Improving Constituent Representation with Hypertree Neural Networks

Hao Zhou[◇] , **Gongshen Liu**[◇] , **Kewei Tu**[†*]

[◇] School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University
[†] School of Information Science and Technology, ShanghaiTech University
Shanghai Engineering Research Center of Intelligent Vision and Imaging
{zhou1998,lgshen}@sjtu.edu.cn
tukw@shanghaitech.edu.cn

Abstract

Many natural language processing tasks involve text spans and thus high-quality span representations are needed to enhance neural approaches to these tasks. Most existing methods of span representation are based on simple derivations (such as max-pooling) from word representations and do not utilize compositional structures of natural language. In this paper, we aim to improve representations of constituent spans using a novel hypertree neural networks (HTNN) that is structured with constituency parse trees. Each node in the HTNN represents a constituent of the input sentence and each hyperedge represents a composition of smaller child constituents into a larger parent constituent. In each update iteration of the HTNN, the representation of each constituent is computed based on all the hyperedges connected to it, thus incorporating both bottom-up and top-down compositional information. We conduct comprehensive experiments to evaluate HTNNs against other span representation models and the results show the effectiveness of HTNN.

1 Introduction

Distributed span representations are useful in various natural language processing tasks such as question answering (Seo et al., 2019), coreference resolution (Lee et al., 2017), sentiment classification (Yin et al., 2020) and semantic role labeling (He et al., 2018). Since spans can have arbitrary length, to get fix-dimensional span representations, existing methods are mostly based on simple derivations from word or sub-word representations. These methods either apply some form of pooling over all the words within the target span or utilize only the boundary words of the target span to compute the span representation. There have also been efforts
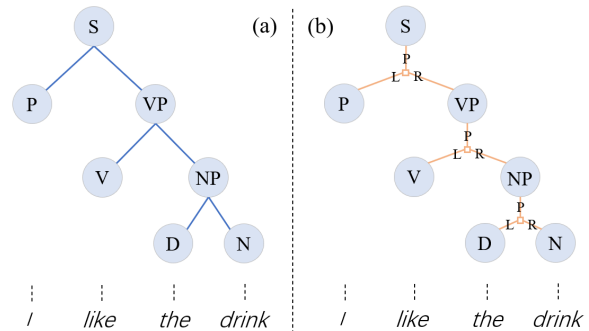

Figure 1: An example binarized constituency parse tree (a) and its corresponding hypertree (b). For each hyperedge, P/L/R denote the parent, left-child, right-child constituent spans respectively.

to improve word representation learning using information from text spans, which may at the same time improve span representations built on top of word representations (Joshi et al., 2020). Further, some previous methods introduce short text spans (phrases or n-grams) as additional inputs to BERT and thus learn contextual representations for such spans along with word representations (Lai et al., 2021; Zhang et al., 2021).

However, it is known that natural language text has underlying compositional structure, which can often be represented with constituency parse trees (Chomsky, 1957). What is missing from the aforementioned span representation methods is the utilization of compositional structures both within and outside text spans. Specifically, the recursive compositional structure inside a text span suggests more structured bottom-up computation of the span representation from word representations; the compositional structure outside a text span specifies how the span joins its sibling spans to form a larger span, implying top-down computation of the span representation from its parent and siblings[1]. By

---

[1]This is only possible for constituent spans and one has to revert to aforementioned methods for distituent spans. However, we note that most NLP tasks concern constituents more

taking into account such compositional structures, we could obtain better span representations and avoid problems with existing methods such as insufficient modeling of meaning composition (Yu and Ettinger, 2020).

There are several existing methods of incorporating constituency tree structures in neural networks that can be used to produce constituent span representations. These methods fall into two main categories: recursive neural networks (RvNNs) (Goller and Kuchler, 1996) and graph neural networks (GNNs). RvNN-based models recursively compose span representations from their sub-spans following a constituency tree structure (Socher et al., 2011; Tai et al., 2015). They hence only model bottom-up composition and miss information from parent and sibling spans in their span representations. Some extensions of RvNNs (Le and Zuidema, 2014; Drozdov et al., 2019) try to tackle this problem by additionally modeling top-down (de-)composition in a similar recursive fashion. However, they separate the representations computed from the directions and disallow them to directly interact with (and potentially disambiguate) each other. GNN-based models, such as graph convolutional networks (GCN) (Kipf and Welling, 2016) and graph attention networks (Veličković et al., 2017), have been applied based on constituency tree structures to compute word and span representations (Marcheggiani and Titov, 2020; Li et al., 2020). One important flaw of GNN-based methods is that a GNN represents each composition with multiple edges that become mixed up with edges from other compositions during GNN updates, thus losing critical information of the compositional structure. More specifically, a GNN does not adequately formulate and differentiate bottom-up computation from all the child spans versus top-down computation from the parent and sibling spans. In addition, most GNN-based methods do not model relations between sibling spans.

To overcome the above drawbacks, we propose hypertree neural networks (HTNN) to improve constituent span representations. Specifically, we view a constituency parse tree as a hypertree (Figure 1(b)), in which each node represents a constituent and each hyperedge is a tuple of multiple nodes representing a composition of smaller child constituents into a larger parent constituent. We then build an HTNN with the hypertree as its skele-

---

than distituents. See statistics in Section 3.4.

ton. During iterative update in the HTNN, we first visit each hyperedge, computing a representation of each node in the hyperedge from the other nodes using a direction-specific composition function; then we visit each node, aggregating its representations computed from all the hyperedges connected to it. In this way, not only can we keep the hierarchical information during encoding process, but also update the representation of constituents simultaneously. HTNNs combine the strengths of RvNNs and GNNs in span representation while avoid their drawbacks. Like GNNs, HTNNs compute a single unified representation for each span that integrates information from all the directions in the constituency tree. On the other hand, HTNNs follow RvNNs and utilize direction-specific composition functions to group and integrate information from different directions.

