Syntax-aware Neural Semantic Role Labeling*

Qingrong Xia[1], Zhenghua Li[1], Min Zhang[1], Meishan Zhang[2], Guohong Fu[2], Rui Wang[3], Luo Si[3]
[1]Institute of Artificial Intelligence, School of Computer Science and Technology, Soochow University, China
[2]School of Computer Science and Technology, Heilongjiang University, China, [3]Alibaba Group, China
[1]kirosummer.nlp@gmail.com, {zhli13, minzhang}@suda.edu.cn
[2]mason.zms@gmail.com, ghfu@hotmail.com, [3]{masi.wr, luo.si}@alibaba-inc.com

Abstract

Semantic role labeling (SRL), also known as shallow semantic parsing, is an important yet challenging task in NLP. Motivated by the close correlation between syntactic and semantic structures, traditional discrete-feature-based SRL approaches make heavy use of syntactic features. In contrast, deep-neural-network-based approaches usually encode the input sentence as a word sequence without considering the syntactic structures. In this work, we investigate several previous approaches for encoding syntactic trees, and make a thorough study on whether extra syntax-aware representations are beneficial for neural SRL models. Experiments on the benchmark CoNLL-2005 dataset show that syntax-aware SRL approaches can effectively improve performance over a strong baseline with external word representations from ELMo. With the extra syntax-aware representations, our approaches achieve new state-of-the-art 85.6 F1 (single model) and 86.6 F1 (ensemble) on the test data, outperforming the corresponding strong baselines with ELMo by 0.8 and 1.0, respectively. Detailed error analysis are conducted to gain more insights on the investigated approaches.

Introduction

Semantic role labeling (SRL), also known as shallow semantic parsing, is an important yet challenging task in NLP. Given an input sentence and one or more predicates, SRL aims to determine the semantic roles of each predicate, i.e., who did what to whom, when and where, etc. Semantic knowledge has been proved informative in many downstream NLP applications, such as question answering (Shen and Lapata, 2007; Wang et al., 2015), text summarization (Genest and Lapalme, 2011; Khan, Salim, and Jaya Kumar, 2015), and machine translation (Liu and Gildea, 2010; Gao and Vogel, 2011).

Depending on how the semantic roles are defined, there are two forms of SRL in the community. The span-based SRL follows the manual annotations in the PropBank (Palmer, Gildea, and Kingsbury, 2005) and NomBank (Meyers et al., 2004) and uses a continuous word span to be a semantic role. In contrast, the dependency-based SRL fulfills a role with a single word, which is usually the syntactic or semantic head of the manually annotated span (Surdeanu et al., 2008).

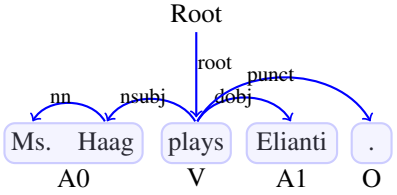

Figure 1: Example of dependency and SRL structures.

This work follows the span-based formulation. Formally, given an input sentence $\boldsymbol{w} = w_1 ... w_n$ and a predicative word $\texttt{prd} = w_p$ $(1 \leq p \leq n)$, the task is to recognize the semantic roles of $\texttt{prd}$ in the sentence, such as A0, A1, AM-ADV, etc. We denote the whole role set as $\mathcal{R}$. Each role corresponds to a word span of $w_j ... w_k$ $(1 \leq j \leq k \leq n)$. Taking Figure 1 as an example,"Ms. Hag" is the A0 role of the predicate "plays".

In the past few years, thanks to the success of deep learning, researchers has proposed effective neural-network-based models and improved SRL performance by large margins (Zhou and Xu, 2015; He et al., 2017; Tan et al., 2018). Unlike traditional discrete-feature-based approaches that make heavy use of syntactic features, recent deep-neural-network-based approaches are mostly in an end-to-end fashion and give little consideration of syntactic knowledge.

Intuitively, syntax is strongly correlative with semantics. Taking Figure 1 as an example, the *A0* role in the SRL structure is also the subject (marked by *nsubj*) in the dependency tree, and the *A1* role is the direct object (marked by *dobj*). In fact, the semantic *A0* or *A1* argument of a verb predicate are usually the syntactic subject or object interchangeably according to the PropBank annotation guideline.

In this work, we investigate several previous approaches for encoding syntactic trees, and make a thorough study on whether extra syntax-aware representations are beneficial for neural SRL models. The four approaches, Tree-GRU, Shortest Dependency Path (SDP), Tree-based Position Feature (TPF), and Pattern Embedding (PE), try to encode useful syntactic information in the input dependency tree from different perspectives. Then, we use the encoded syntax-

*Zhenghua Li is the corresponding Author.

aware representation vectors as extra input word representations, requiring little change of the architecture of the basic SRL model.

For the base SRL model, we employ the recently proposed deep highway-BiLSTM model (He et al., 2017). Considering that the quality of the parsing results has great impact on the performance of syntax-aware SRL models, we employ the state-of-the-art biaffine parser to parse all the data in our work, which achieves 94.3% labeled parsing accuracy on the WSJ test data (Dozat and Manning, 2017).

