Attentive Tensor Product Learning

Qiuyuan Huang
Microsoft Research
Redmond, WA, USA
qihua@microsoft.com

Li Deng
Citadel, USA
deng629@gmail.com

Dapeng Wu
University of Florida
Gainesville, FL, USA
dpwu@ieee.org

Chang Liu
Citadel Securities
Chicago, IL, USA
liuchang2005acm@gmail.com

Xiaodong He
JD AI Research
Beijing, China
xiaohe.ai@outlook.com

Abstract

This paper proposes a novel neural architecture — Attentive Tensor Product Learning (ATPL) — to represent grammatical structures of natural language in deep learning models. ATPL exploits Tensor Product Representations (TPR), a structured neural-symbolic model developed in cognitive science, to integrate deep learning with explicit natural language structures and rules. The key ideas of ATPL are: 1) unsupervised learning of role-unbinding vectors of words via the TPR-based deep neural network; 2) the use of attention modules to compute TPR; and 3) the integration of TPR with typical deep learning architectures including long short-term memory and feedforward neural networks. The novelty of our approach lies in its ability to extract the grammatical structure of a sentence by using role-unbinding vectors, which are obtained in an unsupervised manner. Our ATPL approach is applied to 1) image captioning, 2) part of speech (POS) tagging, and 3) constituency parsing of a natural language sentence. The experimental results demonstrate the effectiveness of the proposed approach in all these three natural language processing tasks.

1 Introduction

Deep learning is an important tool in many speech and natural language processing (NLP) applications (Hinton and others 2012; Deng and Liu 2018). Since natural language is rich in grammatical structures, there has been an increasing interest in learning a vector representation to capture the grammatical structures of the natural language descriptions using deep learning models in recent years (Tai, Socher, and Manning 2015; Kumar et al. 2016; Kong et al. 2017).

In this work, we propose a new architecture, called *Attentive Tensor Product Learning (ATPL)*, to address this representation problem by exploiting Tensor Product Representations (TPR) (Smolensky and Legendre 2006; Smolensky et al. 2016; Palangi et al. 2017; Huang et al. 2017). TPR is a structured neural-symbolic model developed in cognitive science over 20 years ago. In the TPR theory, a sentence can be considered as a sequence of *roles* (i.e., grammatical components) where each role is connected to a *filler* (i.e., tokens). Given each role associated with a *role vector* r_t and each filler associated with a *filler vector* f_t, the TPR of a sentence can be computed as $S = \sum_t f_t r_t^\top$. Comparing with the popular RNN-based representations of a sentence, a good property of TPR is that decoding a token of a time stamp t can be computed directly by providing an *unbinding vector* u_t. That is, $f_t = S \cdot u_t$. Under the TPR theory, encoding and decoding a sentence is equivalent to learning the role vectors r_t or unbinding vectors u_t at each position t.

We employ the TPR theory to develop a novel attention-based neural network architecture for learning the unbinding vectors u_t that serve as the core of ATPL. That is, ATPL employs a form of recurrent neural networks to produce u_t one at a time. In each time, the TPR of the partial prefix of the sentence up to time $t-1$ is leveraged to compute the attention maps, which are then used to compute the TPR S_t as well as the unbinding vector u_t at time t. In doing so, our ATPL can not only be used to generate a sequence of tokens, but also be used to generate a sequence of *roles*, which can in turn interpret the syntactic/semantic structures of the sentence.

To demonstrate the effectiveness of our ATPL architecture, we apply it to three important NLP tasks: 1) image captioning; 2) POS tagging; and 3) constituency parsing of a sentence. The first task showcases our ATPL-based generator, while the latter two tasks are used to demonstrate the power of role vectors in interpreting sentences' syntactic structures. Our evaluation shows that on both image captioning and POS tagging, our approach can outperform previous state-of-the-art approaches. In particular, on the constituency parsing task, when the structural segmentation is given as a ground truth, our ATPL approach can beat the state-of-the-art by 3.5 to 4.4 points on the Penn TreeBank dataset. These results demonstrate that our ATPL is highly effective in capturing the syntactic structures of natural language sentences.

The paper is organized as follows. Section 2 discusses related work. In Section 3, we present the design of ATPL. Section 4 through Section 6 describe three applications of ATPL, i.e., image captioner, POS tagger, and constituency parser, respectively. Section 7 concludes the paper.

2 Related Work

Our proposed method for image captioning, the first NLP task we consider in this paper, follows a great deal of recent caption-generation literature on exploiting end-to-end deep learning with a CNN image-analysis front end producing a distributed representation that is then used to drive a natural-language generation process, typically using RNNs (Mao et al. 2015; Vinyals et al. 2015b; Karpathy and Fei-Fei 2015). Our grammatical interpretation of the structural roles of words in sentences is connected with other work that also incorporates deep learning into grammatically-structured networks (Tai, Socher, and Manning 2015; Andreas et al. 2015; Yogatama et al. 2016; Maillard, Clark, and Yogatama 2017). In these earlier studies, the network itself is not structured to match the grammatical structure of sentences being processed. In our work, the structure is fixed, but it is designed to support the learning of distributed representations that incorporate structure internal to the representations themselves — filler/role structure.

The second NLP task we consider in this paper is POS tagging. Methods for automatic POS tagging are our baselines. They include unigram tagging, bigram tagging, tagging using Hidden Markov Models (which are generative sequence models), maximum entropy Markov models (which are discriminative sequence models), rule-based tagging, and tagging using bidirectional maximum entropy Markov models (Jurafsky and Martin 2017). The celebrated Stanford POS tagger of (Manning 2017) uses a bidirectional version of the maximum entropy Markov model called a cyclic dependency network in (Toutanova et al. 2003).

