Evaluating Relaxations of Logic for Neural Networks: A Comprehensive Study

Anonymous

Abstract

Symbolic knowledge can provide crucial inductive bias for training neural models, especially in low data regimes. A successful strategy for incorporating such knowledge involves relaxing logical statements into sub-differentiable losses for optimization. In this paper, we study the question of how best to relax logical expressions that represent labeled examples and knowledge about a problem; we focus on sub-differentiable t-norm relaxations of logic. We present theoretical and empirical criteria for characterizing which relaxation would perform best in various scenarios. In our theoretical study driven by the goal of preserving tautologies, the Łukasiewicz t-norm performs best. However, in our empirical analysis on the text chunking and digit recognition tasks, the product t-norm achieves best predictive performance. We analyze this apparent discrepancy, and conclude with a list of best practices for defining loss functions via logic.

1 Introduction

Neural networks are remarkably effective across many domains and tasks; but their usefulness is limited by their data hungriness. A promising direction towards alleviating this concern involves augmenting learning with rules written in first-order logic [e.g., Rocktäschel *et al.*, 2015; Li *et al.*, 2019, inter alia]. To make rules amenable with gradient-based learning, this approach calls for relaxing the logical operators to define sub-differentiable loss terms. A systematic method to perform this relaxation uses a well studied family of binary operators, namely *triangular norms* or *t-norms* [Klement *et al.*, 2013]. Different t-norms define different $[0, 1]$-valued interpretations of the boolean operators.

There are infinitely many such t-norm logics, but the most commonly used are: Product [e.g., Rocktäschel *et al.*, 2015; Li *et al.*, 2019; Asai and Hajishirzi, 2020], Gödel [Minervini *et al.*, 2017, e.g.,] and Łukasiewicz [Bach *et al.*, 2017, e.g.,]. While the usefulness of such relaxations is well established, the questions of how they compare against each other, and even how such a comparison should be defined remain open.

This paper is a first step towards answering these important questions by analyzing the three t-norm relaxations and a variant. To do so, we define the criteria for quantifying the goodness of a t-norm based relaxation. On the theoretical side, we rank logic relaxations by their *consistency*, i.e., their ability to preserve the truth of tautologies. On the empirical side, we use a principled approach to construct loss functions using t-norms that subsumes traditional loss functions like cross entropy, and study how well different relaxations compare with standard gradient-based learning. We report the results of experiments on two tasks: jointly recognizing digits and predicting the results of arithmetic operators, and text chunking. Both the theoretical and empirical criteria concur in the recommendation that a variant of the Product t-norm ($\mathcal{R}$-Product) is most suitable for introducing logical rules into neural networks.

In summary, the main contributions of this work are:

- We define theoretical and empirical properties that any relaxation of logic should have to be useful for learning.
- We define the consistency of relaxations to rank them in terms of their ability to preserve tautologies.
- We present empirical comparisons of logic relaxations on two tasks where labeled examples and declaratively stated knowledge together inform neural models.

2 Problem Statement and Notation

Several recent efforts have shown the usefulness of declarative knowledge to guide neural network learning towards improving model quality. While different approaches exist for incorporating such rules into neural models [e.g., Xu *et al.*, 2018], a prominent strategy involves relaxing logic to the real regime using the well-studied t-norm relaxations.

Triangular norms. Triangular norms (t-norms) arose in the context of probabilistic metric spaces [Menger, 1942; Schweizer *et al.*, 1960], and for our purposes, represent a relaxation of the Boolean conjunction which agrees with the definition of conjunctions for $\{0, 1\}$-inputs. Given such a relaxation and a relaxation of $\neg X$ as $1 - x$, we have two axiomatic approaches for defining implications. The first (called $\mathcal{S}$-logic) treats implications as disjunctions (i.e., $X \rightarrow Y = \neg X \vee Y$), while the second (called R-logics) defines implications axiomatically. We refer the reader to Klement *et al.* [2013] for a detailed treatment of t-norms.

Table 1 shows the set of t-norms we consider in this work, which have been used in recent literature to inject knowl-

edge into neural networks. However, despite their increasing prevalence, there is no consensus on which t-norm to employ. A survey of recent papers reveals the use of $\mathcal{S}$-Product [Rocktäschel *et al.*, 2015], $\mathcal{R}$-Product [e.g., Li *et al.*, 2019; Asai and Hajishirzi, 2020], Łukasiewicz [Bach *et al.*, 2017], Gödel [Minervini *et al.*, 2017] and even a mixture of the Gödel and Product [Li *et al.*, 2020] t-norms.

How do these relaxations of logic compare against each other? In this work, we answer this question from both theoretical and empirical perspectives.

Learning from logic: The setup. To compare the different relaxations of logic on an even footing, let us first see a general recipe for converting logic rules to loss functions for a given relaxation[1]. We will use the task of recognizing handwritten digits as a running example.

Given a task of labeling examples, we can represent the fact that the label for an example x is y as a predicate, say $\texttt{Label}(x, y)$. In our running example, we could define a predicate $\texttt{Digit}(x, y)$ to denote that an image x represents the digit y. We could also define a predicate $\texttt{Sum}(x_1, x_2, y)$ to denote the fact that the digits in two images x_1 and x_2 add up to the digit y (mod10). From this perspective, we can treat classifiers as predicting probabilities that the predicates hold.

To train such classifiers, we typically have a training set D of labeled examples (x, y). In our notation, each such example can be represented as the predicate $\texttt{Label}(x, y)$ and the training set is a conjunction of such predicates

$$\bigwedge_{(x,y)\in D} \texttt{Label}(x, y) \tag{1}$$

Sometimes, instead of a labeled dataset, we may have a constraint written in logic. In our running example, from the definition of the `Digit` and `Sum` predicates, we know that

$$\forall x_1, x_2, \bigwedge_{y_1,y_2} \texttt{Digit}\,(x_1, y_1) \wedge \texttt{Digit}\,(x_2, y_2) \rightarrow \texttt{Sum}\,(x_1, x_2, (y_1 + y_2) \bmod 10) \tag{2}$$

Note that such a constraint need not depend on labeled examples, and should hold irrespective of what labels the examples should be assigned. In general, given a large unlabeled set of examples denoted by $U = x$, we can write constraints

$$\bigwedge_{x\in U} \texttt{C}\,(x) \tag{3}$$

These constraints may be composite formulas constructed with predicates as shown in our running example above.

From this perspective, we can envision the goal of learning as that of ensuring that the formulas representing labeled examples (equation 1) and constraints (equation 3) hold. Since we are treating classifiers as predicting probabilities that the atomic predicates hold, we can equivalently state the learning problem as that of finding model parameters which maximize the value of a *relaxation* of the conjunction of the formulas representing the data and constraints. In other words, we can use logic to define loss functions.

[1]The setup described here is implicitly present in Rocktäschel *et al.* [2015]; Li *et al.* [2019, and others].

	$\mathcal{S}$-Gödel	$\mathcal{R}$-Product	Łukasiewicz
$\wedge$	$\min(x, y)$	$x \cdot y$	$\max(0, x + y - 1)$
$\neg$	$1 - x$	$1 - x$	$1 - x$
$\vee$	$\max(x, y)$	$x + y - x \cdot y$	$\min(1, x + y)$
$\rightarrow$	$\max(1 - x, y)$	$\begin{cases} 1 & \text{if } x \leq y \\ \frac{y}{x} & \text{otherwise} \end{cases}$	$\min(1, 1 - x + y)$

Table 1: T-norm relaxations studied in this work. Here, the letters x and y denote the relaxed truth values of the arguments of the formulas. In the implication definitions, x and y denote the antecedent and the consequent respectively. The table does not show $\mathcal{S}$-Products: it agrees with $\mathcal{R}$-Products for all the connectives except the implication, which is defined as $1 - x + x \cdot y$.

In this declarative learning setting, we have the choice of using *any* models (e.g., convolutional networks) for our predicates, and *any* relaxation of logic. Indeed, if we only have labeled examples, and we use one of the Product relaxations, we recover the widely used cross entropy loss.

