$\mathbf{R}_K$ of $\mathbf{R}$, whose union is again $\mathbf{R}$. In RAT-SPNs only 2-partitions are considered, so that all product nodes in resulting SPNs have exactly two children. This assumption, frequently made in SPN literature, simplifies SPN design and seems not to impair performance.

A *region graph* R over $\mathbf{X}$ is a DAG whose nodes are regions and partitions such that:

I. $\mathbf{X}$ is a region in R and has no parents (root region);

II. all other regions have at least one parent;

III. all children of regions are partitions an all children of partitions are regions (i.e., R is bipartitive);

IV. if P is a child of $\mathbf{R}$, then $\bigcup_{\mathbf{R}^* \in P} \mathbf{R}^* = \mathbf{R}$;

V. if $\mathbf{R}$ is a child of P, then $\mathbf{R} \in P$.

From this definition it follows that a region graph dictates a hierarchical partition of the overall scope $\mathbf{X}$. We denote regions which have no child partitions as *leaf regions*.

Given a region graph, a corresponding smooth (complete) and decomposable SPN can be constructed, as illustrated in Algorithm 1. Random region graphs are constructed using the simple procedure depicted in Algorithm 2. The root region is randomly divided into two sub-regions of equal size and . This recursive splitting mechanism is repeated R times. Figure 2.2 shows an example SPN with $C = 3$, $S = 2$ and $I = 3$, following Algorithm 2 and subsequentially Algorithm 1. Note that this construction scheme yields SPNs where input distributions, sums, and products can be naturally organized in alternating layers. Similar to classical multilayer perceptrons (MLPs), each layer takes inputs from its directly preceding layer only. Unlike MLPs, however, layers in RAT-SPNs are connected block-wise sparsely in a random fashion. Thus, layers in MLPs and RAT-SPNs are hardly comparable; however, we suggest to understand each pair of sum and product layer to be roughly corresponding to one layer in an MLP: sum layers play the role of (block-wise sparse) matrix multiplication and product layers as non-linearities (or, more precisely, bilinearities of their inputs). The input layer, containing the SPN's leaf distributions, can be interpreted as a non-linear feature extractors.

Algorithm 1 Construct SPN from Region Graph

Input: R: region graph; C: number of classes; S: number of sum nodes in regions, which are neither leaf nor root regions; I: number of input distributions per leaf region.

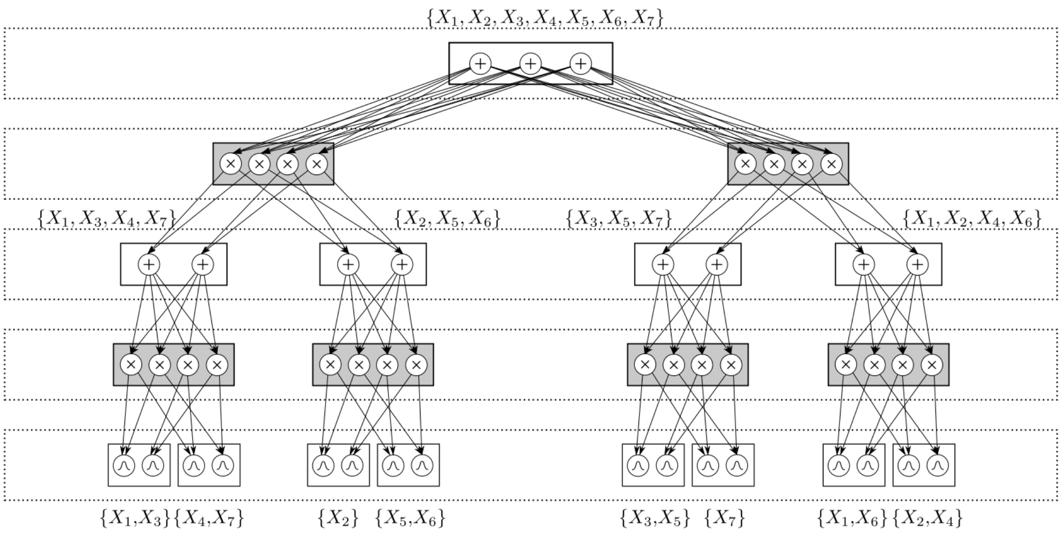
Output: A smooth and decomposable SPN deriving from region graph R

```
1: Make empty SPN
2: for each $\mathbf{R} \in R$ do
3:     if $\mathbf{R}$ is a leaf region then
4:         Equip $\mathbf{R}$ with $I$ distribution nodes
5:     else if $\mathbf{R}$ is the root region then
6:         Equip $\mathbf{R}$ with $C$ sum nodes
7:     else
8:         Equip $\mathbf{R}$ with $S$ sum nodes
9:     end if
10: end for
11: for each $P = \{\mathbf{R}_1, \mathbf{R}_2\} \in R$ do
12:     Let $\mathbf{N_R}$ be the nodes for region $\mathbf{R}$
13:     for each $N_1 \in \mathbf{N_{R_1}}, N_2 \in \mathbf{N_{R_2}}$ do
14:         Introduce product $P = N_1 \cdot N_2$
15:         Let P be a child for each $N \in \mathbf{N_{R_1 u \cup R_2}}$
16:     end for
17: end for
18: return SPN
```

Algorithm 2 Random Region Graph

```
Input: $\mathbf{X}$: set of RVs; $D$: depth; $R$: number of recursive splits.
Output: A random region graph
1: Create an empty region graph $\mathfrak{R}$
2: Insert $\mathbf{X}$ in $\mathfrak{R}$
3: for each $r = 1...R$ do
4:     SPLIT$(R, \mathfrak{R}, D)$
5: end for
1: procedure SPLIT$(R, \mathbf{R}, D)$
2:     Draw balanced partition $P = \{\mathbf{R}_1, \mathbf{R}_2\}$ of $\mathbf{R}$
3:     Insert $\mathbf{R}_1, \mathbf{R}_2$ in $R$
4:     Insert $P$ in $R$
5:     if $D > 1$ then
6:         if $|\mathbf{R}_1| > 1$ then
7:             SPLIT$(R, \mathbf{R}_1, D - 1)$
8:         end if
9:         if $|\mathbf{R}_2| > 1$ then
10:            SPLIT$(R, \mathbf{R}_2, D - 1)$
11:        end if
12:    end if
13: end procedure
```

FIGURE 2.2: Example RAT-SPN over 7 RVs $\{X_1,...,X_7\}$, using parameters $C = 3$, $D = 2$, $R = 2$, $S = 2$ and $I = 2$ in Algorithm 2 and Algorithm 1.



As mentioned above, RAT-SPNs are trained in a "classical deep learning manner", i.e. employing automatic differentiation, SGD and GPU support. More specifically, let $X = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)$ be a training set of input $\mathbf{x}_n$ and class labels $\mathbf{y}_n$. Furthermore, let S_C be the c^{th} output of the RAT-SPN and $\mathbf{w}$ all SPN parameters. RAT-SPNs training is performed by minimizing the objective

$$O(\mathbf{w}) = \lambda \mathbf{CE}(\mathbf{w}) + (1 - \lambda) n\mathbf{LL}(\mathbf{w}) \tag{2.3}$$

where $\mathbf{CE}(\mathbf{w})$ is the cross-entropy

$$\mathbf{CE}(\mathbf{w}) = -\frac{1}{N} \sum_n log \frac{S_{y_n}(\mathbf{x}_n)}{\sum_{y'} S_{y'}(\mathbf{x}_n)} \tag{2.4}$$

and n$\mathbf{LL}(\mathbf{w})$ is the normalized negative log-likelihood

$$n\mathbf{LL}(\mathbf{w}) = -\frac{1}{N|\mathbf{X}|} \sum_n log S_{y_n}. \tag{2.5}$$

When setting $\lambda = 1$, purely cross-entropy is optimized (discriminative setting), while for $\lambda = 0$ maximum likelihood training is performed (generative setting). For $0 < \lambda < 1$, we have a continuum of hybrid objectives, trading off the generative and dsicriminative character of the model.

The size of RAT-SPN can be easily controlled via the structural parameters D, R, S and I. RAT-SPNs with many parameters, however, tend to overfit just like regular neural networks, which requires regularization. One of the classical techniques that boosted deep learning models is the well-known *dropout* heuristic [48], setting inputs and/or hidden units to zero with a certain probability p, and rescaling the remaining units by $\frac{1}{p}$. In RAT-SPNs two types of dropout are exploited:

I. *Dropout at inputs.* It essentially consists in marking input features as missing at random;

II. *Dropout at sums.* As discussed in [[42], [53], [36]], sum nodes in SPNs can be interpreted as marginalized latent variables, akin to the latent variable interpretation in mixture models. In RAT-SPNs a whole region can be interpreted as a single latent variable, and the weights of each sum node in this region as the conditional distribution of this variable. While the latent variables are not observed, the idea is to employ a simple probabilistic version of dropout, by introducing artificial observations for them. For example, if the sum nodes in a particular region have K children (i.e. the corresponding variable Z has K states), then the artificial information that Z assumes a state in some subset of $\{1, ..., K\}$ can be introduced. By doing this for each latent variable in the network, the results essentially consists in selecting a small sub-structure of the whole SPN to explain the data.

2.3 Abduction

A central theme in the study of human reasoning is the construction of explanations that give us an understanding of the world we live in. Broadly speaking, abduction is a reasoning process invoked to explain a puzzling observation. A typical example is a practical competence like medical diagnosis. When a doctor observes a symptom in a patient, he hypothesizes about its possible causes, based on his knowledge of the causal relations between diseases and symptoms.