Methods for automatic constituency parsing, the third NLP task tackled in this paper, include those based on probabilistic context-free grammars (CFGs) (Jurafsky and Martin 2017), the shift-reduce method (Zhu et al. 2013), sequence-to-sequence LSTMs (Vinyals et al. 2015a). Our constituency parser is similar to the sequence-to-sequence LSTMs (Vinyals et al. 2015a) since both use LSTM neural networks to design a constituency parser. Different from (Vinyals et al. 2015a), our constituency parser uses TPR and unbinding role vectors to extract features that contain grammatical information.

3 Attentive Tensor Product Learning

In this section, we present the ATPL architecture. We will first briefly revisit the Tensor Product Representation (TPR) theory, and then introduce several building blocks. In the end, we explain the ATPL architecture, which is illustrated in Figure 1.

3.1 Background: Tensor Product Representation

The TPR theory allows computing a vector representation of a sentence as the summation of its individual tokens while the order of the tokens is represented implicitly (Smolensky 1990; Huang et al. 2018; Lee et al. 2016). For a sentence of T words, denoted by $x_1, \cdots, x_T$, TPR theory considers the sentence as a sequence of *grammatical role slots* with each slot filled with a concrete token x_t. The role slot is often shortened and referred to as a *role*, while the token x_t referred to as a *filler*.

The TPR of the sentence can thus be computed as *binding* each role with a filler. Mathematically, each role is associated with a *role vector* $r_t \in \mathbb{R}^d$, and a filler with a *filler vector* $f_t \in \mathbb{R}^d$. Then the TPR of the sentence is

$$S = \sum_{t=1}^{T} f_t \cdot r_t^\top \tag{1}$$

where $S \in \mathbb{R}^{d\times d}$. Each role is also associated with a dual *unbinding vector* u_t so that $r_t^\top u_t = 1$ and $r_t^\top u_{t'} = 0, t' \neq t$; then

$$f_t = S u_t \tag{2}$$

Intuitively, Eq. (2) requires that $R^\top U = \mathbf{I}$, where $R = [r_1; \cdots; r_T]$, $U = [u_1; \cdots; u_T]$, and $\mathbf{I}$ is an identity matrix. In a simplified case, i.e., r_t is orthogonal to each other and $r_t^\top r_t = 1$, we can easily derive $u_t = r_t$.

Eqs. (1) and (2) provide means to *binding* or *unbinding* a TPR. Through these mechanisms, one can easily construct an encoder and a decoder to convert between a sentence and its TPR. All we need to compute is the role vector r_t (or its dual unbinding vector u_t) at each time step t.

3.2 Building blocks

Before we start introducing ATPL, we first introduce several building blocks repeatedly used in our construction.

An *attention module* over an input vector v is defined as

$$\mathrm{Attn}(v) = \sigma(Wv + b) \tag{3}$$

where σ is the sigmoid function, $W \in \mathbb{R}^{d_1\times d_2}$, $b \in \mathbb{R}^{d_1}$, d_2 is the dimension of v, and d_1 is the dimension of the output. Intuitively, $\mathrm{Attn}(\cdot)$ will output a vector as the attention heatmap, and d_1 is its dimension. W and b are two sets of parameters. Without specific notices, the sets of parameters of different attention modules are disjoint to each other.

We refer to a *Feed-Forward Neural Network* (FFNN) module as a single fully-connected layer:

$$\mathrm{FFNN}(v) = \mathbf{tanh}(Wv + b) \tag{4}$$

where W and b are the parameter matrix and the parameter vector with appropriate dimensions respectively, and $\mathbf{tanh}$ is the hyperbolic tangent function.

3.3 ATPL architecture

In this paper, we mainly focus on an ATPL decoder architecture that can decode a vector representation $\mathbf{v}$ into a sequence $f_1, \cdots, f_T$. The architecture is illustrated in Fig. 1.

If we require that the role vectors be orthogonal to each other, then to decode the filler f_t only needs to unbind the TPR of undecoded words, S_t:

$$f_t = S_t u_t = \big(\sum_{i=t}^{T} (W_e x_i) r_i^\top\big) u_t = W_e x_t \tag{5}$$

where $x_t \in \mathbb{R}^V$ is a one-hot encoding vector of dimension V and V is the size of the vocabulary; $W_e \in \mathbb{R}^{d\times V}$ is a word

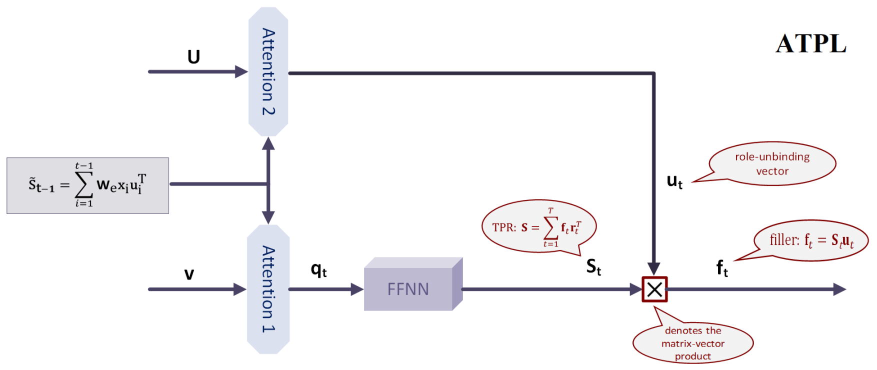
ATPL
U
Attention 2
$\tilde{S}_{t-1} = \sum_{i=1}^{t-1} W_e x_i u_i^T$
Attention 1
v
q_t
FFNN
TPR: $S = \sum_{t=1}^{T} f_t r_t^T$
S_t
u_t
role-unbinding vector
filler: $f_t = S_t u_t$
f_t
denotes the matrix-vector product

Figure 1: ATPL Architecture.

embedding matrix, the i-th column of which is the embedding vector of the i-th word in the vocabulary. We can use any algorithm to obtain the word embedding vectors; in this work, we choose the Stanford GLoVe algorithm with zero mean (Pennington, Socher, and Manning 2017).

