Evaluating Relaxations of Logic for Neural Networks: A Comprehensive Study

Mattia Medina Grespan , **Ashim Gupta** and **Vivek Srikumar**
University of Utah
{mattiamg,ashim,svivek}@cs.utah.edu

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 and Srikumar, 2019; Li *et al.*, 2019; Fischer *et al.*, 2019]. 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 ones are: Product [e.g., Rocktäschel *et al.*, 2015; Li *et al.*, 2019; Asai and Hajishirzi, 2020], Gödel [e.g., Minervini *et al.*, 2017] and Łukasiewicz [e.g., Bach *et al.*, 2017]. 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 their variants. 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 subsume 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], 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 $\mathcal{S}$-logics) 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 knowledge 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], $\mathcal{S}$-Gödel [Minervini *et al.*, 2017] and even a mixture of the $\mathcal{S}$-Gödel and $\mathcal{R}$-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 labeling task, we can represent the fact that the label for an instance 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 $x \in U$, 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 standpoint, 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 that 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, defined as $1 - x + x \cdot y$. We are defining $\mathcal{R}$-Product using its SBL$_\sim$ extension with involutive negation (See Esteva *et al.* [2000]).

In this declarative learning setting, we have the choice of using *any* models (e.g., CNNs) for our predicates, and *any* relaxation of logic. If we only have labeled examples, and we use one of the Product relaxations, we recover the widely used cross-entropy loss [Li *et al.*, 2019; Giannini *et al.*, 2019].

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).

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 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 be 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 under a given relaxation.

Definition 2. *Let `P` be any Boolean formula in predicate logic with a set of atomic predicates $\mathcal{P}$, 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* 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 the tautology* $\mathtt{A} \rightarrow \mathtt{A}$ *using* $\mathcal{S}$*-Product.*

$$[\mathtt{A} \rightarrow \mathtt{A}] = 1 - a + a^2.$$

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

$$\kappa^{\mathcal{S}\text{-}Prod}(\mathtt{A} \rightarrow \mathtt{A}) = \int_0^1 1 - a + a^2 \ \mathrm{d}a = \frac{5}{6} \approx 0.83$$

[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	$\mathcal{S}$-Prod	$\mathcal{S}$-Gödel	Łuka	$\mathcal{R}$-Prod
Axiom Schemata				
$\mathtt{P} \rightarrow (\mathtt{Q} \rightarrow \mathtt{P})$	0.92	0.79	1	1
$(\mathtt{P} \rightarrow (\mathtt{Q} \rightarrow \mathtt{R})) \rightarrow ((\mathtt{P} \rightarrow \mathtt{Q}) \rightarrow (\mathtt{P} \rightarrow \mathtt{R}))$	0.88	0.75	0.96	0.93
$(\neg\mathtt{P} \rightarrow \neg\mathtt{Q}) \rightarrow (\mathtt{Q} \rightarrow \mathtt{P})$	0.86	0.75	1	0.88
Primitive Propositions				
$(\mathtt{P} \vee \mathtt{P}) \rightarrow \mathtt{P}$	0.75	0.75	0.75	0.69
$\mathtt{Q} \rightarrow (\mathtt{P} \vee \mathtt{Q})$	0.92	0.79	1	1
$(\mathtt{P} \vee \mathtt{Q}) \rightarrow (\mathtt{Q} \vee \mathtt{P})$	0.86	0.75	1	1
$(\mathtt{P} \vee (\mathtt{Q} \vee \mathtt{R})) \rightarrow (\mathtt{Q} \vee (\mathtt{P} \vee \mathtt{R}))$	0.91	0.78	1	1
$(\mathtt{Q} \rightarrow \mathtt{R}) \rightarrow ((\mathtt{P} \vee \mathtt{Q}) \rightarrow (\mathtt{P} \vee \mathtt{R}))$	0.90	0.76	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				
$(\mathtt{P} \leftrightarrow \mathtt{Q}) \leftrightarrow (\neg\mathtt{P} \leftrightarrow \neg\mathtt{Q})$	0.61	0.67	1	0.59
Laws of tautology				
$\mathtt{P} \leftrightarrow (\mathtt{P} \wedge \mathtt{P})$	0.69	0.75	0.75	0.50
$\mathtt{P} \leftrightarrow (\mathtt{P} \vee \mathtt{P})$	0.69	0.75	0.75	0.69
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

Table 2: Consistencies of a (subset) of representative set of tautologies under different logic relaxations. We see here that the $\mathcal{R}$-Product and Łukasiewicz relaxations are generally more consistent, suggesting that they are preferable to the other two relaxations. Other tautologies we examined show the same trends.

