Augmenting Neural Networks with First-order Logic

Tao Li
University of Utah
tli@cs.utah.edu

Vivek Srikumar
University of Utah
svivek@cs.utah.edu

Abstract

Today, the dominant paradigm for training neural networks involves minimizing task loss on a large dataset. Using world knowledge to inform a model, and yet retain the ability to perform end-to-end training remains an open question. In this paper, we present a novel framework for introducing declarative knowledge to neural network architectures in order to guide training and prediction. Our framework systematically compiles logical statements into computation graphs that augment a neural network without extra learnable parameters or manual redesign. We evaluate our modeling strategy on three tasks: machine comprehension, natural language inference, and text chunking. Our experiments show that knowledge-augmented networks can strongly improve over baselines, especially in low-data regimes.

1 Introduction

Neural models demonstrate remarkable predictive performance across a broad spectrum of NLP tasks: e.g., natural language inference (Parikh et al., 2016), machine comprehension (Seo et al., 2017), machine translation (Bahdanau et al., 2015), and summarization (Rush et al., 2015). These successes can be attributed to their ability to learn robust representations from data. However, such end-to-end training demands a large number of training examples; for example, training a typical network for machine translation may require millions of sentence pairs (e.g. Luong et al., 2015). The difficulties and expense of curating large amounts of annotated data are well understood and, consequently, massive datasets may not be available for new tasks, domains or languages.

In this paper, we argue that we can combat the data hungriness of neural networks by taking advantage of domain knowledge expressed as first-order logic. As an example, consider the task of reading comprehension, where the goal is to answer a question based on a paragraph of text (Fig. 1). Attention-driven models such as BiDAF (Seo et al., 2017) learn to align words in the question with words in the text as an intermediate step towards identifying the answer. While alignments (e.g. *author* to *writing*) can be learned from data, we argue that models can reduce their data dependence if they were guided by easily stated rules such as: *Prefer aligning phrases that are marked as similar according to an external resource, e.g., ConceptNet (Liu and Singh, 2004).* If such declaratively stated rules can be incorporated into training neural networks, then they can provide the inductive bias that can reduce data dependence for training.

Paragraph: Gaius Julius Caesar (July 100 BC – 15 March 44 BC), Roman general, statesman, Consul and notable author of Latin prose, played a critical role in the events that led to the demise of the Roman Republic and the rise of the Roman Empire through his various military campaigns.

Question: Which Roman general is known for writing prose?

Figure 1: An example of reading comprehension that illustrates alignments/attention. In this paper, we consider the problem of incorporating external knowledge about such alignments into training neural networks.

That general neural networks can represent such Boolean functions is known and has been studied both from the theoretical and empirical perspectives (e.g. Maass et al., 1994; Anthony, 2003; Pan and Srikumar, 2016). Recently, Hu et al. (2016) exploit this property to train a neural network to mimic a teacher network that uses structured rules. In this paper, we seek to directly incorporate such structured knowledge into a neural network architecture without substantial changes to the training methods. We focus on three questions:

1. Can we integrate declarative rules with end-to-end neural network training?

2. Can such rules help ease the need for data?

3. How does incorporating domain expertise compare against large training resources powered by pre-trained representations?

The first question poses the key technical challenge we address in this paper. On one hand, we wish to guide training and prediction with neural networks using logic, which is non-differentiable. On the other hand, we seek to retain the advantages of gradient-based learning without having to redesign the training scheme. To this end, we propose a framework that allows us to systematically augment an *existing* network architecture using constraints about its nodes by deterministically converting rules into differentiable computation graphs. To allow for the possibility of such rules being incorrect, our framework is designed to admit soft constraints from the ground up. Our framework is compatible with off-the-shelf neural networks without extensive redesign or any additional trainable parameters.

To address the second and the third questions, we empirically evaluate our framework on three tasks: machine comprehension, natural language inference, and text chunking. In each case, we use a general off-the-shelf model for the task, and study the impact of simple logical constraints on observed neurons (e.g., attention) for different data sizes. We show that our framework can successfully improve an existing neural design, especially when the number of training examples is limited.

In summary, our contributions are:

1. We introduce a new framework for incorporating first-order logic rules into neural network design in order to guide both training and prediction.

2. We evaluate our approach on three different NLP tasks: machine comprehension, textual entailment, and text chunking. We show that augmented models lead to large performance gains in the low training data regimes.[1]

2 Problem Setup

In this section, we will introduce the notation and assumptions that form the basis of our formalism for constraining neural networks.

Neural networks are directed acyclic computation graphs $G = (V, E)$, consisting of nodes (i.e. neurons) V and weighted directed edges E that represent information flow. Although not all neurons have explicitly grounded meanings, some nodes indeed can be endowed with semantics tied to the task. Node semantics may be assigned during model design (e.g. attention), or incidentally discovered in post hoc analysis (e.g., Le et al., 2012; Radford et al., 2017, and others). In either case, our goal is to augment a neural network with such **named neurons** using declarative rules.

The use of logic to represent domain knowledge has a rich history in AI (e.g. Russell and Norvig, 2016). In this work, to capture such knowledge, we will primarily focus on conditional statements of the form $L \rightarrow R$, where the expression L is the antecedent (or the left-hand side) that can be conjunctions or disjunctions of literals, and R is the consequent (or the right-hand side) that consists of a single literal. Note that such rules include Horn clauses and their generalizations, which are well studied in the knowledge representation and logic programming communities (e.g. Chandra and Harel, 1985).