This is a practical setting. Abduction also occurs in more theoretical scientific contexts. For instance, it has been claimed [38] that when Kepler discovered that Mars had an elliptical orbit, his reasoning was abductive.

Charles Sanders Peirce (1839-1914), the founder of American pragmatism was the first philosopher to give a logical form to abduction. In his early theory Peirce proposed three modes of reasoning: deduction, induction and abduction, each oh which corresponds to a syllogistic form, illustrated by the following, often quoted example [38]:

DEDUCTION

Rule - All the beans from this bag are white.

Case - These beans are from this bag.

Results - These beans are white.

INDUCTION

Case - These beans are from this bag.

Result - These beans are white.

Rule - All the beans from this bag are white.

ABDUCTION

Rule - All the beans from this bag are white.

Results - These beans are white.

Case - These beans are from this bag.

Of these, deduction is the only reasoning which is completely certain, infering its 'Results' as a necessary conclusion. Induction produces a 'Rule' validated only in the 'long run' [38], and abduction merely suggests that something may be 'the Case' [38]. The abduction is, according to Peirce, the only reasoning form that can increase our knowledge, i.e. suggest new ideas, gueess, predict. Actually, all three inference mechanisms described above allow to increase our knowledge, even if in different order and measure, but only abduction is entirely aimed at this increase. It is also true that abduction is the most subjected to error. As with induction, abduction does not contain in itself its logic validity and it has to be empirically validated; it will never be possible to have an absolute validation, but only a probabilistic one.

In the context of formal logic, abduction is often defined as follows:

Definition 2.3.1. (Abduction) Given a logical theory T representing the expert knowledge and a formula Q representing an observation on the problem domain, abductive inference searches for an explanation formula ε such that:

- ε is satisfiable[2] w.r.t. T and
- it holds that[3] $T \models \varepsilon \rightarrow Q$.

In general, ε will be subjected to further restrictions such as the aforementioned minimality criteria and criteria restricting the form of the explanation formula (e.g. by restricting the predicates that may appear in it). This view defines an abductive explanation of an observation as a formula which *logically entails* the observation. However, some have argued, sometimes with good reasons, that it is more natural to view an explanation as a *cause* for the observation. A well-known example is as follows: the disease paresis is caused bya latent untreated form of syphilis. The probability that latent untreated syphilis leads to paresis is only 25%. Note that in this context, the direction of entailment and causalityare opposite: syphilis is the cause of paresis but does not entail it, while paresis entails syphilis but does not cause it. Yet a doctor can *explain* paresis by the hypothesis of syphilis while paresis cannot account for an *explanation*s for syphilis.

In practice, examples where causation and entailment do not correspond are rare. It turns out that in many applications of abduction in AI, the theory T describes explicit *causality information.* This is notably the case in model-based diagnosis and in temporal reasoning, where theories describe effects of actions. By restricting the explanation formulas to the predicates describing

[2]If ε contains free variables, $\exists(\varepsilon)$ should be satisfiable w.r.t. T.

[3]Or, more general, if Q and ε contain free variables: $T \models \forall(\varepsilon \rightarrow Q)$

primitive causes in the domain, an explanation formula which entails an observation gives a cause for the observation. Hence, for this class of theories, the logical entailment view implements the causality view on abductive inference.

In the context of logic programming, the study of abductive inference started at the end of the eighties as an outcome of different attempts to use logic programming for solving AI-problems. Facing the limitations of standard logic programming for solving these problems, different researchers proposed to extend logic programming with abduction. Eshghi [11] introduced abduction in logic programming in order to solve planning problems in the Event Calculus [26]. In this approach, abduction solves a planning goal by explaining it byan ordered sets of events -a plan- that entails the planning goal. This approach was further explored byShanahan [46], Missiaen et al. [11],[32], Denecker [8], Jung [45] and recently in [17],[20]. In parallel to these studies of abduction as an inferential method, Eshghi and Kowalski [12] and later Kakas and Mancarella in [21],[16] and Dung in [10], used abduction as a semantical device to describe the non-monotonic semantics of Logic Programming (in a wayanalogous to Poole in [41]). In [7],[6], abductive logic programming was investigated from a knowledge representation point of view and its suitabilityfor representing and reasoning on incomplete information and definitional and assertional knowledge was shown. For these reasons, Abductive Logic Programming1 (ALP) [18],[19] was recognized as a promising computational paradigm that could resolve many limitations of logic programming with respect to higher level knowledge representation and reasoning tasks. ALP has manifested itself as a framework for declarative problem solving suitable for a broad collection of problems.

2.4 Classification

Classification is a type of supervised learning that consists in building a classifier from a set of examples labeled by their classes or precedents (learning step) and then predicting the class of new examples by using the generated classifiers (classification step). More formally, let:

- X be the instance space (whose elements correspond to the observations);

- C be a non-empty finite set (whose elements correspond to the classes/labels);
- $c : X \rightarrow C$ be the classification function to be learned;
- H be the hypothesis space;
- $h \in H$ such that $h : X \rightarrow C$.

Given a sequence of training examples $\{(\mathbf{x}_1, c_1), ..., (\mathbf{x}_n, c_n)\}$, where $\mathbf{x}_i \in X$, $c_i \in C$ and $c_i = c(\mathbf{x}_i)$, the goal is to find h such that $h(\mathbf{x}_i) = c(\mathbf{x}_i)$ $\forall \mathbf{x}_i \in X$.

In this thesis work, we are interested in a slightly different classification task, in which the known information about training examples is not directly their associated class but an aggregate information deriving from it. More formally, this task differs from the one described above in the following respects:

- let $f : C \times ... \times C \rightarrow F$ be an aggregate non-injective function with arity m, where F is a non-empty finite set;
- training examples form is $\{((\mathbf{x}_1, ..., \mathbf{x}_m), f_1), ..., ((\mathbf{x}_{n-m+1}, ..., \mathbf{x}_n), f_{\frac{n}{m}})\}$, where $\mathbf{x}_i \in X$, $f_i \in F$ and

$$f_i = f(c_{m(i-1)+1}, ..., c_{m(i-1)+1+m}) = f(c(\mathbf{x}_{m(i-1)+1}), ..., c(\mathbf{x}_{m(i-1)+1+m})). \tag{2.6}$$

Basically, training examples are grouped into tuples with cardinality m, on whose corresponding labels a function f is applied. Note that:

- the classifier knows how to compute f (and this information will be encapsulated in the knowledge domain of the symbolic module);
- if f was injective, such classification task just would come back to the "standard" one, since the model would be able to deterministically derive the correct classes from the aggregate information.

Chapter 3

Model

This chapter aims to totally describe the proposed model. Before going into the description details, Section 3.1 tries to frame the difficulties that the model has to deal with, in trying to solve a classification task of the one described in Section 2.4. These observations aim to provide a preliminary justification and comprehension of the reasons behind specific choices.

3.1 What is the challenge?

TABLE 3.1: Performance of some configurations of a simple Neural-Symbolic model. All accuracy values are in percentage and approximated to the second digit after the decimal place.

depth	distributions	splits	sums	dropout input	dropout sums	valid acc	test acc
1	10	9	10	0.5	0.75	95.03	94.88
1	10	9	10	0.5	1	27.55	26.78
1	10	9	10	0.75	1	97 .01	96.11
1	**10**	**9**	**10**	**0.25**	**0.25**	**6.12**	**5.74**
1	10	9	10	0.50	0.25	94.51	94.65
1	10	9	10	0.25	0.5	94.48	94.08
1	33	40	10	0.25	0.5	92.92	92.75
1	33	40	10	0.5	0.25	59.13	58.93
1	33	40	10	0.75	0.25	84.22	84.75
1	33	40	10	0.25	1	19.83	19.96
1	33	40	10	0.5	1	96.27	96.24
1	33	40	10	0.5	0.5	96.53	96.16

Table 3.1 shows the performance of some configurations (for brevity) of a very simple Neural-Symbolic model, whose subsymbolic module training is based on the most probable abduction among the ones suggested by the symbolic module;

the task consists in learning to classify digits from 0 to 9, knowing the sum of n pairs of them. For the moment, further details about the classification task and the model under consideration are intentionally omitted, in order to focus the attention on the findings. Looking at the results, a significant variance in terms of accuracy is evident among the tested configurations. In order to investigate this anomalous behaviour and to understand if such a variance depends on the specific configuration (i.e. on the the sub-symbolic hyperparameters) or not, the configuration highlighted in bold in Table 3.1 has been runned multiple times.

TABLE 3.2: Multiple runs of configuration higlighted in bold in Table 3.1. Reported values refer to accuracy computed on validation set (in percentage).

	run 1	**run 2**	**run 3**
epoch 0	14.98	47.21	33.91
epoch 30	11.65	70.85	52.15
epoch 50	10.32	90.12	78.92
epoch 100	12.79	93.22	88.34

Results of Table 3.2 clearly show that the variance in the model performance does not depend on the considered combination of hyperparameters, since multiple runs of the same configuration bring to very different values in terms of accuracy. With the aim of furtherly investigate this attitude, Tables 3.3, 3.4 and 3.5 show the confusion matrices of the first epoch of such runs, calculated taking into account the real labels of the considered digits and the labels selected by the sub-symbolic module among the ones suggested by symbolic one. In particular, since each value has been obtained dividing the observed frequency by the correspondent row sum, the element in position (i, j) indicates the ratio with which the $i - th$ digit has been labeled as the $j - th$ one by the model (the reverse observation can NOT be done). These matrices point out that the strong variance in terms of accuracy lies in the variability with whom the examples are labelled on the basis of the symbolic module abductions and the subsymbolic module probabilities. Since the initialization of the subsymbolic module is random and since the most probable abduction is chosen according to the probability computed by the SPN, selected abductions to train the model are random likewise. Therefore, since the model has no way to understand if it is going well or not, the first taken direction significantly influences all the training procedure. Essentially, the model goodness depends on chance: only if the first abduced labels are (by chance) correct, the model performance are quite good. It is clear that this behaviour can't be considered as a real learning. Furthermore, the more the task becomes difficult (in terms of the number of possible abductions), the more the chance of randomly abducing the correct labels decreases.