To evaluate the effectiveness of our method, we firstly conduct detailed experiments on three probing tasks: semantic role classification, named entity labeling, coreference arc prediction. Then we apply our method to the task of semantic role labeling. The results show that our method is superior to word-based, RvNN-based models and GNN-based models.

2 Hypertree Neural Networks

2.1 Overview

A hypergraph is a generalization of a graph in which an edge can join any number of nodes. A hypertree is a hypergraph $\mathcal{H}$ such that there exists a tree $\mathcal{T}$ and every hyperedge of $\mathcal{H}$ is the set of nodes of a connected subtree of $\mathcal{T}$. We consider a directed version of hypertree that defines a hyperedge as a *tuple* of nodes instead of a set.

There is a long tradition of representing constituency parse trees with hypergraphs or hypertrees (Klein and Manning, 2001; Huang and Chiang, 2005). We define the hypertree representation of a constituency parse tree as follows. Each node in the hypertree represents a constituent in the parse tree and is labeled with the nonterminal symbol of the constituent. Each hyperedge represents a composition of smaller child constituents into a larger parent constituent. In this paper, we assume that the constituency parse tree is binarized, so each hyperedge contains exactly three nodes, marked with P (parent), L (left child) and R (right child) respectively. Figure 1 shows the hypertree representation of an example constituency parse tree.
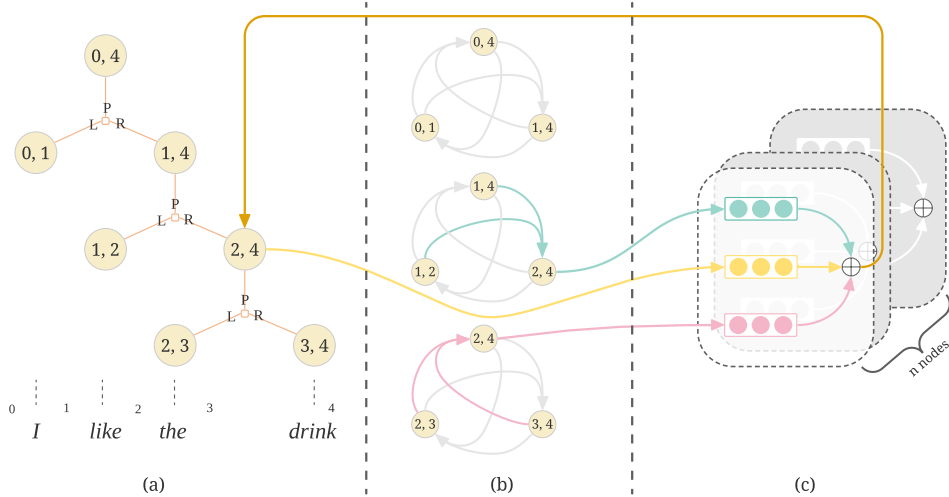
Figure 2: Illustration of an HTNN. We show the update process of node (2,4). (a) The hypertree structure. (b) Composition within hyperedges. (c) Aggregation of multiple representations.

We generalize the methodology of GNNs on hypertrees and propose hypertree neural networks (HTNN, Figure 2). Similar to GNNs, several HTNN layers are stacked to form the model. Each layer is composed of two modules: composition within hyperedges (Section 2.3), which computes a representation of each node in a hyperedge using the other nodes from the same hyperedge; and aggregation of multiple representations (Section 2.4), which aggregates the representations of a node from all the hyperedges connected to it and forms a single unified representation. Each layer is fed with the node representation from the previous layer. The initial node representation in the first layer is computed from word representation using the attention pooling method (Section 2.2).

We treat the node representation from the last layer as the final constituent span representations and feed them into downstream task-specific decoders. For example, we may run an MLP classifier on top of a span representation for named entity recognition, or run a biaffine classifier on top of the representations of two spans for semantic role labeling. Training of HTNN can be done by optimizing any task-specific objective in an end-to-end manner along with the initial span encoder and the task-specific decoder.

2.2 Initialization of Node Representation

We follow Toshniwal et al. (2020) to compute initial span representations from word representations produced by a pretrained language model. We choose to use their attention pooling method because it performs the best in our pilot experiments. Specifically, the input sentence $\mathbf{x} = \{w_1, w_2, ..., w_l\}$ of l words, is firstly tokenized and passed through a pretrained language model to get contextual token embeddings $\{\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_T\}$. These embeddings are then fed to the attention pooling module to get fix-dimensional span embeddings. For a span $s = [i, j]$ whose corresponding contextualized embeddings are $\{\mathbf{e}_i, ..., \mathbf{e}_{j-1}\}$, the span representation $\mathbf{s}_{ij}$ is calculated as:

$$\mathbf{s}_{ij} = \sum_{k=i}^{j-1} a_k \cdot \mathbf{e}_k$$
$$a_k = \mathbf{Softmax}(\mathbf{v}_1^T \cdot \mathbf{e}_k)$$

where $\mathbf{v}_1$ is a learned parameter vector. Note that in a constituency parse tree, each constituent is attached with a nonterminal tag. We use an embedding layer to convert the tag to distributed vector space, and then concatenate it to $\mathbf{s}_{ij}$.

$$\mathbf{s}'_{ij} = \mathbf{Concat}([\mathbf{s}_{ij}; \mathbf{Embedding}(tag)])$$

We project $\mathbf{s}'_{ij}$ using two different learned matrices to obtain $\mathbf{h}_{ij}$ and $\mathbf{c}_{ij}$, the initial hidden state and memory cell of the node representing span $[i, j]$.