To compute S_t and u_t, ATPL employs two attention modules controlled by $\tilde{S}_{t-1}$, which is the TPR of the so-far generated words $x_1, \cdots, x_{t-1}$:

$$\tilde{S}_{t-1} = \sum_{i=1}^{t-1} W_e x_i r_i^\top$$

On one hand, S_t is computed as follows:

$$\mathbf{q}_t = \mathbf{v} \odot \text{Attn}(h_{t-1} \oplus \text{vec}(\tilde{S}_{t-1})) \quad (6)$$

$$S_t = \text{FFNN}(\mathbf{q}_t) \quad (7)$$

where $\odot$ is the point-wise multiplication, $\oplus$ concatenates two vectors, and vec vectorizes a matrix. In this construction, h_{t-1} is the hidden state of an external LSTM, which we will explain later.

The key idea here is that we employ an attention model to place weights on each dimension of the vector $\mathbf{v}$, so that it can be used to compute S_t. Note it has been demonstrated that attention structures can be used to effectively learn any function (Vaswani et al. 2017). Our work adopts a similar idea to compute S_t from $\mathbf{v}$ and $\tilde{S}_{t-1}$.

On the other hand, similarly, u_t is computed as follows:

$$u_t = \mathbf{U}\text{Attn}(h_{t-1} \oplus \text{vec}(\tilde{S}_{t-1}))$$

where $\mathbf{U}$ is a constant normalized Hadamard matrix.

In doing so, ATPL can decode a vector $\mathbf{v}$ by recursively (1) computing S_t and u_t from $\tilde{S}_{t-1}$, (2) computing f_t as $S_t u_t$, and (3) setting $r_t = u_t$ and updating $\tilde{S}_t$. This procedure continues until the full sentence is generated.

3.4 Learning ATPL for NLP tasks

Note that the role and filler vectors can be learned end-to-end in various tasks. That is, the gradients of each parameters in the ATPL module can be computed using standard automatic differentiation mechanisms readily available in major deep learning frameworks, and thus all gradient-based optimiation algorithms can be used to train the ATPL module. At the same time, one the ATPL module is trained for one task with a large corpus, we can extract the module for computing role and filler vectors and apply them to train other tasks.

In the following sections, we will present three applications of ATPL. First, we will apply ATPL to an image captioning task (Section 4) and show that by end-to-end training, ATPL can help improve the performance upon LSTM.

Then, we extract the role vectors trained for the image captioning task (i.e., using the Coco dataset (COCO 2017)), and apply it to two traditional NLP tasks, namely POS tagger (Section 5) and constituency parsing (Section 6), and show that ATPL can achieve promising results.

4 Task 1: Image Captioning

To showcase our ATPL architecture, we first study its application in a widely used image captioning task (Fang et al. 2015; He and Deng 2017). Given an input image $\mathbf{I}$, a standard encoder-decoder can be employed to convert the image into an image feature vector $\mathbf{v}$, and then use the ATPL decoder to convert it into a sentence. The overall architecture is dipected in Fig. 2.

We evaluate our approach with several baselines on the COCO dataset (COCO 2017). The COCO dataset contains 123,287 images, each of which is annotated with at least 5 captions. We use the same pre-defined splits as (Karpathy and Fei-Fei 2015; Gan et al. 2017): 113,287 images for training, 5,000 images for validation, and 5,000 images for testing. We use the same vocabulary as that employed in (Gan et al. 2017), which consists of 8,791 words.

For the CNN of Fig. 2, we used ResNet-152 (He et al. 2016), pretrained on the ImageNet dataset. The image feature vector $\mathbf{v}$ has 2048 dimensions. The model is implemented in TensorFlow (Abadi and others 2015) with the default settings for random initialization and optimization by backpropagation. In our ATPL architecture, we choose $d = 32$, and the size of the LSTM hidden state to be 512. The vocabulary size $V = 8,791$. ATPL uses tags as in (Gan et al. 2017).

In comparison, we compare with (Vinyals et al. 2015b) and the state-of-the-art CNN-LSTM and SCN-LSTM (Gan et al. 2017). The main evaluation results on the MS COCO

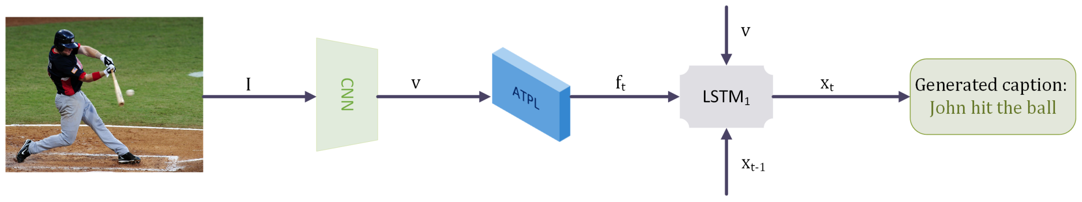

Figure 2: Architecture of image captioning.