The considerations made up to now emphasise that, since the abduction is not a reasoning process that returns always true facts and especially because the initialization of the subsymbolic model is randomic, we need to try to force the model not to learn from abductions it is not sufficently sure of, trying to guide somehow it towards the real most probable ones. All the choices explained in the following section are oriented in this perspective.

Table 3.3: Confusion matrix between real labels (row) and abduced labels (column) for run 1 of Table 3.2. Each reported value has been obtained dividing the observed frequency by the correspondent row sum and approximated to the second digit after the decimal place. Accuracy on validation set: 14.98%

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	**0.27**	0.17	0.27	0.06	0.06	0.04	0.03	0.09	0.00	0.00
1	0.14	**0.21**	0.17	0.08	0.14	0.25	0.00	0.00	0.00	0.00
2	0.14	0.11	**0.26**	0.09	0.12	0.13	0.03	0.06	0.05	0.02
3	0.06	0.09	0.23	**0.09**	0.09	0.25	0.06	0.06	0.05	0.02
4	0.04	0.06	0.17	0.09	**0.09**	0.31	0.12	0.06	0.05	0.02
5	0.06	0.06	0.08	0.07	0.13	**0.26**	0.11	0.09	0.09	0.05
6	0.03	0.02	0.10	0.04	0.16	0.20	**0.16**	0.19	0.06	0.05
7	0.02	0.01	0.05	0.04	0.08	0.27	0.16	**0.10**	0.16	0.11
8	0.00	0.03	0.05	0.02	0.09	0.15	0.13	0.10	**0.31**	0.11
9	0.00	0.00	0.06	0.02	0.03	0.20	0.13	0.09	0.22	**0.25**

Table 3.4: Confusion matrix between real labels (row) and abduced labels (column) for run 2 of Table 3.2. Each reported value has been obtained dividing the observed frequency by the correspondent row sum and approximated to the second digit after the decimal place. Accuracy on validation set: 47.21%

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	**0.39**	0.16	0.18	0.08	0.05	0.05	0.05	0.03	0.01	0.01
1	0.09	**0.41**	0.11	0.18	0.06	0.01	0.01	0.11	0.02	0.00
2	0.12	0.13	**0.25**	0.17	0.1	0.08	0.05	0.04	0.04	0.02
3	0.08	0.10	0.13	**0.29**	0.12	0.07	0.08	0.04	0.04	0.05
4	0.03	0.11	0.10	0.15	**0.15**	0.05	0.12	0.23	0.03	0.03
5	0.03	0.04	0.10	0.16	0.11	**0.12**	0.16	0.18	0.04	0.06
6	0.04	0.03	0.04	0.11	0.06	0.09	**0.37**	0.12	0.07	0.06
7	0.01	0.05	0.03	0.04	0.07	0.04	0.07	**0.47**	0.17	0.05
8	0.00	0.02	0.09	0.04	0.05	0.07	0.14	0.21	**0.25**	0.14
9	0.01	0.00	0.04	0.08	0.03	0.04	0.11	0.26	0.23	**0.21**

Table 3.5: Confusion matrix between real labels (row) and abduced labels (column) for run 3 of Table 3.2. Each reported value has been obtained dividing the observed frequency by the correspondent row sum and approximated to the second digit after the decimal place. Accuracy on validation set: 33.91%

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	**0.39**	0.12	0.22	0.08	0.03	0.03	0.10	0.01	0.00	0.02
1	0.10	**0.37**	0.06	0.23	0.07	0.04	0.01	0.11	0.01	0.00
2	0.14	0.07	**0.33**	0.14	0.08	0.04	0.10	0.03	0.04	0.03
3	0.06	0.12	0.06	**0.41**	0.08	0.10	0.03	0.05	0.04	0.05
4	0.07	0.07	0.09	0.14	**0.19**	0.10	0.11	0.19	0.03	0.03
5	0.04	0.04	0.07	0.18	0.07	**0.22**	0.08	0.17	0.05	0.08
6	0.07	0.01	0.04	0.09	0.05	0.05	**0.44**	0.11	0.07	0.07
7	0.01	0.07	0.02	0.05	0.06	0.06	0.04	**0.54**	0.07	0.08
8	0.01	0.02	0.08	0.06	0.05	0.10	0.11	0.25	**0.17**	0.15
9	0.02	0.00	0.02	0.09	0.02	0.07	0.08	0.34	0.13	**0.23**

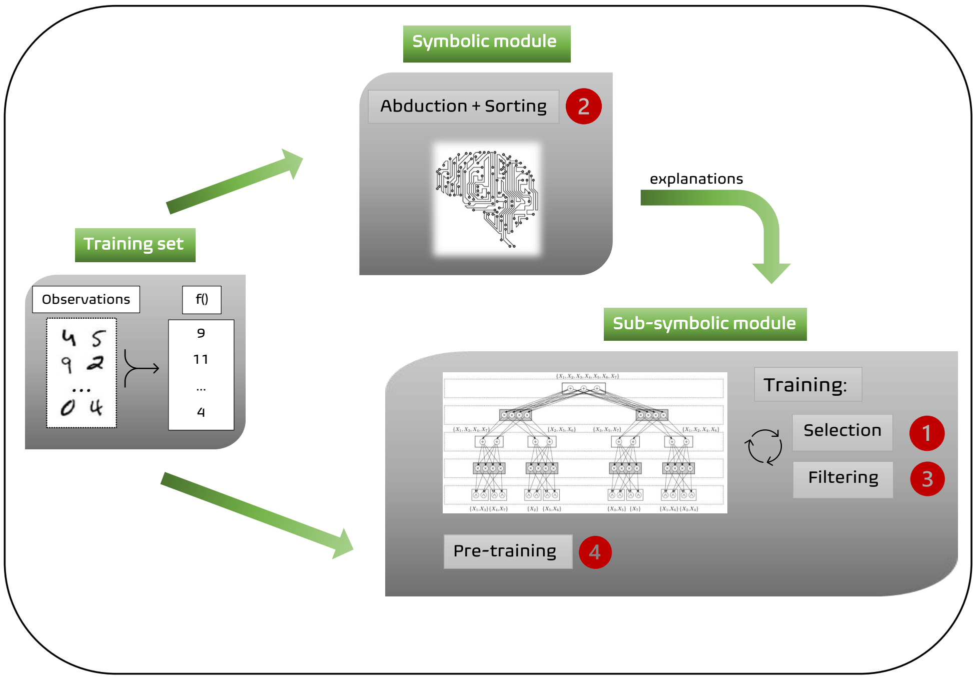
3.2 Architecture and features

Figure 3.1 shows the general architecture of the implemented model. Essentially:

- the symbolic module, containing the domain knowledge, takes in input the aggregate information (elements of F deriving from the application of function f, see Section 2.4) related to the observations and returns all the possible abductions;
- the sub-symbolic module, which consists in a Random And Tensorized SPN (RAT-SPN), receives such explanations together with the observations and, on the basis of the most probable ones (i.e., abductions), performs the training process. Note that RAT-SPN is unuware about f, it just works on observations and labels abduced by symbolic module.

In the following, we describe in detail the implemented features (in the figure, red numbers from 1 to 4) that allow the final model to perform better than the one described in Section 3.1.

FIGURE 3.1: Model architecture.



3.2.1 Abduction selection

Since the function f is not injective, the symbolic module suggests more than one possible abduction for each observation of the training set. The first key-point is to decide how to select the most suitable (ideally the correct) one in order to perform the training process. Let:

- A be the set of abductions returned by symbolic module for observations $\mathbf{x}_r, ..., \mathbf{x}_{r+m} \in X$, where m is f arity;
- $a = (c^a_r, ..., c^a_{r+m})$ be a generic element of A, such that labels $c^a_r, ..., c^a_{r+m}$ respectively correspond to observations $\mathbf{x}_r, ..., \mathbf{x}_{r+m}$;
- c be the classification function to be learned;
- θ be the set of RAT-SPN parameters;

The most probable abduction a^* is the one that maximizes the joint probability that each considered observation is predicted by RAT-SPN as the suggested label. Since observations are supposed to be *i.i.d.* (i.e., indipendent and identically distributed), the joint probability is computed as the product of single probabilities:

$$\begin{aligned} a^* = \arg\max_{a \in A} p(a|\theta) &= \arg\max_{a \in A} p(c(\mathbf{x}_r) = c^a_r, ..., c(\mathbf{x}_{r+m}) = c^a_{r+m}|\theta) \\ &= \arg\max_{a \in A} \prod_{j=r}^{r+m} p(c(\mathbf{x}_j) = c^a_j|\theta) \end{aligned} \tag{3.1}$$

Note that, since the considered probabilities depend on the RAT-SPN parameters, it is not said that the most probable abduction stays the same among the different training epochs (remember that RAT-SPN training works like neural networks one, see Section 2.2.1). Ideally, if the network is correctly learning, the selection will become more and more reliable (i.e., oriented to the correct labels) with incrasing epoch.