2.3 Composition within Hyperedge

Within each hyperedge, there are three nodes: parent (p), left child (l), right child (r). Traditional RvNN-based methods model the relationships among them mainly in a bottom-up manner: a composition function combines the representations of the two children to obtain the representation of the parent. Le and Zuidema (2014) extend such

1684

methods by adding a top-down process, where a node's representation can also be computed from its parent and sibling. We follow this idea that any node can be computed from the others within each hyperedge. The computation is direction-specific, meaning that the composition function is different for each of the three nodes.

Our composition function is inspired by TreeL-STM (Tai et al., 2015). The computing process can be denoted as:

$$[\mathbf{h}'_p; \mathbf{c}'_p] = \textbf{Compose}(\mathbf{h}_p, \mathbf{h}_l, \mathbf{h}_r, \mathbf{c}_l, \mathbf{c}_r) \quad (1)$$

$$[\mathbf{h}'_l; \mathbf{c}'_l] = \textbf{Compose}(\mathbf{h}_l, \mathbf{h}_p, \mathbf{h}_r, \mathbf{c}_p, \mathbf{c}_r) \quad (2)$$

$$[\mathbf{h}'_r; \mathbf{c}'_r] = \textbf{Compose}(\mathbf{h}_r, \mathbf{h}_p, \mathbf{h}_l, \mathbf{c}_p, \mathbf{c}_l) \quad (3)$$

where subscript p, l, r denote the parent, left child and right child respectively, $\mathbf{h}$ and $\mathbf{c}$ denote hidden state and memory cell vectors from the previous layer, and $\mathbf{h}'$ and $\mathbf{c}'$ denote newly composed vectors.

We illustrate the detailed computation process of equation (1), in which we compose the left child node and the right child node to get the representation of the parent node.

$$\mathbf{i} = \sigma\left(\mathbf{W}^{(i)}\mathbf{h}_p + \mathbf{U}^{(i)}_l\mathbf{h}_l + \mathbf{U}^{(i)}_r\mathbf{h}_r + \mathbf{b}^{(i)}\right)$$

$$\mathbf{f}^l = \sigma\left(\mathbf{W}^{(fl)}\mathbf{h}_p + \mathbf{U}^{(fl)}_l\mathbf{h}_l + \mathbf{U}^{(fl)}_r\mathbf{h}_r + \mathbf{b}^{(f)}\right)$$

$$\mathbf{f}^r = \sigma\left(\mathbf{W}^{(fr)}\mathbf{h}_p + \mathbf{U}^{(fr)}_l\mathbf{h}_l + \mathbf{U}^{(fr)}_r\mathbf{h}_r + \mathbf{b}^{(f)}\right)$$

$$\mathbf{o} = \sigma\left(\mathbf{W}^{(o)}\mathbf{h}_p + \mathbf{U}^{(o)}_l\mathbf{h}_l + \mathbf{U}^{(o)}_r\mathbf{h}_r + \mathbf{b}^{(o)}\right)$$

$$\mathbf{u} = \textbf{tanh}\left(\mathbf{W}^{(u)}\mathbf{h}_p + \mathbf{U}^{(u)}_l\mathbf{h}_l + \mathbf{U}^{(u)}_r\mathbf{h}_r + \mathbf{b}^{(u)}\right)$$

$$\mathbf{c}'_p = \mathbf{i} \odot \mathbf{u} + \mathbf{f}^l \odot \mathbf{c}_l + \mathbf{f}^r \odot \mathbf{c}_r$$

$$\mathbf{h}'_p = \mathbf{o} \odot \textbf{tanh}(\mathbf{c}'_p)$$

$\mathbf{i}, \mathbf{f}^l, \mathbf{f}^r, \mathbf{o} \in \mathbb{R}^d$ represent the input gate, two forget gates and output gate respectively. $\mathbf{u} \in \mathbb{R}^d$ is the newly composed input for the memory cell. $\mathbf{U}$ and $\mathbf{W} \in \mathbb{R}^{d \times d}$ denote trainable weight matrices and $\mathbf{b} \in \mathbb{R}^d$ denotes trainable bias vectors. The superscript of a matrix denotes which gate it is used for, and the subscript of a matrix denotes which node representation it is multiplied to. The computation processes of equation (2) and (3) are similarly defined, and they share some of the trainable matrices. (See more details in Appendix A.1)