Notation. We use upper case letters (e.g., `P`, `Digit`) to represent booleans, and lower cased letters (e.g., p, $digit$) to represent their relaxations. In some places, for clarity, we use square brackets to denote the relaxation of a Boolean formula `A` (i.e., `[A]`=a). We use the notation $\texttt{[A]}_\theta$ as the truth value of `A`, obtained from the individual truth values of its atomic predicates given by a neural network parameterized by θ.

3 Validity of Relaxed Logic

In this section, we propose three criteria that a logic relaxation should satisfy to be useful for learning.

Consistency. The language of logic can declaratively introduce domain knowledge, invariants, or even reasoning skills into neural networks. However, to admit reasoning, tautologies (such as *modus ponens*) should always hold. That is, the truth value of any tautology should be 1 irrespective of the value of its constituent atomic predicates. Equivalently, the integral of the relaxation of a tautology over the domain of its atomic predicates should be 1. We can formalize this intuition.

Definition 1. *Let `T` be a tautology in predicate logic formed with a set of atomic predicates $\mathcal{T}$, and let L a logic relaxation. The* consistency *of `T` in L, denoted as $\kappa^L(\texttt{T})$, is defined as*

$$\kappa^L(\texttt{T}) = \int_0^1 [\texttt{T}] d\mathcal{T} \tag{4}$$

If the consistency $\kappa^L(\texttt{T}) = 1$, we will say that the tautology `T` is consistent under the relaxation L.

Self-consistency. Every boolean statement implies itself. That is, the statement $\texttt{P} \leftrightarrow \texttt{P}$ is a tautology for any `P`. This observation gives us the definition of *self-consistency* of a formula for a relaxation.

Definition 2. *Let `P` be any Boolean formula in predicate logic with a set of atomic sentences $\mathcal{P}$, and let L be a logic relaxation. The* self-consistency *of $\mathcal{P}$ in the logic L, denoted as*

$\kappa_S^L(P)$*, is defined as*

$$\kappa_S^L(P) = \kappa^L(P \leftrightarrow P) = \int_0^1 [P \leftrightarrow P] d\mathcal{P} \tag{5}$$

If $\kappa_S^L(\mathtt{P}) \neq 1$ we will say that formula $\mathtt{P}$ is not self-consistent under a relaxation L. Since we consider a dataset to be a conjunction of facts (equation 1), the self-consistency of large conjunctions allows us to judge whether a dataset implies itself under a relaxation.

Sub-differentiability. Since our eventual goal is to relax declaratively stated knowledge to train neural networks, the relaxations should admit training via backpropagation. As a result, the functions defining the relaxed logical operators should at least be sub-differentiable.

In sum, we consider a logic relaxation to be valid if the following properties hold:

(P1) It must be sub-differentiable over the interval $[0, 1]$.

(P2) It must be consistent for any tautology.

(P3) It must be self-consistent for any Boolean formula.

All the relaxations we study here satisfy property P1[2]. Even among sub-differentiable relaxations, some may be easier than others to learn using gradient based approaches. We consider this empirical question in §5. Properties P2 and P3 do not always hold, and we will prefer relaxations that have higher values of consistency and self-consistency.

4 Truth Preservation of Relaxations

In this section, we will assess the relaxations from table 1 with respect to properties P2 and P3.

4.1 Consistency

Property P2 expects *every* tautology to be consistent under a valid logic relaxation. Since predicate logic admits infinitely many tautologies, we will use a representative set of tautologies for our evaluation. This set contains the *Axiom Schemata* and the *modus ponens* rule of the Hilbert proof system for predicate logic[3], the primitive propositions and a set of elementary properties defined by Russell and Whitehead [1910].

Table 2 shows the consistency for each tautology for our t-norm relaxations. Comparing results across the columns, we see that Łukasiewicz and $\mathcal{R}$-Product t-norm logics *preserve truth* the most across our representative set. In general, we find $\mathcal{R}$ logics to be better at preserving truth than $\mathcal{S}$ logics.

Example 1. *Consistency of* modus ponens *(MP) using $\mathcal{S}$-Product. (The appendix provides additional examples.)*

$$[MP] = [(\mathtt{A} \rightarrow \mathtt{B}) \wedge \mathtt{A}) \rightarrow \mathtt{B}] = 1 + a(1 + ab - a)(b - 1).$$

[2]This is not always the case. In $\mathcal{R}$-Gödel logics, for instance, implications are not sub-differentiable: $[\mathtt{X} \rightarrow \mathtt{Y}]$, takes value 1 if $y \geq x$, and y otherwise.

[3]This choice is motivated because, loosely speaking, every tautology is generated using the Axiom Schemata with the *modus ponens* proof rule.

Tautologies	$\mathcal{S}$-Pro.	$\mathcal{S}$-Gödel	Łuka.	$\mathcal{R}$-Pro.
Axiom Schemata				
$\mathtt{P} \rightarrow (\mathtt{Q} \rightarrow \mathtt{P})$	0.92	0.79	1	1
$(\neg\mathtt{P} \rightarrow \neg\mathtt{Q}) \rightarrow (\mathtt{Q} \rightarrow \mathtt{P})$	0.86	0.75	1	1
Primitive Propositions				
$(\mathtt{P} \vee \mathtt{P}) \rightarrow \mathtt{P}$	0.75	0.75	0.75	1
$\mathtt{Q} \rightarrow (\mathtt{P} \vee \mathtt{Q})$	0.92	0.79	1	1
$(\mathtt{P} \vee \mathtt{Q}) \rightarrow (\mathtt{Q} \vee \mathtt{P})$	0.86	0.75	1	1
Law of excluded middle				
$\mathtt{P} \vee \neg\mathtt{P}$	0.83	0.75	1	0.83
Law of contradiction				
$\neg(\mathtt{P} \wedge \neg\mathtt{P})$	0.83	0.75	1	0.83
Law of double negation				
$\mathtt{P} \leftrightarrow \neg(\neg\mathtt{P})$	0.70	0.75	1	1
Principles of transposition				
$(\mathtt{P} \leftrightarrow \mathtt{Q}) \leftrightarrow (\neg\mathtt{P} \leftrightarrow \neg\mathtt{Q})$	0.60	0.66	1	0.59
Laws of tautology				
$\mathtt{P} \leftrightarrow (\mathtt{P} \wedge \mathtt{P})$	0.69	0.75	0.75	0.5
$\mathtt{P} \leftrightarrow (\mathtt{P} \vee \mathtt{P})$	0.69	0.75	0.75	1
De Morgans Laws				
$(\mathtt{P} \wedge \mathtt{Q}) \leftrightarrow \neg(\neg\mathtt{P} \vee \neg\mathtt{Q})$	0.75	0.75	1	1
$\neg(\mathtt{P} \wedge \mathtt{Q}) \leftrightarrow \neg(\neg\mathtt{P} \vee \neg\mathtt{Q})$	0.75	0.75	1	1
Modus Ponens				
$(\mathtt{P} \wedge (\mathtt{P} \rightarrow \mathtt{Q})) \rightarrow \mathtt{Q}$	0.86	0.75	1	1

Table 2: Consistencies of a representative set of tautologies under different logic relaxations. The appendix includes additional tautologies. We see here (and in the appendix) that the $\mathcal{R}$-Product and Łukasiewicz relaxations are generally more consistent, suggesting that they are preferable to the other two relaxations.

By the definition of consistency (equation 4), we have

$$\kappa(MP) = \int_0^1 \int_0^1 1 + (1 + ab - a) \cdot a \cdot (b - 1) \; \mathrm{d}a \, \mathrm{d}b$$
$$= \frac{31}{36} \approx 0.86.$$

4.2 Self-consistency

The validity property P3 states that *every* well-formed Boolean formula should be self-consistent for a relaxation of the logic to be valid. It turns out that the definition of implications for any $\mathcal{R}$ logic (including Łukasiewicz) guarantees self-consistency of every formula.