3.2.2 Observations sorting

Even if the function f is not injective, it is not said that symbolic module assigns the same number of possible abduction to each observation and, moreover, some of them might have just one possible abduction. For example, let us consider the sum of a pair of digits from 0 to 9:

I. if $sum(c_1, c_2) = 4$, then $A = \{(0,4), (1,3), (2,2), (3,1), (4,0)\}$ and $|A| = 5$;

II. if $sum(c_1, c_2) = 2$, then $A = \{(0,2), (1,1), (2,0)\}$ and $|A| = 3$;

III. if $sum(c_1, c_2) = 0$, then $A = \{(0,0)\}$ and $|A| = 1$;

The idea deriving from this observation is to ensure that, at each epoch, RAT-SPN performs the training on the observations sorted by their possible number of abductions. In this way, since the network is less good at classifying at the beginning of each epoch (i.e., for the first batches) rather than at the end, we try to reduce its possible choices at the beginning, trusting that it will perform better predictions going forward. With regard to the example above, observations would be sorted in the following way: III ($|A| = 1$), II ($|A| = 3$), I ($|A| = 5$).

It is pointed out that such sorting is performed by the symbolic module, since it is striclty related to the abductive process; the sub-symbolic model just receives the possible abductions and the order in which it has to consider the observations during the training, ignoring the way in which this sorting has been performed.

3.2.3 Abduction threshold

The type of classification task we are dealing with (see Section 2.4) is intrinsically characterized, by its nature, by a strong uncertainty factor. The labels selected by the sub-symbolic module among the ones suggested by the symbolic one might not always be correct, especially in the first training epochs when the model has not learned enough yet. Even if RAT-SPNs, as will be showed in Section 4.2, turn out to be very good at handling cases of mislabeling, we decided to provide a mechanism to filter the selected labels and consequently the corresponding observations from the training set, according to a specific certainty degree. Basically, the idea is to ensure that the model mainly learns from labels of which it is sufficienlty confident. Labels and observations are filtered on the basis of a value that essentially measures the probability difference between the first two most probable abductions. More precisely, let a_1 and a_2 be the first two probable abductions selected by the sub-symbolic module and respectively $p(a_1)$ and $p(a_2)$ be their probabilities computed by RAT-SPN; the value is calculated as follows:

$$v_{dif} = \frac{p(a_1) - p(a_2)}{p(a_1)} \times 100 \tag{3.2}$$

If v_{dif} is greater than the specified threshold (which is an hyper-parameter of the entire model), then the specific observations and the corresponding selected labels are used in the training process, as usual; otherwise, they are discarded. Naturally, it makes sense to use this threshold only when the possible number of abductions is strictly greater than 1. Note that, since v_{dif} depends on the probability values returned by RAT-SPN, it is computed and compared again with the threshold at each epoch, because both $p(a_1)$, $p(a_2)$ and the same a_1, a_2 might be different. Therefore, it intuitively follows that the more the model learns the more observations will be considered, because it will become more and more good at disambiguating the observations. As we will see from the experiments, in a certain sense, increasing the threshold ensures a sort of "cool start", forcing the model to train from few examples especially in the first epochs, when it is more uncertain about the labels to choose. In some situations, depending on the specific classification task, this may result in starting to learn only from a subset of the labels (e.g., in the case of the sum of 2 digits, only from 0 and 9, since they correspond to the only certain labels), extending the training bit by bit to other classes.

3.2.4 Pre-training

In some situations it might happen that the number of observations for which the number of possible abductions is one (e.g., sum of two digits equal to zero) is not sufficient to allow the model to learn enough in order to be able to select the correct labels later, when the labeling is not as much certain. Nevertheless, if we know for sure the labels associated to some (even if few) observations, we can try in some way to look for other observations of the same class from the training set, on which to perform a *pre-training* of the model and try to solve, at least partially, the initialization problem of RAT-SPN (which is totally randomic).

For this purpose, we decided to exploit the k-Means clustering [30], or Lloyd's algorithm [29], which is an iterative, data-partitioning algorithm that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster. Naturally, k-Means doesn't provide the same performance of other more sophisticated algorithms, but, in this context, we are more interested in performing a fast clustering than a more accurate one, since our aim is just to initialize the model before the real training process.

Once the k-Means has been perfomed, given m observations for which we know for sure that the label is c, we select the correspondent cluster, that is the one associated to observations. Since the correspondent cluster might not be unique, we perform a majority vote among the m observations in order to choose the most frequent one; in case of equality, no cluster is selected for the pre-training. Hence, RAT-SPN is pre-trained for few epochs on the observations of the selected cluster, assigning them the label c, and then the real training starts as usual.

Chapter 4

Experiments

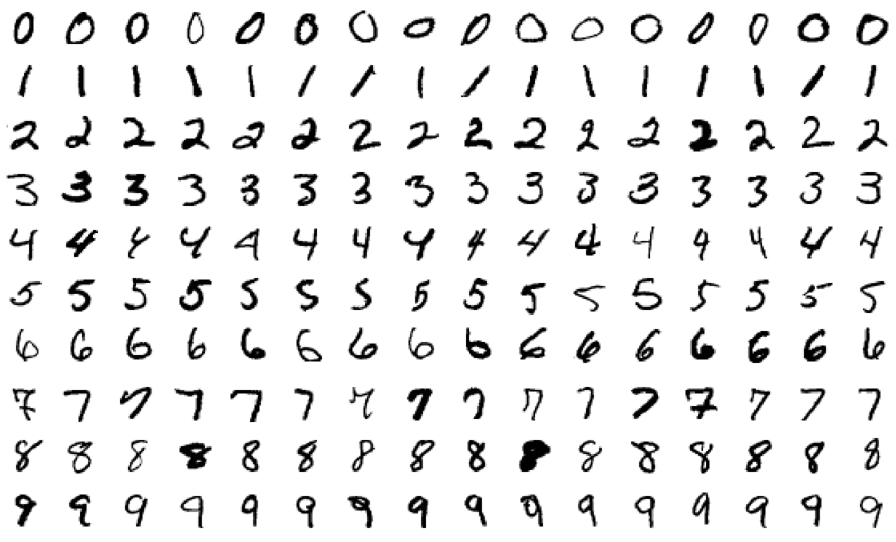
The purpose of the current chapter is, in the first place, to describe in detail the specific tasks and datasets used for the experiments and, in the second place, to show and comment the obtained results.

4.1 Tasks and dataset

In order to test the model depicted in Section 3.2, we considered the following task of the type described in Section 2.4: learning to classify digits from 0 to 9, knowing the sum of n tuples (having cardinality m) of them. With reference to the nomenclature used in Section 2.4, let $f = sum(c_1, ..., c_m)$ be the function that takes in input m digits from 0 to 9 and compute their sums. Note that such function is not injective (e.g., if $m = 2 : 0 + 3 = 3$ as well as $2 + 1 = 3$) and the order of the elements in the tuple is relevant in the interests of the classification task (e.g., if $m = 2 : (0, 2) \neq (0, 2)$ even if $f(0, 2) = f(2, 0) = 2$).

The starting dataset to perform this task is MNIST, that includes handwritten digits and it is commonly used for training and testing in the filed of machine learning. It contains 60'000 training images and 10'000 testing images; each image has 784 pixels (28×28). For the purpose of the experiments, the training set has been split as follows: 54'000 examples for the training set and 6'000 for the validation set. Besides, we removed pixels with very low variance (in particular, where the pixel's variance is smaller than 0.001 times the average variance); the resulting images have 629 pixels.

FIGURE 4.1: Sample images from MNIST dataset.



Starting from the dataset just described, we applied the function f to the labels $c_i \in C$ corresponding to observations $\mathbf{x}_i \in X$ in order to modify the dataset in the form $\{((\mathbf{x}_1, ..., \mathbf{x}_m), f_1), ..., ((\mathbf{x}_{n-m+1}, ..., \mathbf{x}_n), f_{\frac{n}{m}})\}$. In particular, we considered 3 values of m (i.e., number of addends in the sum): 2, 3 and 4. Increasing such a value makes the task harder for two reasons:

- the number of possible abductions increases, with equal sum. For example: if $sum(c_1, c_2) = 2$, then $A = \{(0,2), (1,1), (2,0)\}$ and $|A| = 3$; if $sum(c_1, c_2, c_3) = 2$, then $A = \{(0,0,2), (0,2,0), (2,0,0), (1,1,0), (1,0,1), (0,1,1)\}$ and $|A| = 6$;

- the number of observed cases in the modified dataset in which there is only one possible abduction decreases. In order to explain this observation, let us first consider the case of $m = 2$. The cases in which we have just one possible abduction are when the sum is 0 or 18; in fact, in such cases, we know that the possible addends are respectively only (0,0) and (9,9). Considering $m = 3$, we can make the same argument for sums 0 and 27. The difference lies in the way in which the modified dataset is computed: since we apply the function $f = sum(c_1, ..., c_m)$ to the labels $c_i \in C$, the cases in which we have $m = 3$ zero (or equivalently nine) in succession decreases. Similarly, this number goes on to decrease when we consider $m = 4$.

Basically, we want to evaluate the model making the task more and more difficult, that is decreasing the number of observed certainties (i.e., cases in which there is only one possible abduction) and increasing the number of possible abductions.

These considerations are highlighted in Table 4.1, which shows the number of examples and the corresponding number of possible abductions by varying the value of m. We can observe that the certainties decrease from 1048 when $m = 2$ to 102 when $m = 3$ to only 12 when $m = 4$. Besides, we underline the significative increse of possible abductions for examples.