2.4   Aggregation of Multiple Representations

With the exception of the root and the leaf nodes, a node in the HTNN is connected with two hyper-edges, one connecting to its parent and sibling and the other connecting to its two children. The computation process in Section 2.3 is independently carried out for each of the two hyperedges, resulting in two different representations for the node. Besides, we assume that the representation of the nodes from the previous layer also conveys important information. Therefore, we aggregate these three representations using the attention mechanism. Specifically, we use $\mathbf{h}'_0$ to denote the output hidden state from the previous layer, and $\mathbf{h}'_1$, $\mathbf{h}'_2$ to denote the representations computed from the two hyperedges. We use $\mathbf{h}'_0$ as the query vector, to attend to $\mathbf{h}'_0$, $\mathbf{h}'_1$, $\mathbf{h}'_2$ and compute a single aggregated representation $\mathbf{h}'$.

$$a_i = \textbf{Softmax}(\mathbf{v}^T_2 \textbf{tanh}(\mathbf{W}[\mathbf{h}'_0; \mathbf{h}'_i]))$$

$$\mathbf{h}' = \sum_i a_i \cdot \mathbf{h}'_i, \quad i \in \{0, 1, 2\}$$

where $\mathbf{W} \in \mathbb{R}^{2d \times d}$ and $\mathbf{v}_2 \in \mathbb{R}^d$ are trainable parameters. Note that the memory cell $\mathbf{c}'$ is aggregated in the same process with shared parameters.

3   Experiments

3.1   Implementation Details

We use gold constituency parse trees from the datasets, and binarize them using NLTK Toolkit (Bird et al., 2009). We use pretrained RoBERTa-large[2] (Liu et al., 2019) from HuggingFace (Wolf et al., 2019) as the word encoder. We also tried SpanBERT (Joshi et al., 2020) for word encoding but found it inferior to RoBERTa-large in our pilot experiments. We freeze the parameters in the pretrained language model for all experiments. The dimension of attention pooling is 192 and the dimension of tag embedding is 64. The dimensions of hidden state and memory cell in HTNN are all set to 256. We set the default number of HTNN layers to 3. For each layer, the number of training parameters is 1,707,520. During training, the model is evaluated on the validation set every 1000 training steps. We adopt the Adam optimizer with an initial learning rate of 2e-3, which is halved if the validation score is stuck for 3 evaluations. The batch size is 64 and the dropout probability is 0.2. We train our model for 40 epochs on 4 NVIDIA Tesla P40 GPUs. [3]

---

[2]https://huggingface.co/roberta-large
[3]The source code and trained model are available on https://github.com/GreyChou98/HTNN

3.2 Datasets

CoNLL-2012 We conduct both probing and SRL experiments on the CoNLL-2012 dataset (Pradhan et al., 2012). This dataset is extracted from the OntoNotes v5.0 corpus, which provides gold constituency parses.

CoNLL-2005 We also conduct SRL experiments on the CoNLL-2005 dataset (Carreras and Màrquez, 2005), which takes section 2-21 of the Wall Street Journal (WSJ) corpus as the training set, and section 24 as the development set. The test set consists of section 23 of WSJ for in-domain evaluation together with 3 sections from the Brown corpus for out-of-domain evaluation. CoNLL-2005 is also annotated with gold constituency parses.

3.3 Models for Comparison

Pooling A baseline model that computes span representations using attention pooling (Toshniwal et al., 2020). Our method and all the methods listed below use this representation for initialization.

SentiBERT A model for composing sentiment semantics on top of BERT (Yin et al., 2020). We re-implement their composition module based on the attention mechanism for span representation, instead of using their full model designed for sentiment analysis. We consider SentiBERT as a GNN-based model because it updates representations of all nodes simultaneously, though it only applies the attention mechanism from each parent node to its child nodes.

TreeLSTM An RvNN-based model, proposed by Tai et al. (2015), that generalizes the standard LSTM architecture to tree-structures.

Bi-TreeLSTM Le and Zuidema (2014) proposed IORNN to capture top-down decomposition besides bottom-up composition. We extend their model by using the composition function of TreeLSTMs. We refer to this model as bidirectional TreeLSTM (Bi-TreeLSTM).

GCN/GCN-sib Marcheggiani and Titov (2020) apply GCN structured with constituency trees in an SRL system. We re-implement their constituent GCN module to produce span representations, omitting their decomposition stage that is used for word representations. Furthermore, we extend GCN by adding sibling nodes in constituency trees for better comparison with our method. We refer to this modification as GCN-sib.

GAT/GAT-sib Li et al. (2020) use GAT to generate representations of the nodes in constituency parse trees. Similarly, we extend GAT by connecting sibling nodes, and we refer to this modification as GAT-sib.

3.4 Probing Experiments

We firstly conduct a probing evaluation of span representations, following Toshniwal et al. (2020). We conduct experiments on three of their probing tasks, omitting the other two tasks that probe syntax because constituency syntactic parses are input to most of the methods compared here.

- Named entity labeling (NEL): a task of predicting the entity type of a given span corresponding to an entity.
- Semantic role classification (SRC): a task of predicting the semantic roles of phrases in a sentence. Unlike the standard SRL task, in this probing task, predicates and their argument spans are given , so the goal is to classify the argument spans into specific semantic roles.
- Coreference arc prediction (COREF): a task of predicting whether a pair of spans refer to the same entity.

The task-specific decoder for these probing tasks is a two-layer MLP followed by sigmoid layers to predict the labels. For SRC and COREF, which involve two spans, the MLP takes the concatenation of the representations of the two spans as input. Following Toshniwal et al. (2020), we make predictions for each label independently, which allows using the micro-averaged F1 score as the evaluation metric. Note that in all the methods except Pooling, we only compute representations for constituent spans (those appearing in constituency parse trees). Therefore, given a distituent span, we simply predict no label. Fortunately, almost all the spans of interest in three tasks are constituent spans (98.5% for NEL, 99.7% for SRC, and 99.4% for COREF)[4]. We report F1 scores for all the spans (F1-all) as well as for constituent spans only (F1-const).