Proposition 1. *Every formula is self-consistent under any $\mathcal{R}$-logic relaxation.*

This follows directly from the definition of the t-norm and the properties of residua. The full proof is in the appendix.

However, the same is not true for $\mathcal{S}$-logics. For example, for the conjunction $\mathtt{P} = \mathtt{A} \wedge \mathtt{B}$, the self-consistency under the Gödel relaxation $\kappa_S^{\mathcal{S}-Gödel}(\mathtt{P}) = 0.75$ and under the Product relaxation, we have $\kappa_S^{\mathcal{S}-Prod}(\mathtt{P}) \approx 0.748$. These results suggest that the $\mathcal{R}$-Product and Łukasiewicz relaxations are preferable from the perspective of property P3 as well.

Intriguingly, we find that the $\mathcal{S}$-Product logic is *eventually* self-consistent for large monotone conjunctions.

Proposition 2. *Let $A = \bigwedge_{i=1}^{n} A_i$ be a conjunction consisting of n atomic predicates $A_1, A_2, \ldots, A_n$. The self consistency of A is given by $\kappa_S^{\mathcal{S}-Product}(A) = 1 - \frac{2}{2^n} + \frac{3}{3^n} - \frac{2}{4^n} + \frac{1}{5^n}$.*

Proof. By induction over the size of the conjunction n. The full proof is in the appendix. □

We see that, as $n \to \infty$, the self-consistency of a monotone conjunction approaches 1. In other words, large conjunctions (e.g., representing large datasets) are essentially self-consistent under the $\mathcal{S}$-Product relaxation.

5 Empirical Comparisons of Relaxed Logic

In this section, we empirically study the differences between the different logic relaxations using two tasks: recognizing digits and arithmetic operations, and text chunking. In both tasks, we set up the learning problem in terms of logic, and compare models learned via different logic relaxations.

5.1 Recognizing digits and arithmetic operations

These experiments build upon our running example from §2. We seek to categorize hand written digits; i.e., we learn the predicate `Digit`. In addition, we also seek to predict the sum and product (modulo 10) of two handwritten digit images. These correspond to the predicate `Sum` we have seen, and a new analogous predicate `Product`.

We use the popular MNIST dataset [LeCun and Cortes, 2010] for our experiments, but *only* to supervise the `Digit` classifier. Rather than directly supervising the other two classifiers, we use coherence constraints over *unlabeled* image pairs that connect them to the `Digit` models. The constraint for the `Sum` classifier is shown in equation 2, and the one for the `Product` classifier is similarly defined.

We set up the learning problem as defined in §2 and compare performance across different relaxations.

Data and setup. We partition the 60k MNIST training images into TRAIN and DEV sets, with 50k and 10k images respectively. To supervise the `Digit` model, we sample 1k, 5k and 25k labeled images from TRAIN to form three DIGIT sets. The coherence constraints are grounded in 5k *unlabeled* image pairs consisting of images from TRAIN that are not in any DIGIT set, giving us the PAIR dataset.

For evaluating the `Digit` model, we use the original 10k TEST examples from MNIST. For development and evaluation of the operator models, we sample random image pairs DEV and TEST sets to create the PairDEV and PairTEST sets respectively. The ground truth `Sum` and `Product` labels for these image pairs can be computed by the sum and product modulo 10 of the image labels.

We use CNNs as the `Digit`, `Sum` and `Product` models, with parameters denoted by θ, ϕ, ψ respectively. For the operator models, we concatenated the two images to get inputs for the CNNs. To jointly train these models using the labeled DIGIT data and the unlabeled PAIR datasets, we define a loss function by relaxing the conjunction of `Digit` predicates over the DIGIT examples and the coherence constraints over the PAIR examples. That is, learning requires minimizing:

$$-\Bigg[\Bigg(\bigwedge_{(x,y)\in D} \texttt{Digit}(x,y)\Bigg) \wedge \Bigg(\bigwedge_{PAIR} \texttt{Sum Consistency}\Bigg) \wedge \Bigg(\bigwedge_{PAIR} \texttt{Product Consistency}\Bigg)\Bigg]_{\theta,\phi,\psi} \tag{6}$$

In practice, we found that it is important to use a hyperparameter λ that weights the relaxed coherence constraints in the loss. We used the DEV sets for hyperparameter tuning using the average accuracies of the three models. The appendix provides further details for reproducibility.

	1000	5000	25000
$\mathcal{S}$-Gödel	94.97	97.35	97.74
$\mathcal{S}$-Product	95.13	98.24	99.03
$\mathcal{R}$-Product	**96.32**	**98.42**	**99.23**
Łukasiewicz	95.76	98.00	99.14

Table 3: Jointly trained `Digit` accuracies on DIGIT sizes 1k, 5k, 25k and operators (`Sum`,`Prod`) on PAIR size 5k.

	1000	5000	25000
$\mathcal{S}$-Gödel	87.31	**91.14**	91.54
$\mathcal{S}$-Product	76.90	88.57	90.12
$\mathcal{R}$-Product	**87.98**	90.81	**91.81**
Łukasiewicz	75.92	84.48	82.34

Table 4: Jointly trained operators accuracies (avg.`Sum` and `Product`) on PAIR size 5k. and `Digit` on DIGIT sizes 1k, 5k, 25k

Results. Table 3 reports accuracies for the `Digit` classifier trained with different sizes of DIGIT, and the coherence constraints instantiated over the 5k PAIR examples. We observe that $\mathcal{R}$-Prod relaxation dominates across all settings, with higher gains when there are fewer labeled examples. (The bold entries in this and other tables are statistically significantly better than the other relaxations at $p < 0.05$.) Table 4 reports the average of `Sum`, and `Prod` accuracies for the same settings, and Table 5 shows the fraction of PairTEST examples where the coherence constraints are satisfied. From these results, we see that the $\mathcal{R}$-Product and $\mathcal{S}$-Gödel relaxations offer the best accuracies.

Interestingly, Łukasiewicz is the least accurate relaxation. However, the losses compiled using Łukasiewicz and Gödel t-norms were unstable, and the results shown here were achieved with additional assumptions. We defer this technical discussion to §5.3, and conclude that, the stability and accuracy of the $\mathcal{R}$-Product relaxation suggests that it is the most suitable relaxation for this class of problems.

	1000	5000	25000
$\mathcal{S}$-Gödel	86.40	91.39	91.85
$\mathcal{S}$-Product	79.02	89.25	90.25
$\mathcal{R}$-Product	**89.07**	**91.48**	**92.14**
Łukasiewicz	77.38	85.43	82.62

Table 5: Coherence accuracies (Avg.`Sum` and `Prod`) from jointly training `Digit` with DIGIT sizes 1k, 5k, 25k and operators with PAIR size 5k.

	1000	5000	25000
$\mathcal{S}$-Gödel	95.22	97.56	97.59
$\mathcal{R}/\mathcal{S}$-Product	**96.66**	**98.37**	**99.12**
Łukasiewicz	96.47	98.46	98.76

Table 6: Pipelined `Digit` accuracies over DIGIT sizes 1k, 5k, 25k.

Joint learning vs. Pipelines In the coherence constraints in equation 2, if we knew both the `Digit` terms, we can deterministically compute the value of the `Sum`. This suggests a pipeline strategy for training the three models, where we can train the `Digit` classifier alone, and use it to assign (noisy) labels to the unlabeled PAIR data. Subsequently, we can train the `Sum` and `Product` models independently. How does the joint training strategy compare to this pipeline?

Importantly, we should note that both the joint and the pipeline strategies instantiate the declarative learning approach outlined in §2 where logical facts are compiled into a optimization learning problem via the logic relaxations. However, in the pipeline, the coherence constraints are not explicitly involved in loss, because the noisy labels already satisfy them. Since conjunctions are identical for the $\mathcal{S}$ and $\mathcal{R}$ relaxations, they give the same results.

Tables 6, 7, and 8 show the results of these pipelined experiments. We observe the same trends as in the joint learning experiments: $\mathcal{R}$-Product achieves highest performance.