Integrating rules with neural networks presents three difficulties. First, we need a mapping between the predicates in the rules and nodes in the computation graph. Second, logic is not differentiable; we need an encoding of logic that admits training using gradient based methods. Finally, computation graphs are acyclic, but user-defined rules may introduce cyclic dependencies between the nodes. Let us look at these issues in order.

As mentioned before, we will assume named neurons are given. And by associating predicates with such nodes that are endowed with symbolic meaning, we can introduce domain knowledge about a problem in terms of these predicates. In the rest of the paper, we will use lower cased letters (e.g., a_i, b_j) to denote nodes in a computation graph, and upper cased letters (e.g., A_i, B_j) for predicates associated with them.

To deal with the non-differentiablity of logic, we will treat the post-activation value of a named neuron as the degree to which the associated predicate is true. In §3, we will look at methods

[1] The code used for our experiments is archived here: https://github.com/utahnlp/layer_augmentation

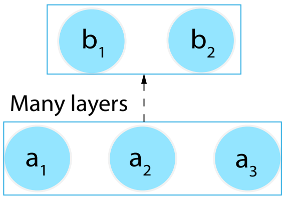

Figure 2: An example computation graph. The statement $A_1 \wedge B_1 \rightarrow A_2 \wedge B_2$ is cyclic with respect to the graph. On the other hand, the statement $A_1 \wedge A_2 \rightarrow B_1 \wedge B_2$ is acyclic.

for compiling conditional statements into differentiable statements that augment a given network.

Cyclicity of Constraints Since we will augment computation graphs with compiled conditional forms, we should be careful to avoid creating cycles. To formalize this, let us define cyclicity of conditional statements with respect to a neural network.

Given two nodes a and b in a computation graph, we say that the node a is *upstream* of node b if there is a directed path from a to b in the graph.

Definition 1 (Cyclic and Acyclic Implications)**.** *Let G be a computation graph. An implicative statement $L \rightarrow R$ is* cyclic *with respect to G if, for any literal $R_i \in R$, the node r_i associated with it is upstream of the node l_j associated with some literal $L_j \in L$. An implicative statement is* acyclic *if it is not cyclic.*

Fig. 2 and its caption gives examples of cyclic and acyclic implications. A cyclic statement sometimes can be converted to an equivalent acyclic statement by constructing its contrapositive. For example, the constraint $B_1 \rightarrow A_1$ is equivalent to $\neg A_1 \rightarrow \neg B_1$. While the former is cyclic, the later is acyclic. Generally, we can assume that we have acyclic implications.[2]

3 A Framework for Augmenting Neural Networks with Constraints

To create constraint-aware neural networks, we will extend the computation graph of an existing network with additional edges defined by constraints. In §3.1, we will focus on the case where the antecedent is conjunctive/disjunctive and the consequent is a single literal. In §3.2, we will cover more general antecedents.

3.1 Constraints Beget Distance Functions

Given a computation graph, suppose we have a acyclic conditional statement: $Z \rightarrow Y$, where Z is a conjunction or a disjunction of literals and Y is a single literal. We define the neuron associated with Y to be $y = g(\mathbf{W}\mathbf{x})$, where g denotes an activation function, $\mathbf{W}$ are network parameters, $\mathbf{x}$ is the immediate input to y. Further, let the vector $\mathbf{z}$ represent the neurons associated with the predicates in Z. While the nodes $\mathbf{z}$ need to be named neurons, the immediate input $\mathbf{x}$ need not necessarily have symbolic meaning.

Constrained Neural Layers Our goal is to augment the computation of y so that whenever Z is true, the pre-activated value of y increases if the literal Y is not negated (and decreases if it is). To do so, we define a *constrained neural layer* as

$$y = g\left(\mathbf{W}\mathbf{x} + \rho d\left(\mathbf{z}\right)\right). \qquad (1)$$

Here, we will refer to the function d as the *distance function* that captures, in a differentiable way, whether the antecedent of the implication holds. The importance of the entire constraint is decided by a real-valued hyper-parameter $\rho \geq 0$.

The definition of the constrained neural layer says that, by compiling an implicative statement into a distance function, we can regulate the preactivation scores of the downstream neurons based on the states of upstream ones.

Designing the distance function The key consideration in the compilation step is the choice of an appropriate distance function for logical statements. The ideal distance function we seek is the indicator for the statement Z:

$$d_{ideal}(\mathbf{z}) = \begin{cases} 1, & \text{if } Z \text{ holds,} \\ 0, & \text{otherwise.} \end{cases}$$

However, since the function d_{ideal} is not differentiable, we need smooth surrogates.

In the rest of this paper, we will define and use distance functions that are inspired by probabilistic soft logic (c.f. Klement et al., 2013) and its use of the Łukasiewicz T-norm and T-conorm to define a soft version of conjunctions and disjunctions.[3]

Table 1 summarizes distance functions corresponding to conjunctions and disjunctions. In all

[2] As we will see in §3.3, the contrapositive does not always help because we may end up with a complex right hand side that we can not yet compile into the computation graph.