Table 1: Performance of the proposed ATPL model on the COCO dataset.

Methods	METEOR	BLEU-1	BLEU-2	BLEU-3	BLEU-4	CIDEr
NIC (Vinyals et al. 2015b)	0.237	0.666	0.461	0.329	0.277	0.855
CNN-LSTM (Gan et al. 2017)	0.238	0.698	0.525	0.390	0.292	0.889
SCN-LSTM (Gan et al. 2017)	0.257	0.728	0.566	0.433	0.330	1.012
ATPL	**0.258**	**0.733**	**0.572**	**0.437**	**0.335**	**1.013**

dataset are reported in Table 1. The widely-used BLEU (Papineni et al. 2002), METEOR (Banerjee and Lavie 2005), and CIDEr (Vedantam, Lawrence Zitnick, and Parikh 2015) metrics are reported in our quantitative evaluation of the performance of the proposed scheme.

We can observe that, our ATPL architecture significantly outperforms all other baseline approaches across all metrics being considered. The results clearly attest to the effectiveness of the ATPL architecture. We attribute the performance gain of ATPL to the use of TPR in replace of a pure LSTM decoder, which allows the decoder to learn not only how to generate the *filler* sequence but also how to generate the *role* sequence so that the decoder can better understand the grammar of the considered language. Indeed, by manually inspecting the generated captions from ATPL, none of them has grammatical mistakes. We attribute this to the fact that our TPR structure enables training to be more effective and more efficient in learning the structure through the role vectors.

Note that the focus of this paper is on developing a Tensor Product Representation (TPR) inspired network to replace the core layers in an LSTM; therefore, it is directly comparable to an LSTM baseline. So in the experiments, we focus on comparison to a strong CNN-LSTM baseline. We acknowledge that more recent papers reported better performance on the task of image captioning. Performance improvements in these more recent models are mainly due to using better image features such as those obtained by Region-based Convolutional Neural Networks (R-CNN), or using reinforcement learning (RL) to directly optimize metrics such as CIDEr to provide a better context vector for caption generation, or using an ensemble of multiple LSTMs, among others. However, the LSTM is still playing a core role in these works and we believe improvement over the core LSTM, in both performance and interpretability, is still very valuable. Deploying these new features and architectures (R-CNN, RL, and ensemble) with ATPL is our future work.

5 Task 2: POS Tagging

In this section, we study the application of ATPL in the POS tagging task. Intuitively, given a sentence $x_1, ..., x_T$, POS tagging is to assign a POS tag denoted as z_t, for each token x_t. In the following, we first present our model using ATPL for POS tagging, and then evaluate its performance.

5.1 ATPL POS tagging architecture

Based on TPR theory, the role vector (as well as its dual unbinding vector) contains the POS tag information of each word. Hence, we first use ATPL to compute a sequence of unbinding vectors u_t which is of the same length as the input sentence. Then we take u_t and x_t as input to a bidirectional LSTM model to produce a sequence of POS tags.

Our training procedure consists of two steps. In the first step, we employ an unsupervised learning approach to learn how to compute u_t. Fig. 3 shows a sequence-to-sequence structure, which uses an LSTM as the encoder, and ATPL as the decoder; during the training phase of Fig. 3, the input is a sentence and the expected output is the same sentence as the input. Then we use the trained system in Fig. 3 to produce the unbinding vectors u_t for a given input sentence $x_1, ..., x_T$.

In the second step, we employ a bidirectional LSTM (B-LSTM) module to convert the sequence of u_t into a sequence of hidden states $\mathbf{h}$. Then we compute a vector $z_{1,t}$ from each $(x_t, \mathbf{h}_t)$ pair, which is the POS tag at position t. This procedure is illustrated in Figure 4.

The first step follows ATPL and is straightforward. Below, we focus on explaining the second step. In particular, given the input sequence u_t, we can compute the hidden states as

$$\overrightarrow{\mathbf{h}}_t, \overleftarrow{\mathbf{h}}_t = BLSTM(u_t, \overrightarrow{\mathbf{h}}_{t-1}, \overleftarrow{\mathbf{h}}_{t+1}) \tag{8}$$

Then, the POS tag embedding is computed as

$$\mathbf{z}_{1,t} = \mathbf{softmax}\big(\overrightarrow{\mathbf{W}}(x_t)\overrightarrow{\mathbf{h}}_t + \overleftarrow{\mathbf{W}}(x_t)\overleftarrow{\mathbf{h}}_t\big) \tag{9}$$

Here $\overrightarrow{\mathbf{W}}(x_t)$ is computed as follows

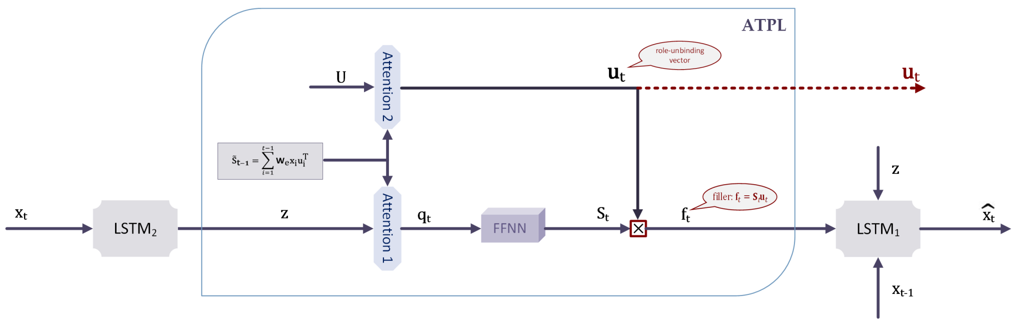

Figure 3: Architecture for acquisition of unbinding vectors of a sentence.