By comparing the joint learning with the pipeline results, we observe that the latter are slightly higher across relaxations in almost all data settings. Of course, as mentioned above, the pipelining strategy is only viable here because of the special form of our constraint.

5.2 Text chunking

Our second set of experiments use the NLP task of text chunking using the CoNLL 2000 dataset [Sang and Buchholz, 2000]. This task illustrates how relaxed logic can be used to derive loss functions for sequential inputs and outputs.

	1000	5000	25000
$\mathcal{S}$-Gödel	88.34	90.89	91.28
$\mathcal{R}/\mathcal{S}$-Product	**89.71**	**92.22**	**92.50**
Łukasiewicz	82.95	86.89	88.58

Table 7: Pipelined operator accuracies (Avg.`Sum` and `Product`) over (noisy) labeled PAIR size 5k with `Digit` model trained on DIGIT sizes 1k, 5k, 10k.

	1000	5000	25000
$\mathcal{S}$-Gödel	86.67	90.62	91.28
$\mathcal{R}/\mathcal{S}$-Product	**89.85**	**92.47**	**92.75**
Łukasiewicz	83.51	87.22	88.70

Table 8: Coherence accuracies (Avg.`Sum` and `Prod`) over (noisy) labeled PAIR size 5k with `Digit` model trained on DIGIT sizes size 1k, 5k, 25k.

Text chunking is a sequence tagging problem, where each word of a sentence is assigned a phrase type. For example, in the sentence '*John is playing in the park.*', the word '*John*' is labeled as `B-NP`, indicating that it starts a noun phrase (`NP`). We use the standard BIO labeling scheme: `B-X` indicates the start of a new phrase labeled `X`, `I-X` indicates the continuation of a phrase labeled `X`, and `O` marks words that do not belong to any of the predefined phrase types.

In case of text chunking, the input x is a sequence of words and correspondingly, output y is a sequence of labels (phrase types). Consequently, each position in the sequence is associated with a predicate. For an input x with n words, we have n predicates, one per word.

Using our notation from section 2 we define the predicate, $\texttt{Tag}(x_i, y_i)$, to denote that i^{th} word in input x is assigned the label y_i. For our preceding example, *John* being assigned the label `B-NP` corresponds to the predicate $\texttt{Tag}(\text{John}, \texttt{B-NP})$.

Constraints. For each position in the sequence, we have constraints defining pairwise label dependencies. If a word in a sentence has a B/I label with a certain phrase type, then the next word cannot have a I label with a *different* phrase type. For example, we have

$C_{i,1}$: $\forall i,\ \texttt{Tag}(x_i, \texttt{B-NP}) \rightarrow \neg\texttt{Tag}(x_{i+1}, \texttt{I-VP})$

$C_{i,2}$: $\forall i,\ \texttt{Tag}(x_i, \texttt{I-NP}) \rightarrow \neg\texttt{Tag}(x_{i+1}, \texttt{I-VP})$

In general, given a labeled dataset $\mathcal{D}$, and a set $\mathcal{C}$ of k constraints for each word position, we can write the conjunction of all constraints as:

$$\bigwedge_{x,y \in \mathcal{D}} \left\{ \bigwedge_i \left(\texttt{Tag}(x_i, y_i) \bigwedge_k C_{i,k} \right) \right\} \tag{7}$$

We can now use this Boolean formula to state the goal of learning as finding the model parameters of a neural network which maximizes the value of the relaxation derived for each t-norm. We used a bidirectional LSTM over GloVe embeddings Pennington *et al.* [2014] to instantiate `Tag`. The appendix provides further details.

Experiments and Results. We compare the four t-norms on two settings. First, in purely supervised learning, the model only learns from labeled examples. Second, we augment the labeled dataset with the constraints described above to study the effectiveness of incorporating simple constraints on outputs. For both of these settings, we also study the models in a low data regime with training data restricted to 10%. Following previous work [Sang and Buchholz, 2000], we use F1 score as our evaluation metric.

We report performances of the models trained with different t-norms in table 9. We observe that for supervised learning (top rows), both variants of product t-norm outperform

% Train	$\mathcal{S}$-Gödel	$\mathcal{S}$-Prod.	$\mathcal{R}$-Prod.	Łukasiewicz
10%	70.29	**79.46**	**79.46**	76.50
100%	85.33	**89.18**	**89.18**	87.25
10% + C_k	70.81	79.71	**80.14**	76.80
100% + C_k	85.49	89.19	**89.75**	87.59

Table 9: Results for Text Chunking. F1 scores on test set of CoNLL 2000 dataset. Product t-norms outperform other t-norms for purely supervised setting (top rows). $\mathcal{R}$-Product performs best among all t-norms when constraints are augmented into the neural models (bottom rows). Rows with + C_k include constraints.

the other two t-norms by a significant margin. We again observe that for purely supervised learning, both $\mathcal{R}$-Product and $\mathcal{S}$-Product produce identical results since no implication constraints are used. Further, $\mathcal{S}$-Gödel performs the worst among the four t-norms. Note that this observation is consistent with our analysis from table 2 where $\mathcal{S}$-Gödel was found to be least consistent with tautologies.

Let us now look at the results of augmenting supervised learning with simple output constraints (table 9, bottom rows). First, we observe that incorporating simple constraints into neural models, using the framework discussed in this work, can indeed improve their performance in all cases. Particularly, this improvement is more pronounced in low data regimes, since models need to rely more on external background knowledge when training data is small.

We also find that both variants of product t-norms outperform $\mathcal{S}$-Gödel and Łukasiewicz. However, in both data regimes, $\mathcal{R}$-Product outperforms $\mathcal{S}$-Product when incorporating constraints into the neural model. We observe that this observation is consistent with our results so far.

5.3 Theory vs. Experiments

From our analysis of t-norms in §4, we found that Łukasiewicz and $\mathcal{R}$-Product are the most preferable. Empirically, $\mathcal{R}$-Product outperforms the other relaxations.

The consistency analysis suggests that since $\mathcal{S}$-Gödel is least consistent, we should also expect $\mathcal{S}$-Gödel to empirically perform the worst. This argument is reinforced by the results across all tasks and settings, if we naively applied the Gödel t-norm to define the loss. The $\mathcal{S}$-Gödel results shown in this work were obtained by warm starting the learning using the $\mathcal{R}$-Product relaxation. Without the warm start, we found that not only is learning unstable, the resulting accuracies were also low.

One other discrepancy bears attention: although Łukasiewicz is most preferable in terms of consistency on tautologies, the experiments suggest otherwise. From an empirical perspective, a valid relaxation of logic should provide sufficient gradient signal to make learning feasible. We discovered that loss functions defined by the Łukasiewicz t-norm are not amenable to gradient based learning. Here, we briefly explain this phenomenon.

Consider the Łukasiewicz conjunction over n atoms:

$$\left[\bigwedge_i^n a_i\right] = \max\left(0, \sum_i^n a_i - (n-1)\right)$$

For this operation to provide a non-zero output, we need $\sum_i^n a_i > (n-1)$, or on average, each of the conjuncts a_i should exceed $\frac{n-1}{n}$. That is, to provide a useful signal, Łukasiewicz t-norm requires the model to assign high probabilities to correct labels. This, of course, is not true for a randomly initialized model when learning starts, contributing to near zero gradients, and no model updates at all.

To make learning feasible with Łukasiewicz t-norm, we implemented a less strict definition of conjunction inspired by the MAX-SAT relaxation of Bach *et al.* [2017]. This transforms our learning objective to the form $\max_\theta\left(\sum_i^n a_i\right)$. An important consequence of this approximation the Łukasiewicz t-norm in the experiments is merely an approximation of the original Łukasiewicz t-norm.

6 Related work and Discussion

The use of t-norm relaxations to encode knowledge into learning is increasingly prevalent in recent years [e.g., Wang *et al.*, 2020; Minervini *et al.*, 2017]. Additionally, other work Grefenstette [2013], and Nandwani *et al.* [2019] also implicitly uses t-norms to similar ends. These works do not frame the learning problem as a declarative statement encoded using a single predefined logical language, as it in the case of Sikka *et al.* [2020] and Giannini *et al.* [2019].