The results of three probing tasks are shown in Table 1. As a baseline model, Pooling's architecture is the simplest. Surprisingly, it outperforms all the other methods (except for HTNN on F1-const) on the NEL task. We believe this is because NEL is so simple that contextual word representations

---

[4]See more details of the statistics in Appendix A.2

	NEL		SRC		COREF		AVG	
	F1-const	F1-all	F1-const	F1-all	F1-const	F1-all	F1-const	F1-all
Pooling	96.18	**96.07**	93.08	93.05	92.99	93.01	94.08	94.04
TreeLSTM	95.05	94.22	90.02	89.88	90.07	89.70	91.71	91.27
Bi-TreeLSTM	95.25	94.42	90.49	90.35	90.33	89.96	92.02	91.58
SentiBERT	92.98	92.84	95.92	95.09	96.01	95.64	94.97	94.52
GAT	96.01	95.18	93.41	93.26	96.13	95.76	95.18	94.73
GCN	96.03	95.20	93.51	93.37	96.22	95.85	95.25	94.81
GAT-sib	95.79	94.96	92.85	92.71	95.66	95.29	94.77	94.32
GCN-sib	95.87	95.04	93.27	93.13	95.68	95.31	94.94	94.50
HTNN	**96.28**	95.45	**93.88**	**93.74**	**96.33**	**95.96**	**95.50**	**95.05**

Table 1: Results of probing experiments.

already provide sufficient information. More complicated models show poor performance possibly because they overfit this simple task. On the other hand, Pooling underperforms many other methods on the SRC and COREF tasks, showing the utility of syntactic information. For COREF in particular, many input spans are pronouns and their meanings may be more difficult to capture without syntactic information.

The RvNN-based models TreeLSTM and Bi-TreeLSTM perform worst on all these three tasks. Previous recursive models are mostly used for sentence-level tasks, and our results show their deficiencies in composing semantics for fine-grained spans rather than sentences.

For GNN-based models, SentiBERT, GAT and GCN all show great improvement over recursive models, and they are also better than Pooling on average on these three tasks. Both GAT and GCN show slight superiority over SentiBERT, probably because they take parent node into computation, rather than only children nodes. Moreover, GCN performs slightly better than GAT.

As for HTNN, we can see that it achieves the best performance overall. The fact that HTNN outperforms GAT and GCN while GAT-sib and GCN-sib underperform GAT and GCN suggests that the superiority of HTNN does not simply come from adding sibling information into computation. It is the ability to group and integrate bottom-up and top-down information that leads to the superior performance of HTNN.

It is worth noting that for the NEL task, HTNN outperforms Pooling on F1-const but underperforms it on F1-all because HTNN predicts no label for distituent spans. A simple remedy is to combine HTNN and Pooling, using HTNN for constituent spans and reverting to Pooling for distituent spans. This yields 96.27 F1-all for NEL, surpassing the strongest Pooling baseline.

3.5 SRL Experiments

We extend SRL in probing tasks to the general setting, in which the predicate is given and we conduct argument identification and role classification. Our SRL model is most closely related to the work of He et al. (2018). The difference is that instead of using a separate argument pruning module, we simply prune all the distituent spans and keep all the constituent spans. Given a predicate and a candidate span, we use their biaffine scorer to predict a label, where the label set is the set of semantic roles plus a null label indicating the span not being an argument.

The results of SRL experiments are shown in Table 2. Once again, we show F1 scores for both all the spans (F1-all) and constituent spans only (F1-const) . We only show the results of GNN-based models since RvNN-based models have shown their disadvantages in the probing experiments. SentiBERT does not perform well in this task, falling behind even the simple baseline Pooling. GCN and GAT show significant gain against baseline, and GCN is still better than GAT on all the three datasets, especially on the out-of-domain BROWN test dataset. We also find that taking sibling nodes into computation causes opposite effects on GAT and GCN. It does great harm to GAT, similar to the case in the probing experiments, but GCN-sib shows small gains over GCN, which is different from the case in the probing experiments.

Our model HTNN achieves the best performance on all the three datasets. Since SRL is more challenging than the probing tasks, we can see that the

	CONLL12		CONLL05 WSJ		CONLL05 BROWN	
	F1-const	F1-all	F1-const	F1-all	F1-const	F1-all
Pooling	82.36	82.23	82.82	81.90	71.51	70.43
SentiBERT	75.31	75.18	74.52	73.69	66.52	65.51
GAT	85.29	85.16	84.69	83.74	73.32	72.24
GCN	87.91	87.77	88.06	87.08	79.22	78.09
GAT-sib	76.41	76.29	78.34	77.46	62.52	61.58
GCN-sib	88.40	88.27	88.45	87.46	80.03	79.87
HTNN	**89.94**	**89.81**	**90.77**	**89.76**	**82.88**	**81.68**
Wang et al. (2019)†	-	84.21	-	85.23	-	75.36
Fei et al. (2021)*	-	87.35	-	88.81	-	81.27

Table 2: Results of SRL experiments. "†": reimplemented and reported by Fei et al. (2021). "∗": results reported in the original paper.