[3] The definitions of the distance functions here as surrogates for the non-differentiable d_{ideal} is reminiscent of the use of hinge loss as a surrogate for the zero-one loss. In both cases, other surrogates are possible.

Antecedent	Distance $d(\mathbf{z})$
$\bigwedge_i Z_i$	$\max(0, \sum_i z_i - \lvert Z\rvert + 1)$
$\bigvee_i Z_i$	$\min(1, \sum_i z_i)$
$\neg \bigvee_i Z_i$	$\max(0, 1 - \sum_i z_i)$
$\neg \bigwedge_i Z_i$	$\min(1, N - \sum_i z_i)$

Table 1: Distance functions designed using the Łukasiewicz T-norm. Here, $|Z|$ is the number of antecedent literals. z_i's are upstream neurons associated with literals Z_i's.

cases, recall that the z_i's are the states of neurons and are assumed to be in the range $[0, 1]$. Examining the table, we see that with a conjunctive antecedent (first row), the distance becomes zero if even one of the conjuncts is false. For a disjunctive antecedent (second row), the distance becomes zero only when all the disjuncts are false; otherwise, it increases as the disjuncts become more likely to be true.

Negating Predicates Both the antecedent (the Z's) and the consequent (Y) could contain negated predicates. We will consider these separately.

For any negated antecedent predicate, we modify the distance function by substituting the corresponding z_i with $1 - z_i$ in Table 1. The last two rows of the table list out two special cases, where the entire antecedents are negated, and can be derived from the first two rows.

To negate consequent Y, we need to reduce the pre-activation score of neuron y. To achieve this, we can simply negate the entire distance function.

Scaling factor ρ In Eq. 1, the distance function serves to promote or inhibit the value of downstream neuron. The extent is controlled by the scaling factor ρ. For instance, with $\rho = +\infty$, the pre-activation score of the downstream neuron is dominated by the distance function. In this case, we have a hard constraint. In contrast, with a small ρ, the output state depends on both the $\mathbf{Wx}$ and the distance function. In this case, the *soft* constraint serves more as a suggestion. Ultimately, the network parameters might overrule the constraint. We will see an example in §4 where noisy constraint prefers small ρ.

3.2 General Boolean Antecedents

So far, we exclusively focused on conditional statements with either conjunctive or disjunctive antecedents. In this section, we will consider general antecedents.

As an illustrative example, suppose we have an antecedent $(\neg A \vee B) \wedge (C \vee D)$. By introducing auxiliary variables, we can convert it into the conjunctive form $P \wedge Q$, where $(\neg A \vee B) \leftrightarrow P$ and $(C \vee D) \leftrightarrow Q$. To perform such operation, we need to: (1) introduce auxiliary neurons associated with the auxiliary predicates P and Q, and, (2) define these neurons to be exclusively determined by the biconditional constraint.

To be consistent in terminology, when considering biconditional statement $(\neg A \vee B) \leftrightarrow P$, we will call the auxiliary literal P the consequent, and the original literals A and B the antecedents.

Because the implication is bidirectional in biconditional statement, it violates our acyclicity requirement in §3.1. However, since the auxiliary neuron state does not depend on any other nodes, we can still create an acyclic sub-graph by defining the new node to be the distance function itself.

Constrained Auxiliary Layers With a biconditional statement $Z \leftrightarrow Y$, where Y is an auxiliary literal, we define a *constrained auxiliary layer* as

$$y = d(\mathbf{z}) \tag{2}$$

where d is the distance function for the statement, $\mathbf{z}$ are upstream neurons associated with Z, y is the downstream neuron associated with Y. Note that, compared to Eq. 1, we do not need activation function since the distance, which is in $[0, 1]$, can be interpreted as producing normalized scores.

Note that this construction only applies to auxiliary predicates in biconditional statements. The advantage of this layer definition is that we can use the same distance functions as before (i.e., Table 1). Furthermore, the same design considerations in §3.1 still apply here, including how to negate the left and right hand sides.

Constructing augmented networks To complete the modeling framework, we summarize the workflow needed to construct an augmented neural network given a conditional statement and a computation graph: (1) Convert the antecedent into a conjunctive or a disjunctive normal form if necessary. (2) Convert the conjunctive/disjunctive antecedent into distance functions using Table 1 (with appropriate corrections for negations). (3) Use the distance functions to construct constrained layers and/or auxiliary layers to augment the computation graph by replacing the original layer with constrained one. (4) Finally, use the augmented network for end-to-end training and inference. We will see complete examples in §4.

3.3 Discussion

Not only does our design not add any more trainable parameters to the existing network, it also admits efficient implementation with modern neural network libraries.

When posing multiple constraints on the same downstream neuron, there could be combinatorial conflicts. In this case, our framework relies on the base network to handle the consistency issue. In practice, we found that summing the constrained pre-activation scores for a neuron is a good heuristic (as we will see in §4.3).

For a conjunctive consequent, we can decompose it into multiple individual constraints. That is equivalent to constraining downstream nodes independently. Handling more complex consequents is a direction of future research.