Our experimental setup is closer to Li *et al.* [2019]; Asai and Hajishirzi [2020]; Wang *et al.* [2020] where both data and background knowledge are declaratively stated and encoded in one single t-norm relaxation that defines the loss. Our experiments for the extended MNIST digit classification are inspired by Manhaeve *et al.* [2018]. Indeed, they use constraints similar to our coherence constraints to help the labeled data towards better prediction. Of course, the goal of this work is to empirically investigate which relaxation is best suited for use in such problems.

To our knowledge, none of the previously mentioned works analyzes the choice of the relaxation they use. Our work focuses on directly comparing these relaxations both theoretically and empirically. Perhaps the closest work to this paper is that of van Krieken *et al.* [2020] in that they also provide theoretical and empirical analysis using different t-norm relaxations. However, there are two key differences : (a) they do not treat labeled data as part of the declarative problem specification and the constraints are added into a standard cross entropy loss, and (b) their analysis involves the derivatives of the losses and does not discuss the consistency and self-consistency properties.

7 Conclusions

In this work, we studied the question of how best to relax declarative knowledge to define loss functions. To this end, we define a set of criteria that characterizes which relaxations of logic would be most amenable for preserving tautologies, and and offer support for gradient based learning. We also present empirical studies on two tasks via the paradigm of formulating entire learning problems via logic. All our analyzes concur that the $\mathcal{R}$-Product relaxation is best suited for learning in this paradigm.

References

Akari Asai and Hannaneh Hajishirzi. Logic-Guided Data Augmentation and Regularization for Consistent Question Answering. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5642–5650, Online, July 2020. Association for Computational Linguistics.

Stephen H Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. Hinge-loss markov random fields and probabilistic soft logic. *The Journal of Machine Learning Research*, 18(1):3846–3912, 2017.

Francesc Esteva, Lluís Godo, Petr Hájek, and Mirko Navara. Residuated fuzzy logics with an involutive negation. *Arch. Math. Log.*, 39(2):103–124, 2000.

Francesco Giannini, Giuseppe Marra, Michelangelo Diligenti, Marco Maggini, and Marco Gori. On the relation between loss functions and t-norms. 07 2019.

Edward Grefenstette. Towards a formal distributional semantics: Simulating logical calculi with tensors. In **SEM@NAACL-HLT*, 2013.

Erich Peter Klement, Radko Mesiar, and Endre Pap. *Triangular Norms*, volume 8. Springer Science & Business Media, 2013.

Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

Tao Li, Vivek Gupta, Maitrey Mehta, and Vivek Srikumar. A Logic-Driven Framework for Consistency of Neural Models. In *EMNLP*, 2019.

Tao Li, Parth Anand Jawale, Martha Palmer, and Vivek Srikumar. Structured Tuning for Semantic Role Labeling. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8402–8412, Online, July 2020. Association for Computational Linguistics.

Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with restarts. *CoRR*, abs/1608.03983, 2016.

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. *CoRR*, abs/1805.10872, 2018.

Karl Menger. Statistical metrics. *Proceedings of the National Academy of Sciences of the United States of America*, 28(12):535, 1942.

Pasquale Minervini, Thomas Demeester, Tim Rocktäschel, and Sebastian Riedel. Adversarial sets for regularising neural link predictors. In *UAI*, 2017.

Yatin Nandwani, Abhishek Pathak, Parag Singla, et al. A primal dual formulation for deep learning with constraints. In *NeurIPS*, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. Injecting logical background knowledge into embeddings for relation extraction. In *NAACL*, 2015.

Bertrand Russell and Alfred North Whitehead. *Principia Mathematica Vol. I*. Cambridge University Press, 1910.

Erik Tjong Kim Sang and Sabine Buchholz. Introduction to the conll-2000 shared task chunking. In *CoNLL*, 2000.

Berthold Schweizer, Abe Sklar, et al. Statistical metric spaces. *Pacific J. Math*, 10(1):313–334, 1960.

Karan Sikka, Andrew Silberfarb, John Byrnes, Indranil Sur, Ed Chow, Ajay Divakaran, and Richard Rohwer. Deep adaptive semantic logic (dasl): Compiling declarative knowledge into deep neural networks, 2020.

Emile van Krieken, Erman Acar, and Frank van Harmelen. Analyzing differentiable fuzzy implications. In Diego Calvanese, Esra Erdem, and Michael Thielscher, editors, *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, Rhodes, Greece, September 12-18, 2020*, pages 893–903, 2020.

Haoyu Wang, Muhao Chen, Hongming Zhang, and Dan Roth. Joint constrained learning for event-event relation extraction. In *EMNLP*, 2020.

Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 5498–5507. PMLR, 2018.

A Appendix

Proof for Proposition 1.

Proof. The canonical t-norms: , Łukasiewicz and Product are left-continuous binary operators. For every left-continuous t-norm T, there is a unique binary operation denoted $\rightarrow$ on the interval $[0,1]$ satisfying:

$$T(z,x) \leq y \text{ iff. } z \leq (x \rightarrow z) \quad \forall x,y,z \in [0,1]$$

This operation $\rightarrow$ is called the Residuum of T and it generalizes the implication connective in the corresponding $t-norm$ logic. From the definition of residua it can be proven that,

$$(x \rightarrow y) = \sup\{z \mid T(z,x) \leq y\} \tag{8}$$

in consequence, for all x and y in the interval $[0,1]$ we have,

$$(x \rightarrow y) = 1 \text{ iff. } x \leq y$$

Now, given any well-formed Boolean formula $\mathtt{P}$, and an $\mathcal{R}$-logic L relaxation we want to prove that $\kappa_S^L(\mathtt{P})$=1. This is the same as to prove $\kappa^L(\mathtt{P} \leftrightarrow \mathtt{P}) = 1$. But the latter is true directly from (8).

□

Proof for Proposition 2.

Proof. Let $F = \bigwedge A_i \leftrightarrow \bigwedge A_i$. By definition the $\mathcal{S}$-Product logic connectives,

$$[F] = \left(1 + \left(\prod_{i=1}^{n} A_i\right)^2 - \left(\prod_{i=1}^{n} A_i\right)\right)^2$$

.

By definition of self-consistency, we have

$$\kappa_S^{\mathcal{S}-Product}(A) = \int_0^1 [F] d\mathcal{F}$$

Let,

$$b_{n-1} = \prod_{i=1}^{n-1} A_i$$

then,

$$[F] = (1 + A_n^2 b_{n-1}^2 - A_n b_{n-1})^2.$$

Integrating out the n^{th} variable A_n by expanding out the polynomial, we obtain

$$\begin{aligned}\kappa_n &= \int_0^1 \left(1 + A_n^2 b_{n-1}^2 - A_n b_{n-1}\right)^2 dA_n \\ &= 1 - b_{n-1} + b_{n-1}^2 - \frac{1}{2} b_{n-1}^3 + \frac{1}{5} b_{n-1}^4.\end{aligned}$$

Applying the same strategy as before, we can define

$$b_{n-2} = \prod_{i=1}^{n-2} A_i$$

then,

$$b_{n-1} = A_{n-1} b_{n-2}.$$

Substituting in b_{n-1} in κ_n and taking the integral over the $(n-1)^{th}$ variable we obtain,

$$\begin{aligned}\kappa_{n-1} &= \int_0^1 \kappa_n \, dA_{n-1} \\ &= \int_0^1 1 - A_{n-1} b_{n-2} + A_{n-1}^2 b_{n-2}^2 \\ &\qquad - \frac{1}{2} A_{n-1}^3 b_{n-2}^3 + \frac{1}{5} A_{n-1}^4 b_{n-2}^4 \, dA_{n-1} \\ &= 1 - \frac{2}{2^2} b_{n-2} + \frac{3}{3^2} b_{n-2}^2 - \frac{2}{4^2} b_{n-2}^3 + \frac{1}{5^2} b_{n-2}^4.\end{aligned}$$

Repeating this procedure, every subsequent integral will add one to the degree of the denominators in each term. This allows us to generalize the integral as

$$\kappa_{n-k} = 1 - \frac{2}{2^k} b_{n-k} + \frac{3}{3^k} b_{n-k}^k - \frac{2}{4^k} b_{n-k}^3 + \frac{1}{5^k} b_{n-k}^4$$

where,

$$b_{n-k} = \prod_{i=1}^{n-k}.$$

In particular, when $k = n$, the product becomes 1. Therefore, substituting b_{n-k} in κ_{n-k} we obtain,

$$\kappa_S^{\mathcal{S}-Product}(A) = \kappa_0 = 1 - \frac{2}{2^n} + \frac{3}{3^n} - \frac{2}{4^n} + \frac{1}{5^n}$$

□

B Reproducibility

B.1 Digit Arithmetic experiments

Model Architecture. For `Digit`, `Sum` and `Product` models we used 2-layer Convolutional Neural Networks with 3-by-3 padded filters, with ReLU activation and 2-by-2 Maxpool layers with stride 2 in between, with 2 fully connected layers with ReLU activation and dropout ($p = 0.5$) in between.