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 the 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.

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 $\kappa_S^{\mathcal{S}\text{-Gödel}}(\mathtt{P}) = 0.75$ and under the Product relaxation, we have $\kappa_S^{\mathcal{S}\text{-Prod}}(\mathtt{P}) \approx 0.748$. 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* $\kappa_S^{\mathcal{S}\text{-}Prod}(A) = 1 - \frac{2}{2^n} + \frac{3}{3^n} - \frac{2}{4^n} + \frac{1}{5^n}$.

Proof. By induction over the size of the conjunction n. □

We see that, as $n \rightarrow \infty$, 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.[4]

5.1 Recognizing digits and arithmetic operations

These experiments build upon our running example from §2. We seek to categorize handwritten 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, 1998] 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 the 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. 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:

$$-\left[\left(\bigwedge_{(x,y)\in D} \texttt{Digit}(x,y)\right) \wedge \left(\bigwedge_{PAIR} \texttt{Sum Coherence}\right) \wedge \left(\bigwedge_{PAIR} \texttt{Product Coherence}\right)\right] \quad (6)$$

[4]Our PyTorch [Paszke *et al.*, 2019] code is archived at https://github.com/utahnlp/neural-logic

	1000	5000	25000
$\mathcal{S}$-Gödel	95.0 (0.3)	97.4 (0.1)	97.7 (0.1)
$\mathcal{S}$-Product	95.1 (0.1)	98.2 (0.0)	99.0 (0.0)
$\mathcal{R}$-Product	**96.3** (0.1)	**98.4** (0.1)	**99.2** (0.0)
Łukasiewicz	95.8 (0.1)	98.0 (0.1)	99.1 (0.0)

Table 3: `Digit` accuracies (and standard deviations) from jointly training `Digit` on DIGIT sizes 1k, 5k, 25k and operators (`Sum`,`Prod`) on PAIR size 5k.

	1000	5000	25000
$\mathcal{S}$-Gödel	87.3 (0.5)	**91.1** (0.1)	91.5 (0.0)
$\mathcal{S}$-Product	76.9 (0.8)	88.6 (0.6)	90.1 (0.0)
$\mathcal{R}$-Product	**88.0** (0.3)	90.8 (0.3)	**91.8** (0.0)
Łukasiewicz	75.9 (3.1)	84.5 (2.8)	82.3 (0.3)

Table 4: Average of `Sum` and `Product` accuracies (and standard deviations) from jointly training `Digit` on DIGIT sizes 1k, 5k, 25k and operators on PAIR size 5k.

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 of the accuracy of the `Digit` classifier and the coherences of the other two.

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 $p < 0.05$. The accuracies are averages from three runs with different seeds along with the standard deviation.) 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 $\mathcal{S}$-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.

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

	1000	5000	25000
$\mathcal{S}$-Gödel	86.4 (0.6)	91.4 (0.1)	91.9 (0.1)
$\mathcal{S}$-Product	79.0 (0.7)	89.3 (0.5)	90.3 (0.0)
$\mathcal{R}$-Product	**89.1** (0.3)	**91.5** (0.4)	**92.1** (0.1)
Łukasiewicz	77.4 (3.0)	85.4 (2.7)	82.6 (0.2)

Table 5: Average of `Sum` and `Prod` Coherence accuracies (and standard deviations) from jointly training `Digit` on DIGIT sizes 1k, 5k, 25k and operators on PAIR size 5k.

	1000	5000	25000
$\mathcal{S}$-Gödel	95.2	97.6	97.59
$\mathcal{R}/\mathcal{S}$-Product	**96.7**	**98.4**	**99.12**
Łukasiewicz	96.5	98.5	98.76

Table 6: Pipelined `Digit` accuracies trained on DIGIT sizes 1k, 5k, 25k. Standard deviation is 0.0 for every entry.