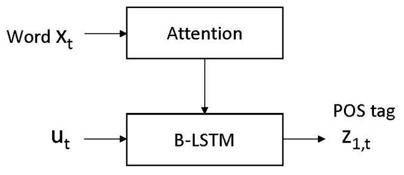

Figure 4: Structure of POS tagger.

Table 2: Performance of POS Tagger.

	(MANNING 2017)		OUR POS TAGGER	
	WSJ 22	WSJ 23	WSJ 22	WSJ 23
ACCURACY	0.972	0.973	**0.973**	**0.974**

$$\overrightarrow{\mathbf{W}}(\mathbf{x}) = \overrightarrow{\mathbf{W}}_a \cdot \mathrm{diag}(\overrightarrow{\mathbf{W}}_b \cdot x_t) \cdot \overrightarrow{\mathbf{W}}_c \qquad (10)$$

where diag$(\cdot)$ constructs a diagonal matrix from the input vector; $\overrightarrow{\mathbf{W}}_a, \overrightarrow{\mathbf{W}}_b, \overrightarrow{\mathbf{W}}_c$ are matrices of appropriate dimensions. $\overleftarrow{\mathbf{W}}_{3,h}(\mathbf{x}_t)$ is defined in the same manner as $\overrightarrow{\mathbf{W}}_{3,h}(\mathbf{x}_t)$, though a different set of parameters is used.

Note that $\mathbf{z}_{1,t}$ is of dimension P, which is the total number of POS tags. Clearly, this model can be trained end-to-end by minimizing a cross-entropy loss.

5.2 Evaluation

To evaluate the effectiveness of our model, we test it using the Penn TreeBank dataset (Marcus et al. 2017). In particular, we first train the sequence-to-sequence in Fig. 3 using the sentences of Wall Street Journal (WSJ) Section 0 through Section 21 and Section 24 in Penn TreeBank data set (Marcus et al. 2017). Afterwards, we use the same dataset to train the B-LSTM module in Figure 4.

Once the model gets trained, we test it on WSJ Section 22 and 23 respectively. We compare the accuracy of our approach against the state-of-the-art Stanford parser (Manning 2017). The results are presented in Table 2. From the table, we can observe that our approach outperforms the baseline. This confirms our hypothesis that the unsupervisely trained unbinding vector u_t indeed captures grammatical information, so as to be used to effectively predict grammar structures such as POS tags.

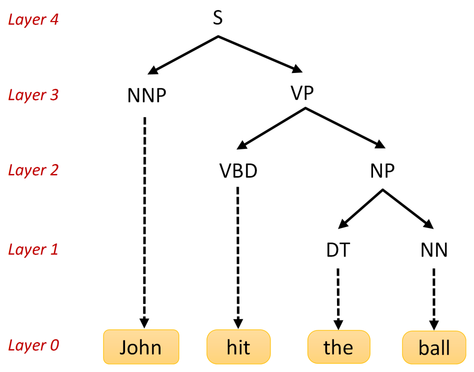

Figure 5: The parse tree of a sentence and its layers.

6 Task 3: Constituency Parsing

In this section, we briefly review the constituency parsing task, and then present our approach, which contains three component: segmenter, classifier, and creator of a parse tree. In the end, we compare our approach against the state-of-the-art approach in (Vinyals et al. 2015a).

6.1 A brief review of constituency parsing

Constituency parsing converts a natural language into its parsing tree. Fig. 5 provides an example of the parsing tree on top of its corresponding sentence. From the tree, we can label each node into layers, with the first layer (Layer 0) consisting of all tokens from the original sentence. Layer k contains all internal nodes whose depth with respect to the closest leaf that it can reach is k.

In particular, at Layer 1 are all POS tags associated with each token. In higher layers, each node corresponds to a *substring*, a consecutive subsequence, of the sentence. Each node corresponds to a grammar structure, such as a single word, a phrase, or a clause, and is associated with a category. For example, in Penn TreeBank, there are over 70 types of categories, including (1) clause-level tags such as S (simple declarative clause), (2) phrase-level tags such as NP (noun

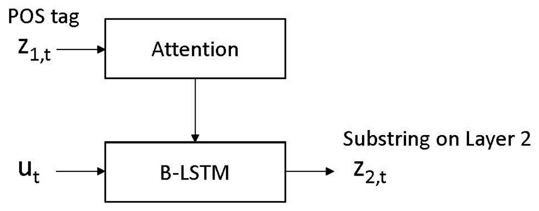

Figure 6: Structure of the segmenter on Layer 2.

phrase), VP (verb phrase), (3) word-level tags such as NNP (Proper noun, singular), VBD (Verb, past tense), DT (Determiner), NN (Noun, singular or mass), (4) punctuation marks, and (5) special symbols such as $.

The task of constituency parsing recovers both the tree-structure and the category associated with each node. In our approach to employ ATPL to construct the parsing tree, we use an encoding z to encode the tree-structure. Our approach first generates this encoding from the raw sentence, layer-by-layer, and then predict a category to each internal node. In the end, an algorithm is used to convert the encoding z with the categories into the full parsing tree. In the following, we present the three sub-routines.