4 Experiments

In this section, we will answer the research questions raised in §1 by focusing on the effectiveness of our augmentation framework. Specifically, we will explore three types of constraints by augmenting: 1) intermediate decisions (i.e. attentions); 2) output decisions constrained by intermediate states; 3) output decisions constrained using label dependencies.

To this end, we instantiate our framework on three tasks: machine comprehension, natural language inference, and text chunking. Across all experiments, our goal is to study the modeling flexibility of our framework and its ability to improve performance, especially with decreasing amounts of training data.

To study low data regimes, our augmented networks are trained using varying amounts of training data to see how performances vary from baselines. For detailed model setup, please refer to the appendices.

4.1 Machine Comprehension

Attention is a widely used intermediate state in several recent neural models. To explore the augmentation over such neurons, we focus on attention-based machine comprehension models on SQuAD (v1.1) dataset (Rajpurkar et al., 2016). We seek to use word relatedness from external resources (i.e., ConceptNet) to guide alignments, and thus to improve model performance.

Model We base our framework on two models: BiDAF (Seo et al., 2017) and its ELMo-augmented variant (Peters et al., 2018). Here, we provide an abstraction of the two models which our framework will operate on:

$$\mathbf{p}, \mathbf{q} = \text{encoder}(p), \text{encoder}(q) \tag{3}$$

$$\overleftarrow{\mathbf{a}}, \overrightarrow{\mathbf{a}} = \sigma(\text{layers}(\mathbf{p}, \mathbf{q})) \tag{4}$$

$$\mathbf{y}, \mathbf{z} = \sigma(\text{layers}(\mathbf{p}, \mathbf{q}, \overleftarrow{\mathbf{a}}, \overrightarrow{\mathbf{a}})) \tag{5}$$

where p and q are the paragraph and query respectively, σ refers to the softmax activation, $\overleftarrow{\mathbf{a}}$ and $\overrightarrow{\mathbf{a}}$ are the bidirectional attentions from q to p and vice versa, $\mathbf{y}$ and $\mathbf{z}$ are the probabilities of answer boundaries. All other aspects are abstracted as $encoder$ and $layers$.

Augmentation By construction of the attention neurons, we expect that related words should be aligned. In a knowledge-driven approach, we can use ConceptNet to guide the attention values in the model in Eq. 4.

We consider two rules to illustrate the flexibility of our framework. Both statements are in first-order logic that are dynamically grounded to the computation graph for a particular paragraph and query. First, we define the following predicates:

$K_{i,j}$ word p_i is related to word q_j in ConceptNet via edges {*Synonym*, *DistinctFrom*, *IsA*, *Related*}.

$\overleftarrow{A}_{i,j}$ unconstrained model decision that word q_j best matches to word p_i.

$\overleftarrow{A}'_{i,j}$ constrained model decision for the above alignment.

Using these predicates, we will study the impact of the following two rules, defined over a set C of content words in p and q:

R_1: $\forall i, j \in C,\ K_{i,j} \rightarrow \overleftarrow{A}'_{i,j}$.

R_2: $\forall i, j \in C,\ K_{i,j} \wedge \overleftarrow{A}_{i,j} \rightarrow \overleftarrow{A}'_{i,j}$.

The rule R_1 says that two words should be aligned if they are related. Interestingly, compiling this statement using the distance functions in Table 1 is essentially the same as adding word relatedness as a static feature. The rule R_2 is more conservative as it also depends on the unconstrained

%Train	BiDAF	$+R_1$	$+R_2$	+ELMo	+ELMo,R_1
10%	57.5	**61.5**	60.7	71.8	**73.0**
20%	65.7	**67.2**	66.6	76.9	**77.7**
40%	70.6	**72.6**	71.9	80.3	**80.9**
100%	75.7	**77.4**	77.0	83.9	**84.1**

Table 2: Impact of constraints on BiDAF. Each score represents the average span F_1 on our test set (i.e. official dev set) among 3 random runs. Constrained models and ELMo models are built on top of BiDAF. We set $\rho = 2$ for both R_1 and R_2 across all percentages.

model decision. In both cases, since $K_{i,j}$ does not map to a node in the network, we have to create a new node $k_{i,j}$ whose value is determined using ConceptNet, as illustrated in Fig. 3.

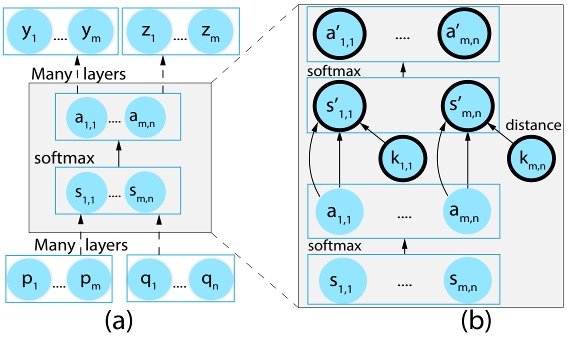

Figure 3: (a) The computation graph of BiDAF where attention directions are obmitted. (b) The augmented graph on attention layer using R_2. Bold circles are extra neurons introduced. Constrained attentions and scores are $\mathbf{a}'$ and $\mathbf{s}'$ respectively. In the augmented model, graph (b) replaces the shaded part in (a).