TABLE 4.1: Number of examples and corresponding number of possible abductions respectively for $m = 2$, $m = 3$ and $m = 4$ for dataset MNIST. Rows in bold highlight the certainties for each task.

#(ex)	#(abd)
1048	**1**
2480	2
3228	3
4580	4
5278	5
6398	6
7332	7
8640	8
9744	9
5272	10

#(ex)	#(abd)
102	**1**
459	3
684	6
1305	10
1788	15
2625	21
2961	28
3927	36
4803	45
5952	55
6585	63
7269	69
7707	73
7833	75

#(ex)	#(abd)
12	**1**
40	4
164	10
296	20
412	35
708	56
1104	84
1468	120
1836	165
2452	220
3164	282
3776	348
4556	415
5288	480
5824	540
6484	592
6336	633
6636	660
3444	670

4.2 Results

In the following, we show and analyse the experiments results on the tree tasks described in Section 4.1. According to the specific task and related difficulties, we performed a grid search by varying the model hyper-parameters, which are in summary:

- split-depth, number of input distributions, number of root nodes, number of sum nodes per inner regions, dropout at input, dropout at sums (from RAT-SPN, see Section 2.2.1);
- abductions threshold and pre-training (see Sections 3.2.3 and 3.2.4).

On the contrary, we fixed the batch size and the number of epochs to 100 and the use of *adam* (adaptive moment estimation) [23] as optimization algorithm for RAT-SPN.

4.2.1 Sum of pairs

Table 4.2 shows the esperiments results in the case of $m = 2$ (i.e., sum of pairs). Because of the training computational cost of RAT-SPNs with split depth striclty greater than 1, we tested only the configurations with split depth equal to 1, exploring almost all the ones proposed by Peharz et al. [37] in the original experiments on mnist dataset (only configurations with dropout equal to 0.25 have been discarded, given th reduced dimension of the considered SPNs). As regards the introduced hyper-parameters (i.e., abduction threshold and pre-training), since the task under consideration presents a sufficiently high number of "certainties" (i.e., observations with just one possible abduction) and a not so significant number of possible abductions (see Table 4.1), we decided not to exploit these additional implemented features in this case (i.e., threshold = 0 and pre-training = false).

TABLE 4.2: Performance for $f = sum(c_1, c_2)$. Batch size = 100, number of epochs = 100, optimization algorithm: *adam*, abductions threshold = 0, pre-training = false. All accuracy values are in percentage and approximated to the second digit after the decimal place. The best configuration is highlighted in bold.

depth	distributions	splits	sums	dropout input	dropout sums	valid acc	test acc
1	10	9	10	1	1	94.87	94.74
1	10	9	10	1	0.75	94.77	94.8
1	10	9	10	1	0.50	95.27	94.84
1	10	9	10	0.75	1	96.07	95.77
1	10	9	10	0.75	0.75	96.18	95.76
1	10	9	10	0.75	0.50	96.53	95.91
1	10	9	10	0.50	1	96.17	95.89
1	10	9	10	0.50	0.75	96.38	95.90
1	10	9	10	0.50	0.50	96.07	95.82
1	15	14	10	1	1	94.90	95.43
1	15	14	10	1	0.75	95.57	95.57
1	15	14	10	1	0.50	95.12	95.30
1	15	14	10	0.75	1	96.57	96.23
1	15	14	10	0.75	0.75	96.48	96.36
1	15	14	10	0.75	0.50	96.72	96.27
1	15	14	10	0.50	1	96.53	96.08
1	15	14	10	0.50	0.75	96.63	96.17
1	15	14	10	0.50	0.50	96.33	95.99
1	20	19	10	1	1	95.60	95.59
1	20	19	10	1	0.75	95.37	95.45
1	20	19	10	1	0.50	95.63	95.22
1	20	19	10	0.75	1	96.88	96.58
1	20	19	10	0.75	0.75	96.60	96.47
1	20	19	10	0.75	0.50	96.57	96.67
1	20	19	10	0.50	1	96.95	96.48
1	20	19	10	0.50	0.75	96.52	96.28
1	20	19	10	0.50	0.50	96.75	96.19
1	25	29	10	1	1	95.90	95.80
1	25	29	10	1	0.75	95.83	95.73
1	25	29	10	1	0.50	95.62	95.90
1	25	29	10	0.75	1	96.93	96.51
1	25	29	10	0.75	0.75	96.72	96.63
1	25	29	10	0.75	0.50	96.87	96.76
1	25	29	10	0.50	1	96.82	96.73
1	25	29	10	0.50	0.75	96.95	96.60
1	25	29	10	0.50	0.50	96.67	96.50
1	33	40	10	1	1	95.97	95.67
1	33	40	10	1	0.75	95.78	95.93
1	33	40	10	1	0.50	95.77	95.71
1	33	40	10	0.75	1	96.77	96.51
1	33	40	10	0.75	0.75	96.77	96.66
1	33	40	10	0.75	0.50	96.90	96.48
1	33	40	10	0.50	1	97.03	96.89
1	33	40	10	0.50	0.75	96.95	96.87
1	**33**	**40**	**10**	**0.50**	**0.50**	**97.10**	**96.88**

First of all, we observe that accuracy values are very similar among all the considered configurations, in contrast with the model showed in Section 3.1. This means that the labeling accuracy (hence the training success) does not depend on the chance, but on a deterministic learning, that always carries out the model to understand (more or less according to the specific configuration and only in a very small part to the chance) the proper "direction" to take. In conclusion, we can infer that, for this specific task, the only observations sorting mechanism (described in Section 3.2.2) turns out to be sufficient in order to achieve great performance.

4.2.2 Sum of triples

Table 4.3 and 4.4 show the esperiments results in the case of $m = 3$ (i.e., sum of triples). Since in the previous experiment (Section 4.2.1) no significant difference was observed increasing the number of input distributions and since we are more interested in understanding the model behaviour than the real performance in terms of accuracy, we explored the ones having less computational cost. In this case, however, given the increased difficulty (see Table 4.1) we introduced both the abduction threshold and the pre-training.

TABLE 4.3: Performance for $f = sum(c_1, c_2, c_3)$. Batch size = 100, number of epochs = 100, optimization algorithm: *adam*, **pre-training = false**. All accuracy values are in percentage and approximated to the second digit after the decimal place. The best configuration is highlighted in bold.

depth	distributions	splits	sums	dropout input	dropout sums	threshold	valid acc	test acc
1	10	9	10	1	1	0	94.10	94.19
1	10	9	10	1	1	0.01	98.05	94.22
1	10	9	10	1	1	0.05	94.47	94.30
1	10	9	10	1	1	0.10	94.35	94.35
1	10	9	10	0.75	0.75	0	96.78	96.46
1	10	9	10	0.75	0.75	0.05	96.27	95.89
1	10	9	10	0.75	0.75	0.01	96.63	96.21
1	10	9	10	0.75	0.75	0.10	95.82	95.45
1	15	14	10	1	1	0	94.22	94.52
1	15	14	10	1	1	0.01	95.23	94.63
1	15	14	10	1	1	0.05	94.67	94.66
1	15	14	10	1	1	0.10	98.14	94.71
1	15	14	10	0.75	0.75	0	96.85	96.14
1	15	14	10	0.75	0.75	0.01	96.00	95.57
1	15	14	10	0.75	0.75	0.05	96.63	96.34
1	15	14	10	0.75	0.75	0.10	95.52	95.49
1	**20**	**19**	**10**	**1**	**1**	**0**	**98.52**	**94.78**
1	20	19	10	1	1	0.01	94.73	94.59
1	20	19	10	1	1	0.05	94.62	94.68
1	20	19	10	1	1	0.10	95.13	94.66
1	20	19	10	0.75	0.75	0	96.27	96.10
1	20	19	10	0.75	0.75	0.01	96.87	96.29
1	20	19	10	0.75	0.75	0.05	96.63	96.35
1	20	19	10	0.75	0.75	0.10	96.83	96.58

TABLE 4.4: Performance for $f = sum(c_1, c_2, c_3)$. Batch size = 100, number of epochs = 100, optimization algorithm: *adam*, **pre-training = true**. All accuracy values are in percentage and approximated to the second digit after the decimal place. The best configuration is highlighted in bold.

depth	distributions	splits	sums	dropout input	dropout sums	threshold	valid acc	test acc
1	10	9	10	1	1	0	94.05	94.23
1	10	9	10	1	1	0.01	94.65	94.24
1	10	9	10	1	1	0.05	94.17	94.37
1	10	9	10	1	1	0.10	94.23	94.02
1	10	9	10	0.75	0.75	0	96.50	95.87
1	10	9	10	0.75	0.75	0.05	96.32	95.96
1	10	9	10	0.75	0.75	0.01	96.38	96.08
1	10	9	10	0.75	0.75	0.10	96.08	95.86
1	15	14	10	1	1	0	94.80	94.79
1	15	14	10	1	1	0.01	95.00	94.96
1	15	14	10	1	1	0.05	95.02	94.92
1	15	14	10	1	1	0.10	94.43	94.08
1	15	14	10	0.75	0.75	0	96.80	96.25
1	15	14	10	0.75	0.75	0.01	96.37	95.41
1	15	14	10	0.75	0.75	0.05	96.57	96.13
1	15	14	10	0.75	0.75	0.10	96.57	96.40
1	20	19	10	1	1	0	95.13	94.59
1	20	19	10	1	1	0.01	94.88	95.26
1	20	19	10	1	1	0.05	95.02	95.14
1	20	19	10	1	1	0.10	95.13	94.87
1	**20**	**19**	**10**	**0.75**	**0.75**	**0**	**96.88**	**96.15**
1	20	19	10	0.75	0.75	0.01	96.82	96.38
1	20	19	10	0.75	0.75	0.05	96.70	96.47
1	20	19	10	0.75	0.75	0.10	96.80	96.45