6.2 Segmenting a sentence into a tree-encoding

We first introduce the concept of the encoding z. For each layer k, we assign a value $\mathbf{z}_{k,t}$ to each location t of the input sentence. In the first layer, $\mathbf{z}_{1,t}$ simply encodes the POS tag of input token x_i. In a higher level, $\mathbf{z}_{k,t}$ is either 0 or 1. Thus the sequence $\mathbf{z}_{k,t}$ forms a sequence with alternating sub-sequences of consecutive 0s and consecutive 1s. Each of the longest consecutive 0s or consecutive 1s indicate one internal node at layer k, and the consecutive positions form the substring of the node. For example, the second layer of Fig. 5 is encoded as $\{0, 1, 0, 0\}$, and the third layer is encoded as $\{0, 1, 1, 1\}$.

The first component of our ATPL-based parser predicts $\mathbf{z}_{k,t}$ layer-by-layer. Note that the first layer is simply the POS tags, so we will not repeat it. In the following, we first explain how to construct the second layer's encoding $\mathbf{z}_{2,t}$, and then we show how it can be expanded to construct higher layer's encoding $\mathbf{z}_{k,t}$ for $k \geq 3$.

Constructing the second layer $\mathbf{z}_{2,t}$. We can view $\mathbf{z}_{2,t}$ as a special tag over the POS tag sequence, and thus the same approach to compute the POS tag can be adapted here to compute $\mathbf{z}_{2,t}$. This model is illustrated in Fig. 6.

In particular, we can compute the hidden state from the unbinding vectors from the raw sentence as before:

$$\overrightarrow{\mathbf{h}}_{2,t}, \overleftarrow{\mathbf{h}}_{2,t} = BLSTM(u_t, \overrightarrow{\mathbf{h}}_{2,t-1}, \overleftarrow{\mathbf{h}}_{2,t+1}) \qquad (11)$$

and the output of the attention-based B-LSTM is given as below

$$\mathbf{z}_{2,t} = \sigma_s(\overrightarrow{\mathbf{W}}_2(\mathbf{z}_{1,t})\overrightarrow{\mathbf{h}}_{2,t} + \overleftarrow{\mathbf{W}}_2(\mathbf{z}_{1,t})\overleftarrow{\mathbf{h}}_{2,t}) \qquad (12)$$

where $\overrightarrow{\mathbf{W}}_{2,h}(\mathbf{z}_{1,t})$ and $\overleftarrow{\mathbf{W}}_{2,h}(\mathbf{z}_{1,t})$ are defined in the same manner as in (10).

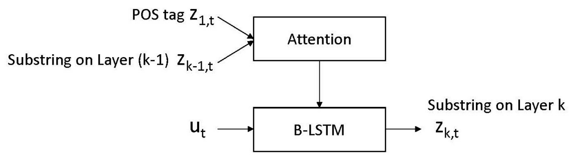

Figure 7: Structure of the segmenter on Layer $k \geq 3$.

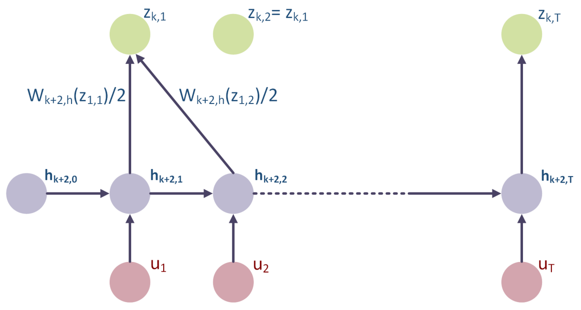

Figure 8: Segmenting Layer $k \geq 3$.

Constructing higher layer's encoding $\mathbf{z}_{k,t}$ $(k \geq 3)$. Now we move to higher levels. For a layer $k \geq 3$, to predict $\mathbf{z}_{k,t}$, our model takes both the POS tag input $\mathbf{z}_{1,t}$ and the $(k-1)$-th layer's encoding $\mathbf{z}_{k-1,t}$. The high-level architecture is illustrated in Fig. 7.

Let us denote

$$\mathbf{z}_{k,t} = \mathbf{softmax}(J_{k,t})$$

the key difference is how to compute $J_{k,t}$. Intuitively, $J_{k,t}$ is an embedding vector corresponding to the node, whose substring contains token x_t. Assume word x_t is in the m-th substring of Layer $k-1$, which is denoted by $s_{k-1,m}$. Then, the embedding $J_{k,t}$ can be computed as follows:

$$J_{k,t} = \sum_{i \in s_{k-1,m}} \frac{\overrightarrow{\mathbf{W}}_k(\mathbf{z}_{1,i})\overrightarrow{\mathbf{h}}_{k,i} + \overleftarrow{\mathbf{W}}_k(\mathbf{z}_{1,i})\overleftarrow{\mathbf{h}}_{k,i}}{|s_{k-1,m}|} \qquad (13)$$

Here, $\overrightarrow{\mathbf{h}}_{k,i}$ and $\overleftarrow{\mathbf{h}}_{k,i}$ are the hidden states of BLSTM running over the unbinding vectors as before, and $\overrightarrow{\mathbf{W}}_k(\cdot)$ and $\overleftarrow{\mathbf{W}}_k(\cdot)$ are defined in a similar fashion as (10). We use $|\cdot|$ to indicate the cardinality of a set.