	1000	5000	25000
$\mathcal{S}$-Gödel	88.3 (0.3)	90.9 (0.1)	91.3 (0.1)
$\mathcal{R}/\mathcal{S}$-Product	**89.7** (0.1)	**92.2** (0.0)	**93.0** (0.0)
Łukasiewicz	83.0 (1.9)	86.9 (3.6)	88.6 (0.4)

Table 7: Average of Pipelined `Sum` and `Product` accuracies (and standard deviations) trained on PAIR size 5k (noisy) labeled with `Digit` model trained on DIGIT sizes 1k, 5k, 25k.

	1000	5000	25000
$\mathcal{S}$-Gödel	86.7 (0.3)	90.6 (0.1)	91.3 (0.0)
$\mathcal{R}/\mathcal{S}$-Product	**89.9** (0.2)	**92.5** (0.0)	**92.8** (0.0)
Łukasiewicz	83.5 (1.8)	87.2 (3.5)	88.7 (0.4)

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

an optimization learning problem via the logic relaxations. However, in the pipeline, the coherence constraints are not explicitly involved in the 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 the 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.

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 the 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 §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 D, and a set C of k constraints for each word position, we can write the conjunction of all constraints as:

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

We can now use this Boolean formula to state the goal of learning as finding the model parameters of a neural network that 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`.

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 the 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 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

% 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	70.81	79.71	**80.14**	76.80
100% + C	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 include 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 a 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 is that Łukasiewicz t-norm in our 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 in the case of Sikka *et al.* [2020] and Giannini *et al.* [2019].

Our framework 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], who employ constraints similar to our coherence constraints. Of course, the goal of this work is to theoretically, and empirically investigate which relaxation is best suited for use in such problems. To our knowledge, none of the previously mentioned works analyze the choice of the relaxation they use.

Similar studies to this paper are that of Evans and Grefenstette [2018] and van Krieken *et al.* [2020]. The former, includes comparison in performance of Łukasiewicz, Product, and Gödel t-norm operators used to induce differentiable functions from definitive clauses in neural program synthesis. The latter, perhaps closer to our approach, provides a general theoretical and empirical analysis for different t-norm relaxations. However, there are two key differences with this paper: (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. We also studied the importance of the loss gradient signal in gradient-based learning, and hypothesise that a less consistent t-norm would perform worse at characterizing the truth of a Boolean statement than a more consistent one, and as a result, would correspond to poorer empirical performance.

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 offer support for gradient-based learning. We also present empirical studies on two tasks using 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 *ACL*, 2020.

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, 2017.

Francesc Esteva, Lluís Godo, Petr Hájek, and Mirko Navara. Residuated Fuzzy Logics with an Involutive Negation. *Archive for Mathematical Logic*, 39, 2000.

Richard Evans and Edward Grefenstette. Learning Explanatory Rules from Noisy Data. *Journal of Artificial Intelligence Research (JAIR)*, 61, 2018.

Marc Fischer, Mislav Balunovic, Dana Drachsler-Cohen, Timon Gehr, Ce Zhang, and Martin Vechev. DL2: Training and Querying Neural Networks with Logic. In *ICML*, 2019.

Francesco Giannini, Giuseppe Marra, Michelangelo Diligenti, Marco Maggini, and Marco Gori. On the Relation Between Loss Functions and T-Norms. In *ILP*, 2019.

Edward Grefenstette. Towards a Formal Distributional Semantics: Simulating Logical Calculi with Tensors. In **SEM@NAACL*, 2013.

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

Yann LeCun. The MNIST Database of Handwritten Digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Tao Li and Vivek Srikumar. Augmenting Neural Networks with First-order Logic. In *ACL*, 2019.

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 *ACL*, 2020.

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. DeepProbLog: Neural Probabilistic Logic Programming. In *NIPS*, 2018.

Karl Menger. Statistical Metrics. *Proceedings of the National Academy of Sciences of the United States of America*, 28, 1942.

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

Yatin Nandwani, Abhishek Pathak, and Parag Singla. 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 *NeurIPS*, 2019.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, 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.

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. *arXiv preprint arXiv:2003.07344*, 2020.

Emile van Krieken, Erman Acar, and Frank van Harmelen. Analyzing Differentiable Fuzzy Implications. In *KR*, 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 *ICML*, 2018.