We conduct our experiments on the benchmark CoNLL-2005 dataset, comparing our syntax-aware SRL approaches with a strong baseline with external word representations from ELMo. Detailed error analyses also give us more insights on the investigated approaches. The results show that, with the extra syntax-aware representations, our approach achieves new state-of-the-art 85.6 F1 (single model) and 86.6 F1 (ensemble) on the test set, outperforming the corresponding strong baselines with ELMo by 0.8 and 1.0, respectively, demonstrating the usefulness of syntactic knowledge.

The Basic SRL Architecture

Following previous works (Zhou and Xu, 2015; He et al., 2017; Tan et al., 2018), we also treat the task as a sequence labeling problem and try to find the highest-scoring tag sequence $\hat{\boldsymbol{y}}$.

$$\hat{\boldsymbol{y}} = \underset{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{w})}{\operatorname{argmax}}\, score(\boldsymbol{w}, \boldsymbol{y}) \quad (1)$$

where $y_i \in \mathcal{R}'$ is the tag of the i-th word w_i, and $\mathcal{Y}(\boldsymbol{w})$ is the set of all legal sequences. Please note that $\mathcal{R}' = (\{B, I\} \times \mathcal{R}) \cup \{O\}$.

In order to compute $score(\boldsymbol{w}, \boldsymbol{y})$, which is the score of a tag sequence $\boldsymbol{y}$ for $\boldsymbol{w}$, we directly adopt the architecture of He et al. (2017), which consists of the following four components, as illustrated in Figure 2.

The Input Layer Given the sentence $\boldsymbol{w} = w_1 ... w_n$ and the predicate $\texttt{prd} = w_p$, the input of the network is the combination of the word embeddings and the predicate-indicator embeddings. Specifically, the input vector at the i-th time stamp is

$$\boldsymbol{x}_i = \boldsymbol{emb}^{\texttt{word}}_{w_i} \oplus \boldsymbol{emb}^{\texttt{prd}}_{i==p} \quad (2)$$

where the predicate-indicator embedding $\boldsymbol{emb}^{\texttt{prd}}_0$ is used for non-predicate positions and $\boldsymbol{emb}^{\texttt{prd}}_1$ is used for the p-th position, in order to distinguish the predicate word from other words, as shown in Figure 2.

With the predicate-indicator embedding, the encoder component can represent the sentence in a predicate-specific way, leading to superior performance (Zhou and Xu, 2015; He et al., 2017; Tan et al., 2018). However, the side effect is that we need to separately encode the sentence for each predicate, dramatically slowing down training and evaluation.

The BiLSTM Encoding Layer Over the input layer, four stacking layers of BiLSTMs are applied to fully encode long-distance dependencies in the sentence and obtain the rich predicate-specific token-level representations.

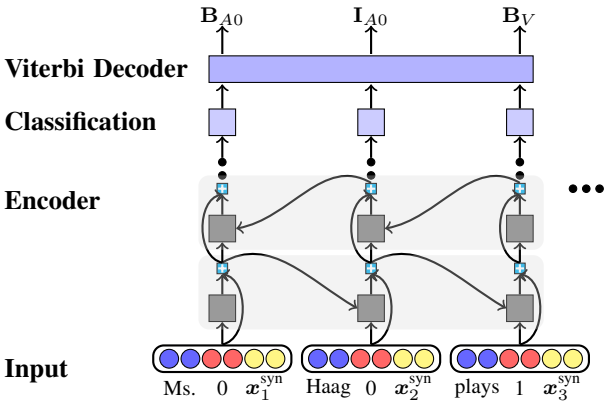

Figure 2: The basic SRL architecture.

Moreover, He et al. (2017) propose to use highway connections (Srivastava, Greff, and Schmidhuber, 2015; Zhang et al., 2016) to alleviate the vanishing gradient problem, improving the parsing performance by 2% F1. As illustrated in Figure 2, the basic idea is to combine the input and output of an LSTM node in some way, and feed the combined result as the final output of the node into the next LSTM layer and the next time-stamp of the same LSTM layer.

We use the outputs of the final (top) backward LSTM layer as the representation of each word, denoted as $\boldsymbol{h}_i$.

Classification Layer With the representation vector $\boldsymbol{h}_i$ of the word w_i, we employ a linear transformation and a softmax operation to compute the probability distribution of different tags, denoted as $p(r|\boldsymbol{w}, i)$ $(r \in \mathcal{R}')$.

Decoder With the local tag probabilities of each word, then the score of a tag sequence is

$$score(\boldsymbol{w}, \boldsymbol{y}) = \sum_{i=1}^{n} \log p(y_i|\boldsymbol{w}, i) \quad (3)$$

Finally, we employ the Viterbi algorithm to find the highest-scoring tag sequence and ensure the resulting sequence does not contain illegal tag transitions such as $y_{i-1} = B_{A0}$ and $y_i = I_{A1}$.