	SRC	NEL	COREF	CONLL12	WSJ	BROWN
GAT -w/o tag	-0.27	-0.14	-0.46	-7.74	-8.49	-7.36
GCN -w/o tag	-0.37	0.02	-0.09	-1.74	-1.38	-0.76
GAT-sib -w/o tag	-0.82	-0.34	-1.85	-7.09	-9.30	-7.05
GCN-sib -w/o tag	-0.07	0.06	0.02	-2.06	-1.90	-1.11
HTNN -w/o tag	-0.04	-0.07	0.13	-1.73	-2.29	-2.28
HTNN -tag-dim=32	0.07	-0.14	0.22	-0.03	0.00	0.15

Table 3: Influence of nonterminal tags in constituency parse tree. The value in this table is the variation against the resuts in Table 1 and Table 2.

gap between HTNN and the other models is more prominet. The improvements over the best baseline model GCN-sib are 1.54 on the CONLL-2012 test dataset, 2.32 on the CONLL-2005 WSJ test dataset, 2.85 on the CONLL-2005 BROWN test dataset.

We also show results of two other SRL models that use gold constituency parse trees in Table 2. HTNN outperforms both of them.

3.6 Ablation Study

Incorporation of Nonterminal Tags We incorporate nonterminal tags into HTNN as described in Section 2.2. Here, we want to explore how nonterminal tags influence HTNN, as well as other models. When we use different embedding dimensions for nonterminal tags, the dimension of the attention pooling module changes accordingly to keep the dimension of span representations unchanged.

The results are shown in Table 3. As we can see, GAT, GCN and HTNN are negatively affected on nearly all the tasks when we remove the incorporation of nonterminal tags. The influence on both GCN and HTNN is quite small on the three probing tasks, compared with the SRL task. There are even improvements for HTNN and GCN-sib when
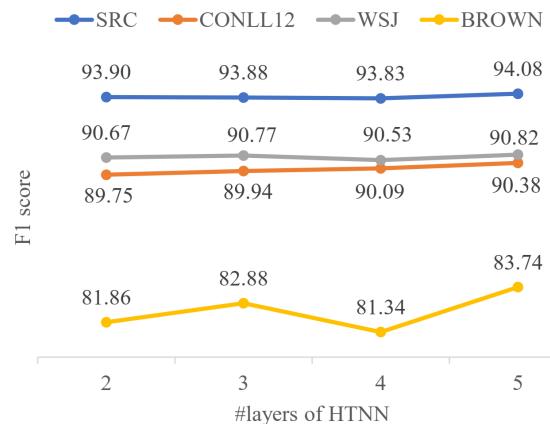

Figure 3: Performance of different tasks as #layers of HTNN changes.

removing nonterminal tags on NEL and COREF. However, on the SRL task, the performance decreases notably for all the models.

We also conduct experiments of HTNN when we set the dimension of nonterminal tags to 32 and we observe little change in performance.

Number of Layers In each layer of an HTNN, a node exchanges information with its neighbor. Therefore, the more layers the HTNN has, the fur-
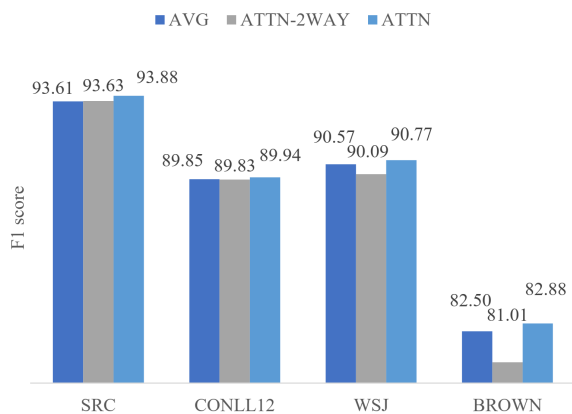
Figure 4: Performance of different aggregation methods.

ther the information of a node would spread. In theory, for a binary constituency tree with n leaf nodes, to make each leaf node receive information from arbitrary leaf nodes, the number of HTNN layers should be set to $n - 1$. In practice, however, this may be unnecessary and local information exchange may be sufficient. We conduct experiments to analyze how the number of HTNN layers affect the performance on different tasks.

The results are shown in Figure 3. We can see that the tendency is not all the same on the four tasks. As we increase the number of layers, the increase of the F1 score is getting smaller, and in some cases it even declines. Therefore, we regard the number of layers as a hyperparameter which should be tuned on specific tasks.

Aggregation Method We compare three different methods for the aggregation module (Section 2.4):

- AVG, which simply takes the average of the representations.
- ATTN, which is the default method we introduce in Section 2.4.
- ATTN-2WAY, similar to ATTN, but does not include the representation from previous layer.

The results are shown in Figure 4. It can be seen that ATTN indeed performs the best.

4 Related Work

To get distributed span representations, existing methods are mostly based on simple derivations from word or sub-word representations. Lee et al. (2017) propose the attention-weighted pooling strategy for coreference resolution. Stern et al. (2017) concatenate the sum and the difference of

the span endpoints for parsing. Seo et al. (2019) use the "coherent" endpoint-based representation for question answering. All of these methods are originally designed for specific tasks, but because of their simplicity, they can be easily employed for other tasks. Toshniwal et al. (2020) make a comprehensive survey on these methods and conduct experiments to compare their performance. Some other studies also focus on the representations of sentences or longer texts (Wieting et al., 2015; Conneau et al., 2017; Shen et al., 2018).

