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 hungerness 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.

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.\(^1\)

---

\(^1\)The code used for our experiments is archived here: https://github.com/utahnlp/layer_augmentation

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-differentiability 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
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 $v_i$ 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.

\(^2\)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.
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 $\rho$  In Eq. 1, the distance function serves to promote or inhibit the value of downstream neuron. The extent is controlled by the scaling factor $\rho$. 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 $\rho$, the output state depends on both the $Wx$ 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 $\rho$.

### 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.

<table>
<thead>
<tr>
<th>Antecedent</th>
<th>Distance $d(z)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bigwedge_i Z_i$</td>
<td>$\max(0, \sum_i z_i -</td>
</tr>
<tr>
<td>$\bigvee_i Z_i$</td>
<td>$\min(1, \sum_i z_i)$</td>
</tr>
<tr>
<td>$\neg \bigvee_i Z_i$</td>
<td>$\max(0, 1 - \sum_i z_i)$</td>
</tr>
<tr>
<td>$\neg \bigwedge_i Z_i$</td>
<td>$\min(1, N - \sum_i z_i)$</td>
</tr>
</tbody>
</table>

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 $(\neg A \lor B) \land (C \lor D)$. By introducing auxiliary variables, we can convert it into the conjunctive form $P \land Q$, where $(\neg A \lor B) \leftrightarrow P$ and $(C \lor D) \leftrightarrow Q$. 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 $(\neg A \lor B) \leftrightarrow P$, 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

$$y = d(z)$$

where $d$ is the distance function for the statement, $z$ 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 Ta-
ble 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 consequence, we can decompose it into multiple individual constraints. That is equivalent to constraining downstream nodes independently. Handling more complex consequences 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:

\[
p, q = \text{encoder}(p), \text{encoder}(q) \quad (3)
\]
\[
\hat{a}, \hat{a} = \sigma(\text{layers}(p, q)) \quad (4)
\]
\[
y, z = \sigma(\text{layers}(p, q, \hat{a}, \hat{a})) \quad (5)
\]
where \(p\) and \(q\) are the paragraph and query respectively, \(\sigma\) refers to the softmax activation, \(\hat{a}\) and \(\hat{a}\) are the bidirectional attentions from \(q\) to \(p\) and vice versa, \(y\) and \(z\) 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 alignments.

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

\[
R_1: \forall i, j \in C, K_{i,j} \rightarrow \hat{A}_{i,j}.
\]
\[
R_2: \forall i, j \in C, K_{i,j} \land \hat{A}_{i,j} \rightarrow \hat{A}_{i,j}.
\]

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
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.

<table>
<thead>
<tr>
<th>%Train</th>
<th>BiDAF +( R_1 ) +( R_2 )</th>
<th>+ELMo</th>
<th>+ELMo,( R_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>57.5 61.5 60.7</td>
<td>71.8</td>
<td>73.0</td>
</tr>
<tr>
<td>20%</td>
<td>65.7 67.2 66.6</td>
<td>76.9</td>
<td>77.7</td>
</tr>
<tr>
<td>40%</td>
<td>70.6 72.6 71.9</td>
<td>80.3</td>
<td>80.9</td>
</tr>
<tr>
<td>100%</td>
<td>75.7 77.4 77.0</td>
<td>83.9</td>
<td>84.1</td>
</tr>
</tbody>
</table>

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:

\[
\begin{align*}
    p, h &= \text{encoder}(p), \text{encoder}(h) \\
    \alpha, \alpha' &= \sigma(\text{layers}(p, h)) \\
    y &= \sigma(\text{layers}(p, h, \alpha, \alpha'))
\end{align*}
\]

Here, \( \sigma \) is the softmax activation, \( \alpha \) and \( \alpha' \) are bidirectional attentions, \( y \) 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:

\[
\begin{align*}
    &N_1: \forall i, j \in C, K_{i,j} \rightarrow A'_{i,j}, \\
    &N_2: \forall i, j \in C, K_{i,j} \land A_{i,j} \rightarrow A'_{i,j},
\end{align*}
\]

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:

\[ N_3: \quad Z_1 \land Z_2 \rightarrow \neg Y_{\text{Entail}}' \text{where} \]
\[ \exists j \in C, \neg (\exists i \in C, \text{A}_{i,j}^T) \leftrightarrow Z_1, \]
\[ \exists j \in C, \neg (\exists i \in C, \text{A}_{i,j}^T) \leftrightarrow Z_2. \]

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

![Computation graph of L-DAtt model](image)

Figure 4: (a) The computation graph of the L-DAtt model (attention directions omitted). (b) The augmented graph on the Entail label using \( N_3 \). Bold circles are extra neurons introduced. Unconstrained pre-activation scores are \( s \) while \( s' \) is the constrained score on Entail. Intermediate neurons are \( z_j \) and \( z_2 \). Constrained attentions \( a' \) 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 \( \rho \). 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.