Seeds and multiple runs. We used a random seed 20 to create all the dataset used for training, evaluation, testing. We also used seed 20 for hyper-parameter tuning. For performed three runs of every experiment using seeds 0, 20, and 50. The standard deviation from the three runs with different seeds was high (up to 5%) for some of the relaxations over the scarce data settings (DIGIT size 1000 and/or PAIR size 1000). In these cases, we performed extra runs with seeds 1, 10, and 60, reporting the average of the best three.

Hyper-parameter tuning Details Validation sets are used for tuning hyper-parameters: learning rate, batch size, optimizer, lambda coefficient for the coherence constraints in the loss. Specifically we performed grid search for learning values: $\{10^{-1}, 5 \times 10^{-2}, 10^{-2}, 5 \times 10^{-3}, 10^{-3}, 5 \times 10^{-4}, 10^{-4}, 5 \times 10^{-5}, 10^{-5}\}$; lambda coefficient values: $\{0.05, 0.1, 0.5, 1, 1.5, 2\}$; batch sizes: {8, 16, 32, 64 }. We also perform treated the optimization used as a hyper-parameters, using the one resulting in best performance between standard Stochastic Gradient Descent and Adam optimization.

Number of epochs for training. The number of epoch for convergence varied between 100 to 1000 epochs across t-norm relaxations and data settings. All of the experiments were run for 1600 epochs.

t-tests. We performed t-tests with p value 0.05 reaching statistical significance in all the comparisons of the models performances upon we support our conclusions.

Warm-restarts. Using the Stochastic Gradient Descent with Restarts techniques Loshchilov and Hutter [2016], brought considerably more stability to the experiments. We used the best number iterations for the first restart (T_0) among 50, 100 and 200, with a factor of increase of 1 (T_{mult})

B.2 Experimental Details for Text Chunking

Model Architecture. For all our models, we use bidirectional RNN based models, experimenting with LSTMs and GRUs. We use GloVe word embeddings of 300 dimension at input along with 100 dimensional character embeddings. We fix the dropout rate to 0.5 for all our experiments.

Hyper-parameter Tuning Since, the original CoNLL 2000 dataset does not accompany a validation set, for each experiment, we select 10% of the total training data for validation. We perform hyperparamter tuning using grid search on the following parameters: hidden unit size of RNN in set 200. 256, 300, 400, 500, 512, learning rate in set 0.001, 0.005, 0.0001.

B.3 Weighted Constrained Loss

We observed that in order to stabilise training, we had to use a λ (hyperparameter) to relatively weigh the loss from constraints. We found that product based t-norms were less sensitive to different values of λ.

For chunking, for example, for Łukasiewicz t-norm, values around 0.0001 worked best, while for $\mathcal{R}$-Product, values greater than 1 provided best results on validation set.

B.4 Code and Computing Infrastructure

We implemented all our experiments in the deep learning library PyTorch Paszke *et al.* [2019] with equipment: CPU: AMD EPYC 7601, 32 cores, 2.2 GHz , Memory: 512 gb, GPU:Titan RTX, Disk Space: 8 TB.

C Learning behaviour of neural models

C.1 Product relaxations

$\mathcal{S}$-Product and $\mathcal{R}$-Product logic were the most stable and less brittle across different data settings.

C.2 Discussion about Gradients in Łukasiewicz relaxation

From an empirical perspective, a valid relaxation of logic should provide enough gradient signal to make the gradient-based learning process possible. One relevant case of a logic relaxation that does not satisfy this property is the Łukasiewicz t-norm. By inspecting the Łukasiewicz conjunction connective definition:

$$\left[\bigwedge_i^n a_i\right] = \max\left(0, \sum_i^n a_i - (n-1)\right)$$

we can see that it requires almost absolute certainty from of each of its conjuncts to result in a non-zero truth value. Meaning that the gradient signal provided from an objective of the form ([**??**]) is almost null.

In order to have some insight about Łukasiewicz, logic in our experiments, for our reported experiments we implemented a less strict definition of conjunction inspired by the MAX SAT relaxation in Bach *et al.* [2017] and assume the best case scenario $\sum_i^n a_i \geq n-1$. In this case, the loss we want to optimize goes from

$$\max_\theta \left(\max\left(0, \sum_i^n a_i - (n-1)\right)\right)$$

to

$$\max_\theta \left(\sum_i^n a_i\right) \tag{9}$$

C.3 Practical considerations for $\mathcal{S}$-Gödel logic

In practice, we found that optimizing the loss function derived from the $\mathcal{S}$-Gödel relaxation was difficult. In order to make it work we applied the following techniques:

- By definition of $\mathcal{S}$-Gödel conjunction, we have to update over the example with the minimum value in the whole epoch. Instead, we applied mini-batch descent over the minimum of the batch, and proceed with the standard SGD process. This means that we used $\mathcal{S}$-Gödel over the batches but the $\mathcal{S}$-Product t-norm, which agrees with the cross-entropy loss, over the epochs.
- Given that at the very early stage the system is uncertain about every example, and $\mathcal{S}$-Gödel conjunction operates over the "worst" example, it is hard to get any meaningful signal. In order to brake these the ties, we "warmed-up"the system by running between 2 to 10 epochs of learning using $\mathcal{S}$-Product logic which is still within the same $\mathcal{S}$ family of relaxations. Interestingly, even though the warm-up was needed to make for both frameworks to work, the joint system needed only one or two to warm-up while the data augmentation system needed between 6 to 10 epochs to start learning on its own. These warm-up processes represented 40% on average of the final accuracies obtained across our experiments.
- In the joint model, we noticed that the accuracies for the `Digit` classifier decreased after a few epochs not letting the model to continue training, this was caused because the model is trying to reach the trivial solution to satisfy the coherence constraints. In order to avoid this, we freezed the parameters corresponding to the `Digit` model right after the warm-up stage; in other words, we trained `Digit` classifier using $\mathcal{S}$-Product relaxation for a few epochs and it was enough to successfully continue the training.

D Examples of Consistency calculation

Calculating consistency for Modus Ponens (MP).