Instead of representing an arbitrary, some studies consider composing span representations via syntactic parse trees, which are more related to our work. Le and Zuidema (2014) proposed a recursive model IORNN to produce constituent representations for supervised task. Drozdov et al. (2019) proposed an unsupervised method for constituency parse tree induction, during which span representations are produced as a byproduct. More recently, graph neural networks have been applied with constituency parse trees. Marcheggiani and Titov (2020) use GCN to encode constituent spans in an SRL system. However, they decompose the constituent span representations back to start and end tokens instead of directly using them. Li et al. (2020) use GAT to generate constituent span representations for the sentiment analysis task.

5 Conclusion and Future Work

In this paper, we propose hypertree neural networks (HTNN) to generate better representations of constituent spans following constituency parse tree structures. Each node in the HTNN represents a constituent span and each hyperedge represents a local composition structure. Each HTNN layer first computes a representation of each node in a hyperedge using the other nodes from the same hyperedge, and then aggregates the representations of a node from all the connected hyperedges. We empirically compare HTNNs with previous RvNN-based models and GNN-based models. The outstanding performance of HTNNs on both probing and SRL tasks shows the effectiveness of HTNN.

For future work, we plan to tackle two related issues of our approach, namely its reliance on high-quality constituency parses and its inability to represent distituent spans (some of which may nonetheless be important). We will also extend HTNNs to more tasks such as sentiment analysis and document classification, and incorporate HTNNs into

popular pretrained language models as a method to inject syntactic information.

Acknowledgments

References

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.".

Xavier Carreras and Lluís Màrquez. 2005. Introduction to the CoNLL-2005 shared task: Semantic role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 152–164, Ann Arbor, Michigan. Association for Computational Linguistics.

Noam Chomsky. 1957. Syntactic structures.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680, Copenhagen, Denmark. Association for Computational Linguistics.

Andrew Drozdov, Patrick Verga, Mohit Yadav, Mohit Iyyer, and Andrew McCallum. 2019. Unsupervised latent tree induction with deep inside-outside recursive auto-encoders. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1129–1141, Minneapolis, Minnesota. Association for Computational Linguistics.

Hao Fei, Shengqiong Wu, Yafeng Ren, Fei Li, and Donghong Ji. 2021. Better combine them together! integrating syntactic constituency and dependency representations for semantic role labeling. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 549–559, Online. Association for Computational Linguistics.

Christoph Goller and Andreas Kuchler. 1996. Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 1, pages 347–352. IEEE.

Luheng He, Kenton Lee, Omer Levy, and Luke Zettlemoyer. 2018. Jointly predicting predicates and arguments in neural semantic role labeling. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 364–369, Melbourne, Australia. Association for Computational Linguistics.

Liang Huang and David Chiang. 2005. Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 53–64, Vancouver, British Columbia. Association for Computational Linguistics.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. 2020. SpanBERT: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77.

Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Dan Klein and Christopher D. Manning. 2001. Parsing and hypergraphs. In *Proceedings of the Seventh International Workshop on Parsing Technologies*, pages 123–134, Beijing, China.

Yuxuan Lai, Yijia Liu, Yansong Feng, Songfang Huang, and Dongyan Zhao. 2021. Lattice-BERT: Leveraging multi-granularity representations in Chinese pretrained language models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1716–1731, Online. Association for Computational Linguistics.

Phong Le and Willem Zuidema. 2014. Inside-outside semantics: A framework for neural models of semantic composition. In *NIPS 2014 Workshop on Deep Learning and Representation Learning*.

Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. 2017. End-to-end neural coreference resolution. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 188–197, Copenhagen, Denmark. Association for Computational Linguistics.

Yuncong Li, Cunxiang Yin, and Sheng-hua Zhong. 2020. Sentence constituent-aware aspect-category sentiment analysis with graph attention networks. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pages 815–827. Springer.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Diego Marcheggiani and Ivan Titov. 2020. Graph convolutions over constituent trees for syntax-aware semantic role labeling. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3915–3928, Online. Association for Computational Linguistics.

Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang. 2012. CoNLL-2012 shared task: Modeling multilingual unrestricted coreference in OntoNotes. In *Joint Conference on EMNLP and CoNLL - Shared Task*, pages 1–40, Jeju Island, Korea. Association for Computational Linguistics.

Minjoon Seo, Jinhyuk Lee, Tom Kwiatkowski, Ankur Parikh, Ali Farhadi, and Hannaneh Hajishirzi. 2019. Real-time open-domain question answering with dense-sparse phrase index. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4430–4441, Florence, Italy. Association for Computational Linguistics.

Dinghan Shen, Guoyin Wang, Wenlin Wang, Martin Renqiang Min, Qinliang Su, Yizhe Zhang, Chunyuan Li, Ricardo Henao, and Lawrence Carin. 2018. Baseline needs more love: On simple word-embedding-based models and associated pooling mechanisms. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450, Melbourne, Australia. Association for Computational Linguistics.

Richard Socher, Cliff Chiung-Yu Lin, Andrew Y Ng, and Christopher D Manning. 2011. Parsing natural scenes and natural language with recursive neural networks. In *ICML*.

Mitchell Stern, Jacob Andreas, and Dan Klein. 2017. A minimal span-based neural constituency parser. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 818–827, Vancouver, Canada. Association for Computational Linguistics.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, Beijing, China. Association for Computational Linguistics.