The most interesting part is that $J_{k,t}$ aggregates all embeddings computed from the substring of the previous layer $s_{k-1,m}$. Note that the set $s_{k-1,m}$ of indexes can be computed easily from $\mathbf{z}_{k-1,t}$. Note that many different aggregation functions can be used. In (13), we choose to use the average function. The process of this calculuation is illustrated in Fig. 8.

6.3 Classification of substrings

Once the tree structure is computed, we attach a category to each internal node. We employ a similar approach as predicting $\mathbf{z}_{k,t}$ for $k \geq 3$ to predict this category $\mathbf{z}_t^{(k)}$. Note that, in this time, the encoding $\mathbf{z}_{k,t}$ of the internal node is already

Table 3: Performance of Constituency Parser.

	(Vinyals et al. 2015a)		Our parser		Our parser with ground-truth $\mathbf{z}_{k,t}$ ($k \geq 2$)	
	WSJ 22	WSJ 23	WSJ 22	WSJ 23	WSJ 22	WSJ 23
Precision	N/A	N/A	0.898	0.910	0.952	0.952
Recall	N/A	N/A	0.901	0.907	0.973	0.978
F-1 measure	0.928	0.921	0.900	0.908	0.963	0.965

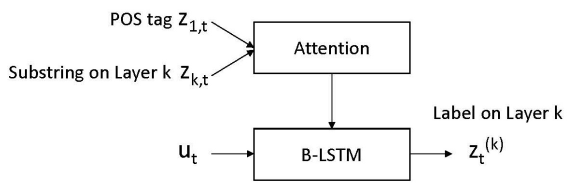

Figure 9: Structure of the classifier on Layer k.

computed. Thus, instead of using the encoding $\mathbf{z}_{k-1,t}$ from the previous layer, we use the encoding of the current layer $\mathbf{z}_{k,t}$ to predict $\mathbf{z}_t^{(k)}$ directly. This procedure is illustrated in Fig. 9.

Similar to (13), we have $\mathbf{z}_t^{(k)} = \mathbf{softmax}(E_{k,t})$, where $E_{k,t}$ is computed by ($\forall t \in \{t : \mathbf{x}_t \in s_{k,m}\}$)

$$E_{k,t} = \sum_{i \in s_{k,m}} \frac{\overrightarrow{\mathbf{W}}_k(\mathbf{z}_{1,i})\overrightarrow{\mathbf{h}}_{k,i} + \overleftarrow{\mathbf{W}}_k(\mathbf{z}_{1,i})\overleftarrow{\mathbf{h}}_{k,i}}{|s_{k,m}|} \tag{14}$$

Here, we slightly overload the variable names. We emphasize that the parameters $\overrightarrow{\mathbf{W}}$ and $\overleftarrow{\mathbf{W}}$ and the hidden states $\overrightarrow{\mathbf{h}}_{k,i}$ and $\overleftarrow{\mathbf{h}}_{k,i}$ are both independent to the ones used in (14).

Note that the main different between (14) and (13) is that, the aggregation is operated over the set $s_{k,t}$, i.e., the substring at layer k, rather than $s_{k-1,t}$, i.e., the substring at layer $k-1$. Also, $E_{k,t}$'s dimension is the same as the total number of categories, while $J_{k,t}$'s dimension is 2.

6.4 Creating a parse tree

Once both $\mathbf{z}_{k,t}$ and $\mathbf{z}_t^{(k)}$ are constructed, we can create the parse tree out of them using a linear-time sub-routine. Due to space limitation, we omit the details.For the example in Fig. 5, the output is (S(NNP John)(VP(VBD hit)(NP(DT the)(NN ball)))).

6.5 Evaluation

We now evaluate our constituency parsing approach against the state-of-the-art approach (Vinyals et al. 2015a) using WSJ data set in Penn TreeBank. Similar to our setup for POS tag, we train our model using WSJ Section 0 through Section 21 and Section 24, and evaluate it on Section 22 and 23.

Table 3 shows the performance for both (Vinyals et al. 2015a) and our proposed approach. In addition, we also evaluate our approach assuming the tree-structure encoding $\mathbf{z}_{k,t}$ is known. In doing so, we can evaluate the performance of our classification module of the parser. Note, the POS tag is not provided.

We observe that the F-1 measure of our approach is two points worse than (Vinyals et al. 2015a); however, when the ground-truth of $\mathbf{z}_{k,t}$ is provided, the F-1 measure becomes four points higher than that reported in (Vinyals et al. 2015a), which is significant. Therefore, we attribute the somewhat lower performance of our approach to the lack of our model's ability in effectively predicting the tree-encoding $\mathbf{z}_{k,t}$.

7 Conclusion

In this paper, we propose a novel ATPL approach to natural language generation and related tasks. The model has a novel architecture motivated by insights derived from the use of Tensor Product Representations for encoding and processing symbolic structure through neural computation. In our experiments, we first evaluate the proposed model on the task of image captioning. Compared with widely adopted LSTM-based models, our proposed ATPL gives significant improvements on all major metrics including METEOR, BLEU, and CIDEr. We further observe that the unbinding vectors contain important grammatical information. This allows us to design an effective POS tagger and constituency parser with unbinding vectors as input, the other two NLP tasks evaluated using ATPL. Our findings reported in this paper demonstrate the effectiveness of the ATPL architecture as well as the underlying TPRs. In the future, we will explore the use of TPR and ATPL methods in a wider set of NLP tasks than reported in this paper, and distill further insight into structured representations of natural language.