As with the case of $m = 2$, we can notice the same constant behaviour (in terms of accuracy) among the tested configurations. Such consideration confirms that, also in the case of a task that is more difficult than the previous one, the learning process is deterministic. As regards the abduction threshold, its use doesn't seem to bring advantages to the model, from a purely perfomance point of view; the best observed configurations (highlighted in bold) actually work without threshold (i.e., equal to 0). Analogous observation can be done for the pre-training: even if the best configuration of the pre-trained model performs better on the test set with respect to the best of the not pre-trained one, the comparison between correspondent configurations points out a substancial similarity. Nevertheless, it is interesting to see if some differences come out in the general model behaviour by varying the threshold value and enabling the pre-training. To this end, Figures 4.2 show *(i)* the number of considered examples, *(ii)* the examples labeling accuracy (to perform the successive training) and *(iii)* the model accuracy on validation set per epoch, by varying the abduction threshold and enabling the pre-training. We can make the following observations:

- the greater the threshold, the more the number of considered examples increases slowly; of course, when the threshold is equal to 0, all examples are taken into account from the first epoch to the last one. Besides, when the pre-training is enabled, much more examples are considered from the beginning and the number of considered examples increases faster, since the initialization of the model is no longer randomic;

- in the case of threshold uqual to 0.05 and 0.1 without pre-training, the examples labeling accuracy starts from the top value (since it considers all and only the examples with just one possible abduction), but then it drops to very small values, because the low number of considered examples was not sufficient to learn enough. Nevertheless, in both cases, the model is able to improve again its examples labeling accuracy and to easily overcome the model without threshold (i.e., equal to 0) after some epochs. When pre-training is enabled, the model directly starts from low values of labeling accuracy, since it considers more examples from the beginning, and it again learns faster than the model wituout threshold. The motivation behind such improvement definitely lies in the fact the such threshold values allow the model to label (and consequenlty consider) examples with a certain degree of certainty. On the contrary, when the threshold is set to 0.01, no advantage is observed, actually the no threshold model performs better;

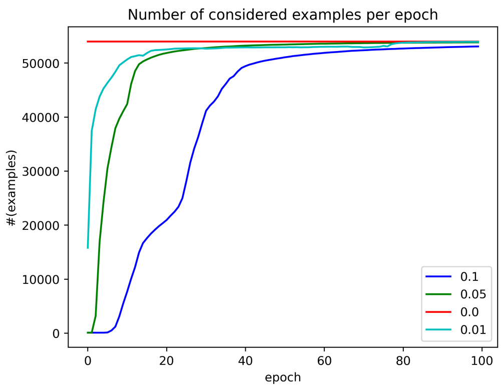
- the model accuracy behaviour on validation set is very similar to the examples labeling accuracy one (except for the drop in the case of thresholds

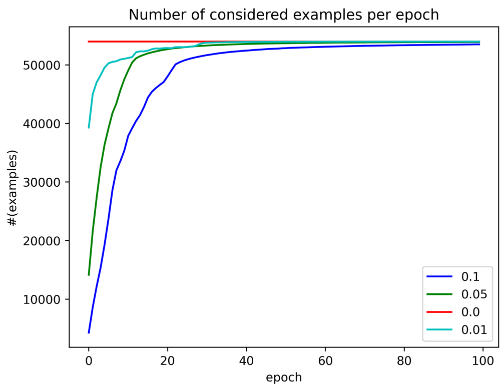
0.05 and 0.1 and disabled pre-training). The motivation of such observation lies in the way in which the most probable abduction is selected (see 3.2.1): since this choice depends on the sub-symbolic model, the more it is learning, the more its accuracy increases both on the validation set and in selecting the abduction for the training examples.

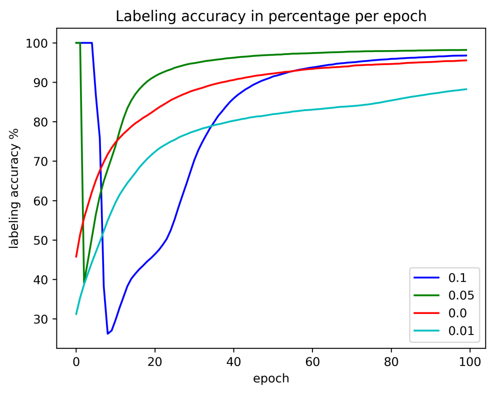
In the light of the above considerations, we can conclude that, although all configurations achieve good performance in terms of accuracy, the use both of a sufficiently high threshold and of the pre-training allows the model to converge faster, even if starting from lower accuracy values.

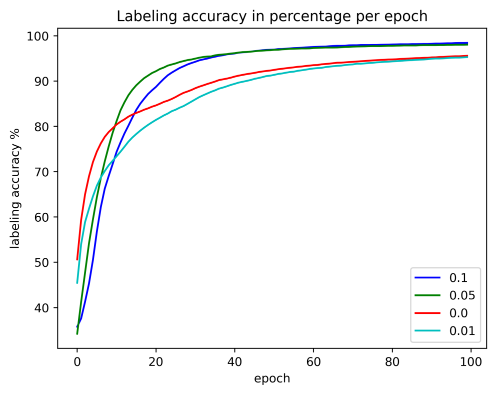
FIGURE 4.2: Behaviour comparison between the not pre-trained model and the pre-trained one for $f = sum(c_1, c_2, c_3)$, taking into account: number of considered examples, examples labeling accuracy and model accuracy on validation set per epoch by varying the abduction threshold. The average values have been computed among different configurations but with the same threshold.

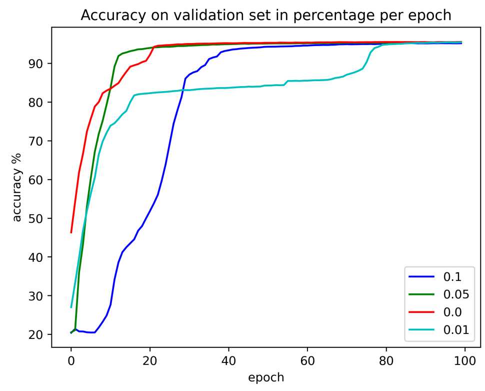
Pre-training = false **Pre-training = true**

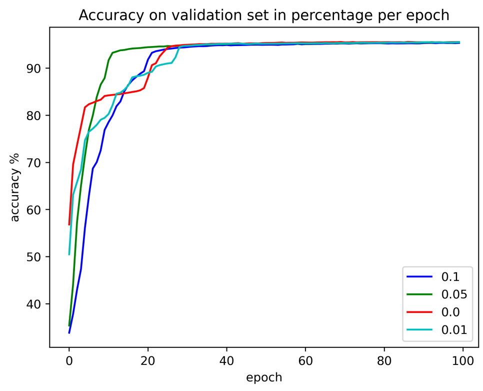












4.2.3 Sum of quadruples

Table 4.3 and 4.4 show the esperiments results in the case of $m = 4$ (i.e., sum of quadruples). We considered the same configurations tested for $m = 3$, just slighlty increasing the threshold values. In addition, given the task extreme difficulty (see 4.1), we also explored RAT-SPNs with split depth equal to 2 in the case of enabled pre-training.

TABLE 4.6: Performance for $f = sum(c_1, c_2, c_3, c_4)$. Batch size = 100, number of epochs = 100, optimization algorithm: *adam*, **pre-training** = **false**. All accuracy values are in percentage and approximated to the second digit after the decimal place.

depth	distributions	splits	sums	dropout input	dropout sums	threshold	valid acc	test acc
1	10	9	10	1	1	0	25.57	26.13
1	10	9	10	1	1	0.05	31.30	32.00
1	10	9	10	1	1	0.08	30.85	32.41
1	10	9	10	1	1	0.11	33.02	34.30
1	10	9	10	0.75	0.75	0	95.10	95.16
1	10	9	10	0.75	0.75	0.05	93.48	93.56
1	10	9	10	0.75	0.75	0.08	34.95	35.86
1	10	9	10	0.75	0.75	0.11	22.92	23.67
1	15	14	10	1	1	0	44.37	44.59
1	15	14	10	1	1	0.05	25.51	26.02
1	15	14	10	1	1	0.08	25.75	26.15
1	15	14	10	1	1	0.11	29.35	30.11
1	15	14	10	0.75	0.75	0	22.92	24.03
1	15	14	10	0.75	0.75	0.05	25.42	25.40
1	15	14	10	0.75	0.75	0.08	30.42	31.32
1	15	14	10	0.75	0.75	0.11	21.32	21.94
1	20	19	10	1	1	0	36.21	36.78
1	20	19	10	1	1	0.01	28.15	28.99
1	20	19	10	1	1	0.05	35.68	36.73
1	20	19	10	1	1	0.10	41.23	43.58
1	20	19	10	0.75	0.75	0	26.18	26.89
1	20	19	10	0.75	0.75	0.01	93.12	93.26
1	20	19	10	0.75	0.75	0.05	23.88	24.00
1	20	19	10	0.75	0.75	0.10	34.11	35.79

TABLE 4.7: Performance for $f = sum(c_1, c_2, c_3, c_4)$. Batch size = 100, number of epochs = 100, optimization algorithm: *adam*, **pre-training = true**. All accuracy values are in percentage and approximated to the second digit after the decimal place.