Can our framework use rules over named neurons to improve model performance? The answer is yes. We experiment with rules R_1 and R_2 on incrementally larger training data. Performances are reported in Table 2 with comparison with baselines. We see that our framework can indeed use logic to inform model learning and prediction without any extra trainable parameters needed. The improvement is particularly strong with small training sets. With more data, neural models are less reliant on external information. As a result, the improvement with larger datasets is smaller.

How does it compare to pretrained encoders? Pretrained encoders (e.g. ELMo and BERT (Devlin et al., 2018)) improve neural models with improved representations, while our framework augments the graph using first-order logic. It is important to study the interplay of these two orthogonal directions. We can see in Table 2, our augmented model consistently outperforms baseline even with the presence of ELMo embeddings.

Does the conservative constraint R_2 help? We explored two options to incorporate word relatedness; one is a straightforward constraint (i.e. R_1), another is its conservative variant (i.e. R_2). It is a design choice as to which to use. Clearly in Table 2, constraint R_1 consistently outperforms its conservative alternative R_2, even though R_2 is better than baseline. In the next task, we will see an example where a conservative constraint performs better with large training data.

4.2 Natural Language Inference

Unlike in the machine comprehension task, here we explore logic rules that bridge attention neurons and output neurons. We use the SNLI dataset (Bowman et al., 2015), and base our framework on a variant of the decomposable attention (DAtt, Parikh et al., 2016) model where we replace its projection encoder with bidirectional LSTM (namely L-DAtt).

Model Again, we abstract the pipeline of L-DAtt model, only focusing on layers which our framework works on. Given a premise p and a hypothesis h, we summarize the model as:

$$\mathbf{p}, \mathbf{h} = \text{encoder}(p), \text{encoder}(h) \tag{6}$$

$$\overleftarrow{\mathbf{a}}, \overrightarrow{\mathbf{a}} = \sigma(\text{layers}(\mathbf{p}, \mathbf{h})) \tag{7}$$

$$\mathbf{y} = \sigma(\text{layers}(\mathbf{p}, \mathbf{h}, \overleftarrow{\mathbf{a}}, \overrightarrow{\mathbf{a}})) \tag{8}$$

Here, σ is the softmax activation, $\overleftarrow{\mathbf{a}}$ and $\overrightarrow{\mathbf{a}}$ are bidirectional attentions, $\mathbf{y}$ are probabilities for labels *Entailment*, *Contradiction*, and *Neutral*.

Augmentation We will borrow the predicate notation defined in the machine comprehension task (§4.1), and ground them on premise and hypothesis words, e.g. $K_{i,j}$ now denotes the relatedness between premise word p_i and hypothesis word h_j. In addition, we define the predicate Y_l to indicate that the label is l. As in §4.1, we define two rules governing attention:

N_1: $\forall i, j \in C,\ K_{i,j} \rightarrow A'_{i,j}$.

N_2: $\forall i, j \in C,\ K_{i,j} \wedge A_{i,j} \rightarrow A'_{i,j}$.

where C is the set of content words. Note that the two constraints apply to both attention directions.

Intuitively, if a hypothesis content word is not aligned, then the prediction should not be *Entailment*. To use this knowledge, we define the following rule:

$$
\begin{aligned}
N_3\text{:}\quad & Z_1 \vee Z_2 \rightarrow \neg Y_{\text{Entail}}, \text{ where} \\
& \exists j \in C,\ \neg\left(\exists i \in C,\ \overleftarrow{A}'_{i,j}\right) \leftrightarrow Z_1, \\
& \exists j \in C,\ \neg\left(\exists i \in C,\ \overrightarrow{A}'_{i,j}\right) \leftrightarrow Z_2.
\end{aligned}
$$

where Z_1 and Z_2 are auxiliary predicates tied to the Y_{Entail} predicate. The details of N_3 are illustrated in Fig. 4.

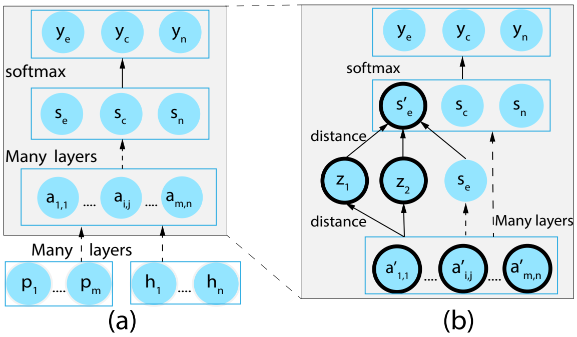

Figure 4: (a) The computation graph of the L-DAtt model (attention directions obmitted). (b) The augmented graph on the *Entail* label using N_3. Bold circles are extra neurons introduced. Unconstrained pre-activation scores are s while s'_e is the constrained score on *Entail*. Intermediate neurons are z_1 and z_2. constrained attentions $\mathbf{a}'$ are constructed using N_1 or N_2. In our augmented model, the graph (b) replaces the shaded part in (a).

How does our framework perform with large training data? The SNLI dataset is a large dataset with over half-million examples. We train our models using incrementally larger percentages of data and report the average performance in Table 3. Similar to §4.1, we observe strong improvements from augmented models trained on small percentages ($\leq$10%) of data. The straightforward constraint N_1 performs strongly with $\leq$2% data while its conservative alternative N_2 works better with a larger set. However, with full dataset, our augmented models perform only on par with baseline even with lowered scaling factor ρ. These observations suggest that if a large dataset is available, it may be better to believe the data, but with smaller datasets, constraints can provide useful inductive bias for the models.