Shubham Toshniwal, Haoyue Shi, Bowen Shi, Lingyu Gao, Karen Livescu, and Kevin Gimpel. 2020. A cross-task analysis of text span representations. In *Proceedings of the 5th Workshop on Representation Learning for NLP*, pages 166–176, Online. Association for Computational Linguistics.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Yufei Wang, Mark Johnson, Stephen Wan, Yifang Sun, and Wei Wang. 2019. How to best use syntax in semantic role labelling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5338–5343, Florence, Italy. Association for Computational Linguistics.

John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2015. Towards universal paraphrastic sentence embeddings. *arXiv preprint arXiv:1511.08198*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Da Yin, Tao Meng, and Kai-Wei Chang. 2020. SentiB-ERT: A transferable transformer-based architecture for compositional sentiment semantics. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3695–3706, Online. Association for Computational Linguistics.

Lang Yu and Allyson Ettinger. 2020. Assessing phrasal representation and composition in transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4896–4907, Online. Association for Computational Linguistics.

Xinsong Zhang, Pengshuai Li, and Hang Li. 2021. AM-BERT: A pre-trained language model with multi-grained tokenization. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 421–435, Online. Association for Computational Linguistics.

A Appendix

A.1 Details of Composition

The following is the detailed computation process of equation (2), in which we compose the right child node and the parent node to get the representation of the left child node:

$$\mathbf{i} = \sigma\left(\mathbf{W}^{(i)}\mathbf{h}_r + \mathbf{U}_l^{(i)}\mathbf{h}_l + \mathbf{U}_p^{(i)}\mathbf{h}_p + \mathbf{b}^{(i)}\right)$$

$$\mathbf{f}^l = \sigma\left(\mathbf{W}^{(fl)}\mathbf{h}_r + \mathbf{U}_l^{(fl)}\mathbf{h}_l + \mathbf{U}_p^{(fl)}\mathbf{h}_p + \mathbf{b}^{(f)}\right)$$

$$\mathbf{f}^p = \sigma\left(\mathbf{W}^{(fp)}\mathbf{h}_r + \mathbf{U}_l^{(fp)}\mathbf{h}_l + \mathbf{U}_p^{(fp)}\mathbf{h}_p + \mathbf{b}^{(f)}\right)$$

$$\mathbf{o} = \sigma\left(\mathbf{W}^{(o)}\mathbf{h}_r + \mathbf{U}_l^{(o)}\mathbf{h}_l + \mathbf{U}_p^{(o)}\mathbf{h}_p + \mathbf{b}^{(o)}\right)$$

$$\mathbf{u} = \tanh\left(\mathbf{W}^{(u)}\mathbf{h}_r + \mathbf{U}_l^{(u)}\mathbf{h}_l + \mathbf{U}_p^{(u)}\mathbf{h}_p + \mathbf{b}^{(u)}\right)$$

The following is the detailed computation process of equation (3), in which we compose the left child node and the parent node to get the represen-

Task	$|\mathcal{L}|$	#Instances (Train / Val. / Test)	#Missing (Train / Val. / Test)
NEL	18	128738 / 20354 / 12586	1860 / 297 / 215
SRC	66	598983 / 83362 / 61716	1508 / 281 / 185
COREF	2	207830 / 26333 / 27800	1156 / 286 / 213
CoNLL-2005	53	235353 / 8549 / (14463 / 2228)	5897 / 225 / (315 / 63)

Table 4: Statistics of the CoNLL-2012 and CoNLL-2005 datasets. The NEL, SRC and COREF are from the CoNLL-2012 dataset. "$|\mathcal{L}|$" is the number of labels. "#Instances" means the total instances in the original datasets. "#Missing" means the instances that contain distituent spans. Note that for the CoNLL-2005 dataset, the test set is composed of WSJ and BROWN, so we show their numbers respectively.

tation of the right child node:

$$\mathbf{i} = \sigma \left(\mathbf{W}^{(i)}\mathbf{h}_l + \mathbf{U}_r^{(i)}\mathbf{h}_r + \mathbf{U}_p^{(i)}\mathbf{h}_p + \mathbf{b}^{(i)} \right)$$

$$\mathbf{f}^r = \sigma \left(\mathbf{W}^{(fr)}\mathbf{h}_l + \mathbf{U}_r^{(fr)}\mathbf{h}_r + \mathbf{U}_p^{(fr)}\mathbf{h}_p + \mathbf{b}^{(f)} \right)$$

$$\mathbf{f}^p = \sigma \left(\mathbf{W}^{(fp)}\mathbf{h}_l + \mathbf{U}_r^{(fp)}\mathbf{h}_r + \mathbf{U}_p^{(fp)}\mathbf{h}_p + \mathbf{b}^{(f)} \right)$$

$$\mathbf{o} = \sigma \left(\mathbf{W}^{(o)}\mathbf{h}_l + \mathbf{U}_r^{(o)}\mathbf{h}_r + \mathbf{U}_p^{(o)}\mathbf{h}_p + \mathbf{b}^{(o)} \right)$$

$$\mathbf{u} = \tanh \left(\mathbf{W}^{(u)}\mathbf{h}_l + \mathbf{U}_r^{(u)}\mathbf{h}_r + \mathbf{U}_p^{(u)}\mathbf{h}_p + \mathbf{b}^{(u)} \right)$$

A.2 Dataset Statistics

We show the statistics of the datasets in Table 4.