depth	distributions	splits	sums	dropout input	dropout sums	threshold	valid acc	test acc
1	10	9	10	1	1	0	43.92	44.67
1	10	9	10	1	1	0.05	30.05	31.48
1	10	9	10	1	1	0.08	28.80	28.62
1	10	9	10	1	1	0.11	26.63	26.67
1	10	9	10	0.75	0.75	0	96.55	96.10
1	10	9	10	0.75	0.75	0.05	96.23	95.79
1	10	9	10	0.75	0.75	0.08	34.78	34.94
1	10	9	10	0.75	0.75	0.11	32.07	31.75
1	15	14	10	1	1	0	94.83	94.43
1	15	14	10	1	1	0.05	24.83	25.73
1	15	14	10	1	1	0.08	34.62	35.90
1	15	14	10	1	1	0.11	35.38	35.71
1	15	14	10	0.75	0.75	0	96.05	96.07
1	15	14	10	0.75	0.75	0.05	96.35	96.17
1	15	14	10	0.75	0.75	0.08	32.92	33.92
1	15	14	10	0.75	0.75	0.11	27.30	28.35
1	20	19	10	1	1	0	93.00	94.12
1	20	19	10	1	1	0.01	31.65	32.78
1	20	19	10	1	1	0.05	25.89	26.37
1	20	19	10	1	1	0.10	44.20	45.99
1	20	19	10	0.75	0.75	0	31.54	33.42
1	20	19	10	0.75	0.75	0.01	94.11	95.56
1	20	19	10	0.75	0.75	0.05	28.23	28.35
1	20	19	10	0.75	0.75	0.10	33.51	33.93
2	10	8	10	1	1	0	19.92	20.35
2	10	8	10	1	1	0.05	24.18	25.09
2	10	8	10	1	1	0.08	32.20	33.14
2	10	8	10	1	1	0.11	37.87	37.39
2	10	8	10	0.75	0.75	0	97.60	97.15
2	10	8	10	0.75	0.75	0.05	33.80	33.48
2	10	8	10	0.75	0.75	0.08	36.53	36.09
2	10	8	10	0.75	0.75	0.11	37.98	38.01
2	15	12	15	1	1	0	20.73	21.00
2	15	12	15	1	1	0.05	25.07	25.58
2	15	12	15	1	1	0.08	33.50	34.02
2	15	12	15	1	1	0.11	39.42	40.47
2	15	12	15	0.75	0.75	0	26.00	26.81
2	15	12	15	0.75	0.75	0.08	37.48	38.26
2	15	12	15	0.75	0.75	0.05	27.82	28.84
2	15	12	15	0.75	0.75	0.11	41.52	42.50

Unfortunately, the behaviour shown by the above tables is similar to the one highlighted in Table 3.1: performance varies greatly from one configuration to another, indicating that the lerning is no longer deterministic, but prone to chance. Nevertheless, with respect to Table 3.1, we note a significant difference: there are no "intermediate" accuracy values, basically the model converges or doesn't learn at all. This could mean that, if the model "finds the correct direction", it is always able to converge and achieve good performance.

Given the non-deterministic nature of learning, in this case it is not interesting to evaluate the model behaviour as in Figures 4.2. Nevertheless, we want to understand if the pre-training can still bring some advantages. For this purpose, Table 4.8 compares corresponding configurations (with depth equal to 1) of the pre-trained model and the not pre-trained one, taking into account *(i)* the number of considered examples and *(ii)* the ratios of examples labeling accuracy of digit from 0 to 2. The analysis is limited to the first epoch, because it is the most influenced by the pre-training, and to such digits, because we observed that the pre-training perform only on *zero* examples (for this specific task). Results highlighted by such analysis are evident: the examples labeling accuracy is always higher when the pre-training is enabled, except for two not relevant cases in which it is slighlty lower, but in the face of a greater number of considered examples.

We can conclude that the pre-training still correctly works, but it is no longer sufficient, in conjunction with the abduction threshold, to solve such a difficult task.

TABLE 4.8: Comparison between corresponding configurations (with split depth = 1) of the pre-trained model and the not pre-trained one, with reference to Tables 4.6 and 4.7. #(examples) refers to the number of considered examples during the current training epoch, while R_i refers to the ratio of examples labeling accuracy of the *i-th* digit. All the reported values refer to the **first** training epoch. Greater R_i values between corresponding configurations are highlighted in bold.

Pre-training = true				**Pre-training = false**			
#(examples)	R_0	R_1	R_2	#(examples)	R_0	R_1	R_2
54000	**0.90**	**0.89**	**0.06**	54000	0.64	0.02	0.05
32	0.95	0.33	-	12	**1**	-	-
12	1	-	-	12	1	-	-
12	1	-	-	12	1	-	-
54000	**0.93**	**0.86**	**0.14**	54000	0.83	0.05	0.08
36	1	**1**	-	12	1	-	-
12	1	-	-	12	1	-	-
12	1	-	-	12	1	-	-
54000	**0.90**	**0.81**	**0.13**	54000	0.89	0.14	0.08
52	1	0.33	0.67	12	1	-	-
12	1	-	-	12	1	-	-
12	1	-	-	12	1	-	-
54000	**0.89**	**0.94**	**0.31**	54000	0.88	0.85	0.06
12	1	-	-	12	1	-	-
12	1	-	-	12	1	-	-
12	1	-	-	12	1	-	-
54000	**0.91**	**0.88**	**0.12**	54000	0.60	0.06	0.02
40	0.97	0.41	-	12	**1**	-	-
12	1	-	-	12	1	-	-
12	1	-	-	12	1	-	-
54000	**0.92**	**0.85**	**0.12**	54000	0.79	0.25	0.04
12	1	-	-	12	1	-	-
12	1	-	-	12	1	-	-
12	1	-	-	12	1	-	-

Chapter 5

Conclusions and Future Work

Neural-symbolic computing aims at integrating two most fundamental cognitive abilities: the ability to learn from the environment, and the ability to reason from what has been learned. Neural-symbolic computing has been an active topic of research for many years, reconciling the advantages of robust learning in neural networks and reasoning and interpretability of symbolic representation.

In this thesis, we explored the integration between *Sum-Product networks* (specifically *Random-And-Tensorized SPNs*) and *abductive reasoning*, applied to a more difficult classification task, in which the known information about training examples is not directly their associated class but an aggregate information deriving from it. We stressed the model robustness by increasing the task difficulty, that is increasing the number of possible abductions and decreasing the number of certainties (i.e., observations for which just one possible label can be abduced). Experiments results showed that both the implemented solutions and *RAT-SPNs* have proved to be adequate (but not always sufficient) in dealing with a task of this kind.

In the following, we outline some possible future works:

- to explore the model behaviour with SPNs having higher split depths, in order to understand if the use of more complex models might bring some advantages;
- to deal with tasks in which the number of possible abductions for some observations is so high that it requires a pruning mechanism. Such cases would need to increase the interaction between symbolic and sub-symbolic modules, specifically during the training phase;

- to implement an *adaptive* abduction threshold, that varies according to the learning model degree within both the same training epoch and different ones;
- to use a loss function that penalizes errors in a manner directly proportional to the probability of the corresponding abduction;
- to ensure in some way a consistency in examples labeling by the model.

Bibliography

[1] Nikhil Buduma and Nicholas Locascio. *Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms*. 1st. O'Reilly Media, Inc., 2017. ISBN: 1491925612.

[2] Arthur Choi, Guy Van den Broeck, and Adnan Darwiche. "Tractable learning for structured probability spaces: A case study in learning preference distributions". In: *Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI)*. Vol. 2015. IJCAI-INT JOINT CONF ARTIF INTELL. 2015, pp. 2861–2868.

[3] YooJung Choi, Antonio Vergari, and Guy Van den Broeck. "Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Models". In: (2020). URL: `http://starai.cs.ucla.edu/papers/ProbCirc20.pdf`.

[4] A. Darwiche. "A logical approach to factoring belief networks". In: *Proceedings of KR*. 2002, pp. 409–420.

[5] Adnan Darwiche. "A Differential Approach to Inference in Bayesian Networks". In: *CoRR* abs/1301.3847 (2013). arXiv: `1301.3847`. URL: `http://arxiv.org/abs/1301.3847`.

[6] Marc Denecker. "A terminological interpretation of (abductive) logic programming". In: *Logic Programming and Nonmonotonic Reasoning*. Ed. by V. Wiktor Marek, Anil Nerode, and M. Truszczyński. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 15–28. ISBN: 978-3-540-49282-5.

[7] Marc Denecker and Danny De Schreye. "Representing Incomplete Knowledge in Abductive Logic Programming". In: *Journal of Logic and Computation* 5 (Dec. 2001). DOI: `10.1093/logcom/5.5.553`.

[8] Marc Denecker, Lode Missiaen, and Maurice Bruynooghe. "Temporal reasoning with Abductive Event Calculus". In: (Dec. 2001).

[9] Mattia Desana and Christoph Schnörr. "Expectation Maximization for Sum-Product Networks as Exponential Family Mixture Models". In: *CoRR* abs/1604.07243 (2016). arXiv: `1604.07243`. URL: `http://arxiv.org/abs/1604.07243`.

[10] P. Dung. "Negations as Hypotheses: An Abductive Foundation for Logic Programming". In: *ICLP*. 1991.

[11] Kave Eshghi. "Abductive Planning with Event Calculus." In: Jan. 1988, pp. 562–579.