Are noisy constraints helpful? It is not always easy to state a constraint that all examples satisfy. Comparing N_2 and N_3, we see that N_3 performed even worse than baseline, which suggests it contains noise. In fact, we found a significant amount of counter examples to N_3 during preliminary analysis. Yet, even a noisy rule can improve model performance with $\leq$10% data. The same observation holds for N_1, which suggests conservative constraints could be a way to deal with noise. Finally, by comparing N_2 and $N_{2,3}$, we find that the good constraint N_2 can not just augment the network, but also amplify the noise in N_3 when they are combined. This results in degrading performance in the $N_{2,3}$ column starting from 5% of the data, much earlier than using N_3 alone.

%Train	L-DAtt	+N_1	+N_2	+N_3	+$N_{2,3}$
1%	61.2	**64.9**	63.9	62.5	64.3
2%	66.5	**70.5**	69.8	67.9	70.2
5%	73.4	76.2	**76.6**	74.0	76.4
10%	78.9	80.1	**80.4**	79.3	80.3
100%	**87.1**	86.9	**87.1**	87.0	86.9

Table 3: Impact of constraints on L-DAtt network. Each score represents the average accuracy on SNLI test set among 3 random runs. For both N_1 and N_2, we set $\rho = (8, 8, 8, 8, 4)$ for the five different percentages. For the noisy constraint N_3, $\rho = (2, 2, 1, 1, 1)$.

4.3 Text Chunking

Attention layers are a modeling choice that do not always exist in all networks. To illustrate that our framework is not necessarily grounded to attention, we turn to an application where we use knowledge about the output space to constrain predictions. We focus on the sequence labeling task of text chunking using the CoNLL2000 dataset (Tjong Kim Sang and Buchholz, 2000). In such sequence tagging tasks, global inference is widely used, e.g., BiLSTM-CRF (Huang et al., 2015). Our framework, on the other hand, aims to promote local decisions. To explore the interplay of global model and local decision augmentation, we will combine CRF with our framework.

Model Our baseline is a BiLSTM tagger:

$$\mathbf{x} = \text{BiLSTM}(x) \tag{9}$$

$$\mathbf{y} = \sigma(\text{linear}(\mathbf{x})) \tag{10}$$

where x is the input sentence, σ is softmax, $\mathbf{y}$ are the output probabilities of BIO tags.

Augmentation We define the following predicates for input and output neurons:

%Train	BiLSTM	+CRF	+$C_{1:5}$	+CRF,$C_{1:5}$
5%	87.2	86.6	**88.9**	88.6
10%	89.1	88.8	**90.7**	90.6
20%	90.9	90.8	**92.1**	**92.1**
40%	92.5	92.5	93.4	**93.5**
100%	94.1	94.4	94.8	**95.0**

Table 4: Impact of constraints on BiLSTM tagger. Each score represents the average accuracy on test set of 3 random runs. The columns of +CRF, +$C_{1:5}$, and +CRF,$C_{1:5}$ are on top of the BiLSTM baseline. For $C_{1:4}$, $\rho = 4$ for all percentages. For C_5, $\rho = 16$.

$Y_{t,l}$ The unconstrained decision that t^{th} word has label l.

$Y'_{t,l}$ The constrained decision that t^{th} word has label l.

N_t The t^{th} word is a noun.

Then we can write rules for pairwise label dependency. For instance, if word t has B/I- tag for a certain label, word t+1 can not have an I- tag with a different label.

C_1: $\forall t,\ Y_{t,\text{B-VP}} \rightarrow \neg Y'_{t+1,\text{I-NP}}$

C_2: $\forall t,\ Y_{t,\text{I-NP}} \rightarrow \neg Y'_{t+1,\text{I-VP}}$

C_3: $\forall t,\ Y_{t,\text{I-VP}} \rightarrow \neg Y'_{t+1,\text{I-NP}}$

C_4: $\forall t,\ Y_{t,\text{B-PP}} \rightarrow \neg Y'_{t+1,\text{I-VP}}$

Our second set of rules are also intuitive: A noun should not have non-NP label.

C_5: $\forall t, N_t \rightarrow \bigwedge_{l \in \{\text{B-VP,I-VP,B-PP,I-PP}\}} \neg Y'_{t,l}$

While all above rules can be applied as hard constraints in the output space, our framework provides a differentiable way to inform the model during training and prediction.

How does local augmentation compare with global inference? We report performances in Table 4. While a first-order Markov model (e.g., the BiLSTM-CRF) can learn pairwise constraints such as $C_{1:4}$, we see that our framework can better inform the model. Interestingly, the CRF model performed even worse than the baseline with ≤40% data. This suggests that global inference relies on more training examples to learn its scoring function. In contrast, our constrained models performed strongly even with small training sets. And by combining these two orthogonal methods, our locally augmented CRF performed the best with full data.

5 Related Work and Discussion

Artificial Neural Networks and Logic Our work is related to neural-symbolic learning (e.g. Besold et al., 2017) which seeks to integrate neural networks with symbolic knowledge. For example, Cingillioglu and Russo (2019) proposed neural models that multi-hop logical reasoning.