For $\mathcal{S}$-Gödel t-norm logic:

$$
\begin{aligned}
[MP] &= [(\mathtt{A} \to \mathtt{B}) \wedge \mathtt{A}) \to \mathtt{B}] \\
&= \max\Big(1 - ([\mathtt{A} \to \mathtt{B}] \wedge [\mathtt{A}]), [\mathtt{B}]\Big) \\
&= \max\Big(1 - \min\Big(\max(1-a, b), a\Big), b\Big)
\end{aligned}
$$

Calculating consistency,

$$
\begin{aligned}
&\kappa^{\mathcal{S}-Gödel}(MP) \\
&= \int_0^1 \int_0^1 \max\Big(1 - \min\Big(\max(1-a, b), a\Big), b\Big)\ dadb \\
&= \frac{3}{4} = 0.75
\end{aligned}
$$

For $\mathcal{R}$-Prod t-norm logic We have two cases:

1. $[\mathtt{A}] \leq [\mathtt{B}]$

   By the definition of $\mathcal{R}$-Prod implication

   $$[\mathtt{A} \to \mathtt{B}] = 1$$

   then,

   $$[(\mathtt{A} \to \mathtt{B}) \wedge \mathtt{A}] = 1 \cdot [\mathtt{A}] = [\mathtt{A}]$$

   therefore,

   $$
   \begin{aligned}
   [MP] &= [(\mathtt{A} \to \mathtt{B}) \wedge \mathtt{A}) \to \mathtt{B}] \\
   &= \begin{cases} 1 & \text{if} \quad [(\mathtt{A} \to \mathtt{B}) \wedge \mathtt{A}] \leq [\mathtt{B}] \\ \frac{[\mathtt{B}]}{[(\mathtt{A} \to \mathtt{B}) \wedge \mathtt{A}]} & \text{else} \end{cases} \\
   &= \begin{cases} 1 & \text{if} \quad [\mathtt{A}] \leq [\mathtt{B}] \\ \frac{[\mathtt{B}]}{[(\mathtt{A} \to \mathtt{B}) \wedge \mathtt{A}]} & \text{else} \end{cases} \\
   &= 1
   \end{aligned}
   $$

2. $[\mathtt{A}] > [\mathtt{B}]$

   By the definition of $\mathcal{R}$-Prod implication

   $$[\mathtt{A} \to \mathtt{B}] = \frac{[\mathtt{B}]}{[\mathtt{A}]}$$

   then,

   $$[(\mathtt{A} \to \mathtt{B}) \wedge \mathtt{A}] = \frac{[\mathtt{B}]}{[\mathtt{A}]} \cdot [\mathtt{A}] = [\mathtt{B}]$$

   therefore,

   $$
   \begin{aligned}
   [MP] &= [(\mathtt{A} \to \mathtt{B}) \wedge \mathtt{A}) \to \mathtt{B}] \\
   &= \begin{cases} 1 & \text{if} \quad [(\mathtt{A} \to \mathtt{B}) \wedge \mathtt{A}] \leq [\mathtt{B}] \\ \frac{[\mathtt{B}]}{[(\mathtt{A} \to \mathtt{B}) \wedge \mathtt{A}]} & \text{else} \end{cases} \\
   &= \begin{cases} 1 & \text{if} \quad [\mathtt{B}] \leq [\mathtt{B}] \\ \frac{[\mathtt{B}]}{[(\mathtt{A} \to \mathtt{B}) \wedge \mathtt{A}]} & \text{else} \end{cases} \\
   &= 1
   \end{aligned}
   $$

Hence, $\kappa^{\mathcal{R}-Prod}(MP) = 1$

	1000	5000	25000
$\mathcal{S}$-Gödel	94.47	97.01	96.73
$\mathcal{S}$-Product	92.53	97.80	**99.08**
$\mathcal{R}$-Product	**95.53**	**97.88**	96.03
Łukasiewicz	95.13	96.15	97.86

Table 10: Jointly trained `Digit` accuracies on DIGIT sizes 1k, 5k, 25k and operators (`Sum`,`Prod`) on PAIR size 1k.

	1000	5000	25000
$\mathcal{S}$-Gödel	93.51	97.71	98.03
$\mathcal{S}$-Product	95.17	98.17	98.97
$\mathcal{R}$-Product	**96.22**	**98.43**	*99.06*
Łukasiewicz	94.76	98.15	99.09

Table 11: Jointly trained `Digit` accuracies on DIGIT sizes 1k, 5k, 25k and operators (`Sum`,`Prod`) on PAIR size 25k.

E $\mathcal{R}$ logics and the connective for negation

In table 1, we abuse the nomenclature for $\mathcal{R}$-Product and $\mathcal{R}$-Gödel logics. In these relaxations, the negation and conjunction connectives are respectively defined as

$$n_\neg(x) = \begin{cases} 1 & \text{if x=0} \\ 0 & \text{otherwise} \end{cases}$$

and,

$$x \wedge y = \begin{cases} 0 & \text{if x=y=0} \\ 1 & \text{otherwise} \end{cases}$$

Unfortunately, these function are not sub-differentiable. Instead, we introduce the corresponding SBL$_\sim$ extensions for residuated t-norm logics $\mathcal{R}$ Esteva *et al.* [2000] which are constructed using the involutive negation $n_\neg(x) = 1 - x$ and sub-differentiable in all of their connectives. We keep the same names $\mathcal{R}$-Product and $\mathcal{R}$-Gödel for simplicity.

F Additional Experiments and Results

We additionally report experimental results for different PAIR and DIGIT sizes not reported in the main paper. We observe similar trends to those observed for the experiments reported in the main paper.

For the experiments described in section 5, we also evaluate the models for mathematical properties not covered by constraints. For instance, although models were only trained using coherence constraints, we evaluate how well they satisfy other common mathematical properties of sum and product operations such as Commutativity, Associativity, and Distributivity (see table 16, table 18, and table 17). Again, our main observation is that in most of the experiments, models trained with $\mathcal{R}$-Product based relaxations outperform other t-norm relaxations on these properties.

Finally, in table 26, we report consistency values for the five (including $\mathcal{R}$-Gödel) t-norms on 25 different tautologies.

	1000	5000	25000
$\mathcal{S}$-Gödel	**62.68**	**62.03**	64.06
$\mathcal{S}$-Product	49.60	59.11	60.61
$\mathcal{R}$-Product	53.32	59.32	**66.47**
Łukasiewicz	44.50	45.28	50.78

Table 12: Jointly trained operators accuracies (avg.`Sum` and `Product`) on PAIR size 1k. and `Digit` on DIGIT sizes 1k, 5k, 25k

	1000	5000	25000
$\mathcal{S}$-Gödel	88.34.06	94.91	95.86
$\mathcal{S}$-Product	83.63	90.91	95.03
$\mathcal{R}$-Product	**89.64**	**95.65**	**96.07**
Łukasiewicz	70.76	89.35	90.36

Table 13: Jointly trained operators accuracies (avg.`Sum` and `Product`) on PAIR size 25k. and `Digit` on DIGIT sizes 1k, 5k, 25k

	1000	5000	25000
$\mathcal{S}$-Gödel	**64.06**	**62.14**	63.86
$\mathcal{S}$-Product	52.07	59.41	60.70
$\mathcal{R}$-Product	54.39	59.08	**67.04**
Łukasiewicz	45.42	45.76	50.88

Table 14: Coherence accuracies (Avg.`Sum` and `Prod`) from jointly training `Digit` with DIGIT sizes 1k, 5k, 25k and operators with PAIR size 1k.

	1000	5000	25000
$\mathcal{S}$-Gödel	85.96	94.56	95.35
$\mathcal{S}$-Product	85.80	91.66	95.55
$\mathcal{R}$-Product	**90.75**	**95.91**	**96.34**
Łukasiewicz	72.89	90.23	90.65

Table 15: Coherence accuracies (Avg.`Sum` and `Prod`) from jointly training `Digit` with DIGIT sizes 1k, 5k, 25k and operators with PAIR size 25k.

		Commut.	Assoc.	Dist.
1000	$\mathcal{S}$-Gödel	**59.12**	**47.64**	**43.29**
	$\mathcal{S}$-Product	55.82	45.59	39.66
	$\mathcal{R}$-Product	49.44	41.42	32.11
	Łukasiewicz	43.00	37.35	27.20
5000		89.98	85.78	**82.60**
		79.54	69.72	66.47
		90.91	**87.03**	82.02
		76.49	65.66	64.30
25000		93.90	89.45	85.72
		89.83	82.21	78.99
		95.30	**91.80**	**86.28**
		79.85	63.31	55.67

Table 16: Joint average (%) arithmetic properties accuracies (Sum and Product classifiers) with DIGIT set of size 1000.