[12] Kave Eshghi and Robert Kowalski. "Abduction Compared with Negation by Failure." In: Jan. 1989, pp. 234–254.

[13] Artur S. d'Avila Garcez and Dov M. Gabbay. "Fibring Neural Networks". In: *Proceedings of the 19th National Conference on Artifical Intelligence*. AAAI'04. San Jose, California: AAAI Press, 2004, 342–347. ISBN: 0262511835.

[14] Robert Gens and Domingos Pedro. "Learning the Structure of Sum-Product Networks". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 2013, pp. 873–880. URL: `http://proceedings.mlr.press/v28/gens13.html`.

[15] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. "A Fast Learning Algorithm for Deep Belief Nets". In: *Neural Comput.* 18.7 (July 2006), 1527–1554. ISSN: 0899-7667. DOI: `10.1162/neco.2006.18.7.1527`. URL: `https://doi.org/10.1162/neco.2006.18.7.1527`.

[16] A. Kakas and P. Mancarella. "On the relation between truth maintenance and abduction". In: 1990.

[17] A. Kakas, A. Michael, and Costas Mourlas. "ACLP: A case for Non-Monotonic Reasoning". In: 1998.

[18] A. C. KAKAS, R. A. KOWALSKI, and F. TONI. "Abductive Logic Programming". In: *Journal of Logic and Computation* 2.6 (Dec. 1992), pp. 719–770. ISSN: 0955-792X. DOI: `10.1093/logcom/2.6.719`. eprint: `https://academic.oup.com/logcom/article-pdf/2/6/719/2776092/2-6-719.pdf`. URL: `https://doi.org/10.1093/logcom/2.6.719`.

[19] A. C. Kakas, R. A. Kowalski, and F. Toni. *The Role of Abduction in Logic Programming.* 1998.

[20] A.C. Kakas, A. Michael, and C. Mourlas. "ACLP: Abductive Constraint Logic Programming". In: *The Journal of Logic Programming* 44.1 (2000), pp. 129 –177. ISSN: 0743-1066. DOI: `https://doi.org/10.1016/S0743-1066(99)00075-8`. URL: `http://www.sciencedirect.com/science/article/pii/S0743106699000758`.

[21] Antonis Kakas and Paolo Mancarella. "Database Updates through Abduction." In: Jan. 1990, pp. 650–661.

[22] Pasha Khosravi et al. "On tractable computation of expected predictions". In: *Advances in Neural Information Processing Systems*. 2019, pp. 11169–11180.

[23] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2017. arXiv: `1412.6980 [cs.LG]`.

[24] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques.* Adaptive computation and machine learning. MIT Press, 2009. ISBN: 9780262013192. URL: `https://books.google.co.in/books?id=7dzpHCHzNQ4C`.

[25] Robert Kowalski. *Computational Logic and Human Thinking: How to Be Artificially Intelligent.* Cambridge University Press, 2011. DOI: `10.1017/CBO9780511984747`.

[26] Robert Kowalski and Marek Sergot. "A Logic-Based Calculus of Events". In: *Foundations of Knowledge Base Management: Contributions from Logic, Databases, and Artificial Intelligence Applications.* Ed. by Joachim W.

Schmidt and Constantino Thanos. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, pp. 23–55. ISBN: 978-3-642-83397-7. DOI: 10.1007/978-3-642-83397-7_2. URL: https://doi.org/10.1007/978-3-642-83397-7_2.

[27] Hannes Leitgeb. "Interpreted Dynamical Systems and Qualitative Laws: from Neural Networks to Evolutionary Systems". In: *Synthese* 146 (Aug. 2005), pp. 189–202. DOI: 10.1007/s11229-005-9086-5.

[28] Yitao Liang and Guy Van den Broeck. "Towards compact interpretable models: Shrinking of learned probabilistic sentential decision diagrams". In: *IJCAI 2017 Workshop on Explainable Artificial Intelligence (XAI)*. 2017.

[29] S. Lloyd. "Least squares quantization in PCM". In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137. DOI: 10.1109/TIT.1982.1056489.

[30] J. MacQueen. "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*. Berkeley, Calif.: University of California Press, 1967, pp. 281–297. URL: https://projecteuclid.org/euclid.bsmsp/1200512992.

[31] Robin Manhaeve et al. "DeepProbLog: Neural Probabilistic Logic Programming". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Montréal, Canada: Curran Associates Inc., 2018, 3753–3763.

[32] LODE MISSIAEN, MAURICE BRUYNOOGHE, and MARC DENECKER. "CHICA, An Abductive Planning System Based on Event Calculus". In: *Journal of Logic and Computation* 5.5 (Oct. 1995), pp. 579–602. ISSN: 0955-792X. DOI: 10.1093/logcom/5.5.579. eprint: https://academic.oup.com/logcom/article-pdf/5/5/579/3081061/5-5-579.pdf. URL: https://doi.org/10.1093/logcom/5.5.579.

[33] Alejandro Molina et al. “Sum-Product Networks for Hybrid Domains”. In: *CoRR* abs/1710.03297 (2017). arXiv: 1710.03297. URL: http://arxiv.org/abs/1710.03297.

[34] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN: 0262018020.

[35] Robert Peharz, Bernhard C. Geiger, and Franz Pernkopf. “Greedy Part-Wise Learning of Sum-Product Networks”. In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Hendrik Blockeel et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 612–627. ISBN: 978-3-642-40991-2.

[36] Robert Peharz et al. “On the Latent Variable Interpretation in Sum-Product Networks”. In: *CoRR* abs/1601.06180 (2016). arXiv: 1601.06180. URL: http://arxiv.org/abs/1601.06180.

[37] Robert Peharz et al. “Probabilistic Deep Learning using Random Sum-Product Networks”. In: *CoRR* abs/1806.01910 (2018). arXiv: 1806.01910. URL: http://arxiv.org/abs/1806.01910.

[38] Charles S. Peirce. *The Collected Papers of Charles Sanders Peirce, Vol. II: Elements of Logic*. English. Ed. by Charles Hartshorne and Paul Weiss. From the Commens Bibliography | http://www.commens.org/bibliography/anthology_volume/peirce-charles-s-1932-collected-papers-charles-sanders-peirce-vol-ii. Cambridge: Harvard University Press, 1932.

[39] Leo de Penning. “An Integrated Neural Symbolic Cognitive Agent Architecture for Training and Assessment in Simulators”. In: Jan. 2010.

[40] Gadi Pinkas. “Symmetric Neural Networks and Propositional Logic Satisfiability”. In: *Neural Computation - NECO* 3 (June 1991), pp. 282–291. DOI: 10.1162/neco.1991.3.2.282.

[41] David Poole. “A logical framework for default reasoning”. In: *Artificial Intelligence* 36.1 (1988), pp. 27 –47. ISSN: 0004-3702. DOI: https://

doi.org/10.1016/0004-3702(88)90077-X. URL: http://www.sciencedirect.com/science/article/pii/000437028890077X.

[42] Hoifung Poon and Pedro M. Domingos. "Sum-Product Networks: A New Deep Architecture". In: *CoRR* abs/1202.3732 (2012). arXiv: 1202.3732. URL: http://arxiv.org/abs/1202.3732.

[43] Iris Rooij. "The Tractable Cognition Thesis". In: *Cognitive science* 32 (Sept. 2008), pp. 939–84. DOI: 10.1080/03640210801897856.

[44] Amirmohammad Rooshenas and Daniel Lowd. "Learning Sum-Product Networks with Direct and Indirect Variable Interactions". In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 1. Bejing, China: PMLR, 2014, pp. 710–718. URL: http://proceedings.mlr.press/v32/rooshenas14.html.

[45] Murray Shanahan. "An abductive event calculus planner". In: *The Journal of Logic Programming* 44.1 (2000), pp. 207 –240. ISSN: 0743-1066. DOI: https://doi.org/10.1016/S0743-1066(99)00077-1. URL: http://www.sciencedirect.com/science/article/pii/S0743106699000771.

[46] Murray Shanahan. "Prediction is Deduction but Explanation is Abduction." In: Jan. 1989, pp. 1055–1060.

[47] Hava T. Siegelmann. *Neural Networks and Analog Computation: Beyond the Turing Limit*. USA: Birkhauser Boston Inc., 1999. ISBN: 0817639497.

[48] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting." In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958. URL: http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf.

[49] Ron Sun. "Theoretical Status of Computational Cognitive Modeling". In: *Cogn. Syst. Res.* 10.2 (June 2009), 124–140. ISSN: 1389-0417. DOI: 10.1016/j.cogsys.2008.07.002. URL: https://doi.org/10.1016/j.cogsys.2008.07.002.

[50] Leslie G. Valiant. *Knowledge Infusion: In Pursuit of Robustness in Artificial Intelligence.* 2008.

[51] Leslie G. Valiant. "Three Problems in Computer Science". In: *J. ACM* 50.1 (Jan. 2003), 96–99. ISSN: 0004-5411. DOI: `10.1145/602382.602410`. URL: `https://doi.org/10.1145/602382.602410`.

[52] Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. "Simplifying, Regularizing and Strengthening Sum-Product Network Structure Learning". In: *Machine Learning and Knowledge Discovery in Databases.* Ed. by Annalisa Appice et al. Cham: Springer International Publishing, 2015, pp. 343–358. ISBN: 978-3-319-23525-7.

[53] Han Zhao, Mazen Melibari, and Pascal Poupart. "On the Relationship between Sum-Product Networks and Bayesian Networks". In: *CoRR* abs/1501.01239 (2015). arXiv: `1501.01239`. URL: `http://arxiv.org/abs/1501.01239`.