KBANN (Towell et al., 1990) constructs artificial neural networks using connections expressed in propositional logic. Along these lines, França et al. (2014, CILP++) build neural networks from a rule set for relation extraction. Our distinction is that we use first-order logic to *augment* a given architecture instead of designing a new one. Also, our framework is related to Kimmig et al. (2012, PSL) which uses a smooth extension of standard Boolean logic.

Hu et al. (2016) introduced an imitation learning framework where a specialized teacher-student network is used to distill rules into network parameters. This work could be seen as an instance of knowledge distillation (Hinton et al., 2015). Instead of such extensive changes to the learning procedure, our framework retains the original network design and augments existing interpretable layers.

Regularization with Logic Several recent lines of research seek to guide training neural networks by integrating logical rules in the form of additional terms in the loss functions (e.g., Rocktäschel et al., 2015) that essentially promote constraints among output labels (e.g., Du et al., 2019; Mehta et al., 2018), promote agreement (Hsu et al., 2018) or reduce inconsistencies across predictions (Minervini and Riedel, 2018).

Furthermore, Xu et al. (2018) proposed a general design of loss functions using symbolic knowledge about the outputs. Fischer et al. (2019) describe a method for for deriving losses that are friendly to gradient-based learning algorithms. Wang and Poon (2018) proposed a framework for integrating indirect supervision expressed via probabilistic logic into neural networks.

Learning with Structures Traditional structured prediction models (e.g. Smith, 2011) naturally admit constraints of the kind described in this paper. Indeed, our approach for using logic as a template-language is similar to Markov Logic Networks (Richardson and Domingos, 2006), where logical forms are compiled into Markov networks. Our formulation augments model scores with constraint penalties is reminiscent of the Constrained Conditional Model of Chang et al. (2012).

Recently, we have seen some work that allows backpropagating through structures (e.g. Huang et al., 2015; Kim et al., 2017; Yogatama et al., 2017; Niculae et al., 2018; Peng et al., 2018, and the references within). Our framework differs from them in that structured inference is not mandantory here. We believe that there is room to study the interplay of these two approaches.

6 Conclusions

In this paper, we presented a framework for introducing constraints in the form of logical statements to neural networks. We demonstrated the process of converting first-order logic into differentiable components of networks without extra learnable parameters and extensive redesign. Our experiments were designed to explore the flexibility of our framework with different constraints in diverse tasks. As our experiments showed, our framework allows neural models to benefit from external knowledge during learning and prediction, especially when training data is limited.

7 Acknowledgements

We thank members of the NLP group at the University of Utah for their valuable insights and suggestions; and reviewers for pointers to related works, corrections, and helpful comments. We also acknowledge the support of NSF SaTC-1801446, and gifts from Google and NVIDIA.

References

Martin Anthony. 2003. Boolean functions and artificial neural networks. *Boolean Functions*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. *International Conference on Learning Representations*.

Tarek R Besold, Artur d'Avila Garcez, Sebastian Bader, Howard Bowman, Pedro Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, Luis C Lamb, Daniel Lowd, Priscila Machado Vieira Lima, et al. 2017. Neural-symbolic learning and reasoning: A survey and interpretation. *arXiv preprint arXiv:1711.03902*.

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Ashok K Chandra and David Harel. 1985. Horn clause queries and generalizations. *The Journal of Logic Programming*, 2.

Ming-Wei Chang, Lev Ratinov, and Dan Roth. 2012. Structured learning with constrained conditional models. *Machine learning*, 88.

Nuri Cingillioglu and Alessandra Russo. 2019. Deeplogic: End-to-end logical reasoning. *AAAI 2019 Spring Symposium on Combining Machine Learning with Knowledge Engineering*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Xinya Du, Bhavana Dalvi, Niket Tandon, Antoine Bosselut, Wen tau Yih, Peter Clark, and Claire Cardie. 2019. Be consistent! improving procedural text comprehension using label consistency. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Marc Fischer, Mislav Balunovic, Dana Drachsler-Cohen, Timon Gehr, Ce Zhang, and Martin Vechev. 2019. Dl2: Training and querying neural networks with logic. In *International Conference on Machine Learning*.

Manoel VM França, Gerson Zaverucha, and Artur S d'Avila Garcez. 2014. Fast relational learning using bottom clause propositionalization with artificial neural networks. *Machine learning*, 94.

Kuzman Ganchev, Jennifer Gillenwater, Ben Taskar, et al. 2010. Posterior regularization for structured latent variable models. *Journal of Machine Learning Research*, 11.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. In *Neural Information Processing Systems*.

Wan-Ting Hsu, Chieh-Kai Lin, Ming-Ying Lee, Kerui Min, Jing Tang, and Min Sun. 2018. A unified model for extractive and abstractive summarization using inconsistency loss. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. 2016. Harnessing deep neural networks with logic rules. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

Yoon Kim, Carl Denton, Luong Hoang, and Alexander M Rush. 2017. Structured attention networks. In *International Conference on Learning Representations*.

Angelika Kimmig, Stephen Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. 2012. A short introduction to probabilistic soft logic. In *Proceedings of the NIPS Workshop on Probabilistic Programming: Foundations and Applications*.