<table>
<thead>
<tr>
<th>%Train</th>
<th>L-DAtt</th>
<th>+N1</th>
<th>+N2</th>
<th>+N3</th>
<th>+N_{2,3}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1%</td>
<td>61.2</td>
<td>64.9</td>
<td>63.9</td>
<td>62.5</td>
<td>64.3</td>
</tr>
<tr>
<td>2%</td>
<td>66.5</td>
<td>70.5</td>
<td>69.8</td>
<td>67.9</td>
<td>70.2</td>
</tr>
<tr>
<td>5%</td>
<td>73.4</td>
<td>76.2</td>
<td>76.6</td>
<td>74.0</td>
<td>76.4</td>
</tr>
<tr>
<td>10%</td>
<td>78.9</td>
<td>80.1</td>
<td>80.4</td>
<td>79.3</td>
<td>80.3</td>
</tr>
<tr>
<td>100%</td>
<td>87.1</td>
<td>86.9</td>
<td>87.1</td>
<td>87.0</td>
<td>86.9</td>
</tr>
</tbody>
</table>

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:

\[ x = \text{BiLSTM}(x) \quad (9) \]
\[ y = \sigma(\text{linear}(x)) \quad (10) \]

where \( x \) is the input sentence, \( \sigma \) is softmax, \( y \) are the output probabilities of BIO tags.

**Augmentation** We define the following predicates for input and output neurons:
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, +C1.5, and +CRF,C1.5, are on top of the BiLSTM baseline. For C1.4, ρ = 4 for all percentages. For C5, ρ = 16.

<table>
<thead>
<tr>
<th>%Train</th>
<th>BiLSTM</th>
<th>+CRF</th>
<th>+C1.5</th>
<th>+CRF,C1.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>5%</td>
<td>87.2</td>
<td>86.6</td>
<td><strong>88.9</strong></td>
<td>88.6</td>
</tr>
<tr>
<td>10%</td>
<td>89.1</td>
<td>88.8</td>
<td><strong>90.7</strong></td>
<td>90.6</td>
</tr>
<tr>
<td>20%</td>
<td>90.9</td>
<td>90.8</td>
<td><strong>92.1</strong></td>
<td><strong>92.1</strong></td>
</tr>
<tr>
<td>40%</td>
<td>92.5</td>
<td>92.5</td>
<td>93.4</td>
<td><strong>93.5</strong></td>
</tr>
<tr>
<td>100%</td>
<td>94.1</td>
<td>94.4</td>
<td>94.8</td>
<td><strong>95.0</strong></td>
</tr>
</tbody>
</table>

Yt,l The unconstrained decision that t-th word has label l.

Yt,l The constrained decision that t-th word has label l.

Nt 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.

C1: ∀t, Yt,B-VP → ¬Yt+1,I-NP
C2: ∀t, Yt,I-NP → ¬Yt+1,I-VP
C3: ∀t, Yt,I-VP → ¬Yt+1,I-NP
C4: ∀t, Yt,B-PP → ¬Yt+1,I-VP

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

C5: ∀t, Nt → ∨l∈{B-VP,I-VP,B-PP,I-PP} ¬Yt,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 C1.4, 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, Cingioglu 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).


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 mandatory here. We believe that there is room to study the interplay of these two approaches.

Also related to our attention augmentation is using word relatedness as extra input feature to attention neurons (e.g. Chen et al., 2018).

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


Use a learning rate $0$ for accessing official test set. The number of models we need to evaluate is impractical because that the official test set is hidden, and the 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.

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 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 $\rho$’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 implemtation 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 $\rho$’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 $\rho$’s are manually grid-searched in \{1, 2, 4, 8, 16, 32, 64\} without extensively tuning.