References

Abadi, M., et al. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Andreas, J.; Rohrbach, M.; Darrell, T.; and Klein, D. 2015. Deep compositional question answering with neural module networks. *arXiv preprint arXiv:1511.02799* 2.

Banerjee, S., and Lavie, A. 2005. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the ACL workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, 65–72. Association for Computational Linguistics.

COCO. 2017. Coco dataset for image captioning. http://mscoco.org/dataset/#download.

Deng, L., and Liu, Y. 2018. *Deep Learning in Natural Language Processing*. Springer.

Fang, H.; Gupta, S.; Iandola, F.; Srivastava, S.; Deng, L.; Dollar, P.; Gao, J.; and He, X. 2015. From captions to visual concepts and back. In *Proc. IEEE conference on computer vision and pattern Recognition*.

Gan, Z.; Gan, C.; He, X.; Pu, Y.; Tran, K.; Gao, J.; Carin, L.; and Deng, L. 2017. Semantic compositional networks for visual captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

He, X., and Deng, L. 2017. Deep learning for image-to-text generation: A technical overview. In *IEEE Signal Processing Magazine*, volume 34.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.

Hinton, G., et al. 2012. Deep neural networks for acoustic modeling in speech recognition. In *IEEE Signal Processing Magazine*, volume 29, 82–97.

Huang, Q.; Smolensky, P.; He, X.; Deng, L.; and Wu, D. 2017. A neural-symbolic approach to design of captcha. In *Advances in Neural Information Processing Systems Workshop*.

Huang, Q.; Smolensky, P.; He, X.; Deng, L.; and Wu, D. 2018. Tensor product generation networks for deep NLP modeling. In *Proc. Conf. NAACL (Long Papers)*, volume 1, 1263–1273.

Jurafsky, D., and Martin, J. H. 2017. *Speech and Language Processing*. 3rd ed. draft edition edition.

Karpathy, A., and Fei-Fei, L. 2015. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3128–3137.

Kong, L.; Alberti, C.; Andor, D.; Bogatyy, I.; and Weiss, D. 2017. Dragnn: A transition-based framework for dynamically connected neural networks. *arXiv preprint arXiv:1703.04474*.

Kumar, A.; Irsoy, O.; Ondruska, P.; Iyyer, M.; Bradbury, J.; Gulrajani, I.; Zhong, V.; Paulus, R.; and Socher, R. 2016. Ask me anything: Dynamic memory networks for natural language processing. In *International Conference on Machine Learning*, 1378–1387.

Lee, M.; He, X.; Yih, S.; Gao, J.; Deng, L.; and Smolensky, P. 2016. Reasoning in vector space: An exploratory study of question answering. In *Proc. Int. Conf. Learning Representations (ICLR)*.

Maillard, J.; Clark, S.; and Yogatama, D. 2017. Jointly learning sentence embeddings and syntax with unsupervised tree-lstms. *arXiv preprint arXiv:1705.09189*.

Manning, C. 2017. Stanford parser. https://nlp.stanford.edu/software/lex-parser.shtml.

Mao, J.; Xu, W.; Yang, Y.; Wang, J.; Huang, Z.; and Yuille, A. 2015. Deep captioning with multimodal recurrent neural networks (m-rnn). In *Proceedings of International Conference on Learning Representations*.

Marcus, M. P.; Santorini, B.; Marcinkiewicz, M. A.; and Taylor, A. 2017. Penn treebank. https://catalog.ldc.upenn.edu/ldc99t42.

Palangi, H.; Huang, Q.; Smolensky, P.; He, X.; and Deng, L. 2017. Grammatically-interpretable learned representations in deep nlp models. In *Advances in Neural Information Processing Systems Workshop*.

Papineni, K.; Roukos, S.; Ward, T.; and Zhu, W.-J. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, 311–318. Association for Computational Linguistics.

Pennington, J.; Socher, R.; and Manning, C. 2017. Stanford glove: Global vectors for word representation. https://nlp.stanford.edu/projects/glove/.

Smolensky, P., and Legendre, G. 2006. *The harmonic mind: From neural computation to optimality-theoretic grammar. Volume 1: Cognitive architecture*. MIT Press.

Smolensky, P.; Lee, M.; He, X.; Yih, S.; Gao, J.; and Deng, L. 2016. Basic reasoning with tensor product representations. In *arXiv:1601.02745*.

Smolensky, P. 1990. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial intelligence* 46(1-2):159–216.

Tai, K. S.; Socher, R.; and Manning, C. D. 2015. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*.

Toutanova, K.; Klein, D.; Manning, C. D.; and Singer, Y. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proc. NAACL*, 173–180.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, 6000–6010.

Vedantam, R.; Lawrence Zitnick, C.; and Parikh, D. 2015. Cider: Consensus-based image description evaluation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4566–4575.

Vinyals, O.; Kaiser, Ł.; Koo, T.; Petrov, S.; Sutskever, I.; and Hinton, G. 2015a. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, 2773–2781.

Vinyals, O.; Toshev, A.; Bengio, S.; and Erhan, D. 2015b. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3156–3164.

Yogatama, D.; Blunsom, P.; Dyer, C.; Grefenstette, E.; and Ling, W. 2016. Learning to compose words into sentences with reinforcement learning. *arXiv preprint arXiv:1611.09100*.

Zhu, M.; Zhang, Y.; Chen, W.; Zhang, M.; and Zhu, J. 2013. Fast and accurate shift-reduce constituent parsing. In *Proc. Annual Meeting of the Association for Computational Linguistics (ACL)*, 434–443.