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

Quoc V Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg S Corrado, Jeff Dean, and Andrew Y Ng. 2012. Building high-level features using large scale unsupervised learning. In *International Conference on Machine Learning*.

Hugo Liu and Push Singh. 2004. ConceptNet – A Practical Commonsense Reasoning Tool-Kit. *BT technology journal*, 22.

Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Wolfgang Maass, Georg Schnitger, and Eduardo D Sontag. 1994. A comparison of the computational power of sigmoid and boolean threshold circuits. In *Theoretical Advances in Neural Computation and Learning*, pages 127–151. Springer.

Sanket Vaibhav Mehta, Jay Yoon Lee, and Jaime Carbonell. 2018. Towards semi-supervised learning for deep semantic role labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Pasquale Minervini and Sebastian Riedel. 2018. Adversarially regularising neural nli models to integrate logical background knowledge. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*.

Vlad Niculae, André FT Martins, Mathieu Blondel, and Claire Cardie. 2018. SparseMAP: Differentiable sparse structured inference. In *International Conference on Machine Learning*.

Xingyuan Pan and Vivek Srikumar. 2016. Expressiveness of rectifier networks. In *International Conference on Machine Learning*.

Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. In *NIPS 2017 Autodiff Workshop*.

Hao Peng, Sam Thomson, and Noah A Smith. 2018. Backpropagating through Structured Argmax using a SPIGOT. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. 2017. Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*.

Matthew Richardson and Pedro Domingos. 2006. Markov logic networks. *Machine learning*, 62.

Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. 2015. Injecting logical background knowledge into embeddings for relation extraction. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Alexander M Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Stuart J Russell and Peter Norvig. 2016. *Artificial Intelligence: A Modern Approach*. Pearson Education Limited.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2017. Bidirectional attention flow for machine comprehension. *International Conference on Learning Representations*.

Noah A Smith. 2011. Linguistic structure prediction. *Synthesis lectures on human language technologies*, 4.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15.

Erik F Tjong Kim Sang and Sabine Buchholz. 2000. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of the 2nd Workshop on Learning Language in Logic and the 4th Conference on Computational Natural Language Learning*.

Geoffrey G Towell, Jude W Shavlik, and Michiel O Noordewier. 1990. Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence*.

Hai Wang and Hoifung Poon. 2018. Deep probabilistic logic: A unifying framework for indirect supervision. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. 2018. A semantic loss function for deep learning with symbolic knowledge. In *International Conference on Machine Learning*.

Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. 2017. Learning to compose words into sentences with reinforcement learning. In *International Conference on Machine Learning*.

A Appendices

Here, we explain our experiment setup for the three tasks: machine comprehension, natural language inference, and text chunking. For each task, we describe the model setup, hyperparameters, and data splits.

For all three tasks, we used Adam (Paszke et al., 2017) for training and use 300 dimensional GloVe (Pennington et al., 2014) vectors (trained on 840B tokens) as word embeddings.

A.1 Machine Comprehension

The SQuAD (v1.1) dataset consists of $87,599$ training instances and $10,570$ development examples. Firstly, for a specific percentage of training data, we sample from the original training set. Then we split the sampled set into 9/1 folds for training and development. The original development set is reserved for testing only. This is because that the official test set is hidden, and the number of models we need to evaluate is impractical for accessing official test set.

In our implementation of the BiDAF model, we use a learning rate 0.001 to train the model for 20 epochs. Dropout (Srivastava et al., 2014) rate is 0.2. The hidden size of each direction of BiLSTM encoder is 100. For ELMo models, we train for 25 epochs with learning rate 0.0002. The rest hyperparameters are the same as in (Peters et al., 2018). Note that we did neither pre-tune nor post-tune ELMo embeddings. The best model on the development split is selected for evaluation. No exponential moving average method is used. The scaling factor ρ's are manually grid-searched in $\{1, 2, 4, 8, 16\}$ without extensively tuning.

A.2 Natural Language Inference

We use Stanford Natural Language Inference (SNLI) dataset which has $549,367$ training, $9,842$ development, and $9,824$ test examples. For each of the percentages of training data, we sample the same proportion from the orginal development set for validation. To have reliable model selection, we limit the minimal number of sampled development examples to be 1000. The original test set is only for reporting.

In our implimentation of the BiLSTM variant of the Decomposable Attention (DAtt) model, we adopt learning rate 0.0001 for 100 epochs of training. The dropout rate is 0.2. The best model on the development split is selected for evaluation. The scaling factor ρ's are manually grid-searched in $\{0.5, 1, 2, 4, 8, 16\}$ without extensively tuning.

A.3 Text Chunking

The CoNLL2000 dataset consists of $8,936$ examples for training and $2,012$ for testing. From the original training set, both of our training and development examples are sampled and split (by 9/1 folds). Performances are then reported on the original full test set.

In our implementation, we set hidden size to 100 for each direction of BiLSTM encoder. Before the final linear layer, we add a dropout layer with probability 0.5 for regularization. Each model was trained for 100 epochs with learning rate 0.0001. The best model on the development split is selected for evaluation. The scaling factor ρ's are manually grid-searched in $\{1, 2, 4, 8, 16, 32, 64\}$ without extensively tuning.