		Commut.	Assoc.	Dist.
1000	$\mathcal{S}$-Gödel	**55.05**	42.05	**37.82**
	$\mathcal{S}$-Product	54.21	**43.53**	37.13
	$\mathcal{R}$-Product	44.00	38.88	31.41
	Łukasiewicz	43.00	39.87	30.87
5000		**91.83**	**89.22**	**86.80**
		89.39	85.56	83.03
		90.87	88.39	83.88
		85.34	80.15	74.67
25000		96.59	95.04	93.24
		92.27	88.79	86.52
		97.48	**96.38**	**93.77**
		95.50	93.67	83.58

Table 17: Joint average (%) arithmetic properties accuracies (Sum and Product classifiers) with DIGIT set of size 5000.

		Commut.	Assoc.	Dist.
1000	$\mathcal{S}$-Gödel	57.72	49.98	42.71
	$\mathcal{S}$-Product	52.75	44.70	38.99
	$\mathcal{R}$-Product	**62.95**	**51.77**	**44.73**
	Łukasiewicz	52.20	44.90	36.79
5000		**91.80**	**89.43**	**87.21**
		90.06	86.81	84.29
		91.40	89.27	84.10
		86.16	81.15	67.68
25000		96.71	95.62	**94.46**
		96.43	94.93	93.52
		97.54	**96.48**	94.11
		94.92	91.50	83.15

Table 18: Joint average (%) arithmetic properties accuracies (Sum and Product classifiers) with DIGIT set of size 25000.

	1000	5000	25000
$\mathcal{S}$-Gödel	60.73	**63.04**	62.91
$\mathcal{R}/\mathcal{S}$-Product	**61.49**	60.92	**62.48**
Łukasiewicz	54.85	55.45	55.10

Table 19: Pipelined operator accuracies (Avg.`Sum` and `Product`) over (noisy) labeled PAIR size 1k with `Digit` model trained on DIGIT sizes 1k, 5k, 10k.

.

	1000	5000	25000
$\mathcal{S}$-Gödel	90.46	94.53	95.41
$\mathcal{R}/\mathcal{S}$-Product	**91.11**	**95.30**	**96.89**
Łukasiewicz	85.85	93.25	95.36

Table 20: Pipelined operator accuracies (Avg.`Sum` and `Product`) over (noisy) labeled PAIR size 25k with `Digit` model trained on DIGIT sizes 1k, 5k, 10k.

.

	1000	5000	25000
$\mathcal{S}$-Gödel	60.71	**63.50**	62.30
$\mathcal{R}/\mathcal{S}$-Product	**62.08**	61.17	**62.58**
Łukasiewicz	55.55	55.69	55.58

Table 21: Coherence accuracies (Avg.`Sum` and `Prod`) over (noisy) labeled PAIR size 1k with `Digit` model trained on DIGIT sizes size 1k, 5k, 25k.

	1000	5000	25000
$\mathcal{S}$-Gödel	88.49	93.76	94.81
$\mathcal{R}/\mathcal{S}$-Product	**90.98**	**95.26**	**96.87**
Łukasiewicz	85.87	93.28	95.42

Table 22: Coherence accuracies (Avg.`Sum` and `Prod`) over (noisy) labeled PAIR size 25k with `Digit` model trained on DIGIT sizes size 1k, 5k, 25k.

		Commut.	Assoc.	Dist.
1000	$\mathcal{S}$-Gödel	56.44	44.65	33.93
	$\mathcal{R}/\mathcal{S}$-Product	**56.61**	**46.55**	**39.06**
	Łukasiewicz	55.02	43.84	21.51
5000		90.03	86.03	82.44
		92.35	**88.85**	**85.06**
		85.36	74.12	58.72
25000		94.39	90.96	87.18
		95.51	**91.92**	**87.61**
		92.87	79.87	74.69

Table 23: Supervised average (%) arithmetic properties accuracies (Sum and Product classifiers) with DIGIT set of size 1000.

		Commut.	Assoc.	Dist.
1000	$\mathcal{S}$-Gödel	**55.90**	**46.12**	**39.94**
	$\mathcal{R}/\mathcal{S}$-Product	54.25	44.59	36.76
	Łukasiewicz	48.67	39.92	27.84
5000		90.64	88.00	85.66
		92.40	**90.09**	**92.61**
		88.78	80.10	88.73
25000		95.86	94.21	92.95
		96.80	**95.55**	**93.26**
		94.33	89.97	89.68

Table 24: Supervised average (%) arithmetic properties accuracies (Sum and Product classifiers) with DIGIT set of size 5000.

		Commut.	Assoc.	Dist.
1000	$\mathcal{S}$-Gödel	53.01	44.89	33.37
	$\mathcal{R}/\mathcal{S}$-Product	**54.91**	**45.15**	**35.38**
	Łukasiewicz	52.86	44.53	29.32
5000		90.47	88.12	85.58
		92.39	**90.38**	**87.87**
		87.48	80.46	78.41
25000		96.04	94.74	93.48
		97.44	**96.79**	**95.28**
		95.95	94.68	92.93

Table 25: Supervised average (%) arithmetic properties accuracies (Sum and Product classifiers) with DIGIT set of size 25000.

Tautologies	$\mathcal{S}$-Prod.	$\mathcal{S}$-Gödel	Łuka.	$\mathcal{R}$-Pro.	$\mathcal{R}$-Gödel
Axiom Schemata					
$P \to (Q \to P)$	0.92	0.79	1	1	1
$(P \to (Q \to R)) \to ((P \to Q) \to (P \to R))$	0.88	0.75	0.96	0.93	1
$(\neg P \to \neg Q) \to (Q \to P)$	0.86	0.75	1	1	0.79
Primitive Propositions					
$(P \vee P) \to P$	0.75	0.75	0.75	1	1
$Q \to (P \vee Q)$	0.92	0.79	1	1	1
$(P \vee Q) \to (Q \vee P)$	0.86	0.75	1	1	1
$(P \vee (Q \vee R)) \to (Q \vee (P \vee R))$	0.91	0.78	1	1	1
$(Q \to R) \to ((P \vee Q) \to (P \vee R))$	0.90	0.76	1	1	1
Law of excluded middle					
$P \vee \neg P$	0.83	0.75	1	0.83	0.75
Law of contradiction					
$\neg(P \wedge \neg P)$	0.83	0.75	1	0.83	0.75
Law of double negation					
$P \leftrightarrow \neg(\neg P)$	0.70	0.75	1	1	1
Principles of transposition					
$(P \leftrightarrow Q) \leftrightarrow (\neg P \leftrightarrow \neg Q)$	0.60	0.66	1	0.59	0.16
$((P \wedge Q) \to R) \leftrightarrow ((P \wedge \neg R) \to \neg Q)$	0.84	0.78	1	0.85	0.65
Laws of tautology					
$P \leftrightarrow (P \wedge P)$	0.69	0.75	0.75	0.5	1
$P \leftrightarrow (P \vee P)$	0.69	0.75	0.75	1	1
Laws of absorption					
$(P \to Q) \leftrightarrow (P \leftrightarrow (P \wedge Q))$	0.66	0.71	0.83	0.66	1
$Q \to (P \leftrightarrow (P \wedge Q))$	0.82	0.75	1	1	1
Assoc., Comm., Dist. laws					
$(P \wedge (Q \vee R)) \leftrightarrow ((P \wedge Q) \vee (P \wedge R))$	0.69	0.72	0.90	0.89	0.72
$(P \vee (Q \wedge R)) \leftrightarrow ((P \vee Q) \wedge (P \vee R))$	0.69	0.72	0.90	0.90	1
De Morgans Laws					
$(P \wedge Q) \leftrightarrow \neg(\neg P \vee \neg Q)$	0.75	0.75	1	1	1
$\neg(P \wedge Q) \leftrightarrow \neg(\neg P \vee \neg Q)$	0.75	0.75	1	1	1
Modus Ponens					
$(P \wedge (P \to Q)) \to Q$	0.86	0.75	1	1	1
Material excluded middle					
$(P \to Q) \vee (Q \to P)$	0.97	0.83	1	1	1

Table 26: Degrees of consistency given by the different logic relaxations under consideration over a representative set of tautologies.