The Syntax-aware SRL Approaches

The previous section introduces the basic model architecture of SRL, and in this section, we will illustrate how to encode the syntactic features into a dense vector and use it as extra inputs. Intuitively, dependency syntax has a strong correlation with semantics. For instance, a subject of a verb in dependency trees usually corresponds to the agent or patient of the verb. Therefore, traditional discrete-feature based SRL approaches make heavy use of syntax-related features. In contrast, the state-of-the-art neural network based SRL models usually adopt the end-to-end framework without consulting the syntax.

This work tries to make a thorough investigation on whether integrating syntactic knowledge is beneficial for state-of-the-art neural network based SRL approaches. We

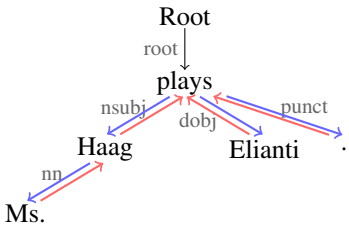

Figure 3: Bi-directional (bottom-up and top-down) Tree-GRU.

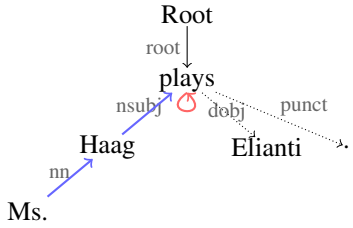

Figure 4: The SDP approach: where the blue line marks the path from the focused word "*Ms.*" to the common ancestor "*plays*", and the red line is the path from the predicate to the ancestor.

investigate and compare four different approaches for encoding syntactic trees.

- The *Tree-GRU* (tree-structured gated recurrent unit) approach globally encodes an entire dependency tree once for all, and produces unified representations of all words in a predicate-independent way.
- The *SDP* (shortest dependency path) approach considers the shortest path from the focused word w_i and the predicated w_p, and use the max-pooling of the syntactic labels in the path as the predicate-specific (different from Tree-GRU) syntax-aware representation of w_i.
- The *TPF* (tree position feature) approach considers the relative positions of w_i and w_p to their common ancestor word in the parse tree, and use the embedding of such position features as the representation of w_i.
- The *PE* (pattern embedding) approach classifies the relationship between w_i and w_p in the parse tree into several pre-defined patterns (or types), and use the embeddings of the pattern and a few dependency relations as the representation of w_i.

The four approaches encode dependency trees from different perspectives and in different ways. Each approach produces a syntax-aware representation of the focused word.

Formally, we denote these representations as $\boldsymbol{x}_i^{\text{syn}}$ for word w_i. We treat these syntactic representations as the external model input, and concatenate them with the basic input $\boldsymbol{x}_i$. In the following, we introduce the four approaches in detail.

The Tree-GRU Approach

As a straightforward method, tree-stuctured recurrent neural network (Tree-RNN) (Tai, Socher, and Manning, 2015; Chen et al., 2017) can globally encode a parse tree and return syntax-aware representation vectors of all words in the sentence. Previous works have successfully employed Tree-RNN for exploiting syntactic parse trees for different tasks, such as sentiment classification (Tai, Socher, and Manning, 2015), relation extraction (Miwa and Bansal, 2016; Feng et al., 2017) and machine translation (Chen et al., 2017; Wang et al., 2018).

Following Chen et al. (2017), we employ a bi-directional Tree-GRU to encode the dependency tree of the input sentence, as illustrated in Figure 3.

The bottom-up Tree-GRU computes the representation vector $\boldsymbol{h}_i^{\uparrow}$ of the word w_i based on its children, as marked by the red lines in Figure 3. The detailed equations are as follows.

$$
\begin{aligned}
\bar{\boldsymbol{h}}_{i,L}^{\uparrow} &= \sum_{j \in lchild(i)} \boldsymbol{h}_j^{\uparrow}, \qquad \bar{\boldsymbol{h}}_{i,R}^{\uparrow} = \sum_{k \in rchild(i)} \boldsymbol{h}_k^{\uparrow} \\
\boldsymbol{r}_{i,L} &= \sigma(\mathbf{W}^{rL}\boldsymbol{l}_i + \mathbf{U}^{rL}\bar{\boldsymbol{h}}_{i,L}^{\uparrow} + \mathbf{V}^{rL}\bar{\boldsymbol{h}}_{i,R}^{\uparrow}) \\
\boldsymbol{r}_{i,R} &= \sigma(\mathbf{W}^{rR}\boldsymbol{l}_i + \mathbf{U}^{rR}\bar{\boldsymbol{h}}_{i,L}^{\uparrow} + \mathbf{V}^{rR}\bar{\boldsymbol{h}}_{i,R}^{\uparrow}) \\
\boldsymbol{z}_{i,L} &= \sigma(\mathbf{W}^{zL}\boldsymbol{l}_i + \mathbf{U}^{zL}\bar{\boldsymbol{h}}_{i,L}^{\uparrow} + \mathbf{V}^{zL}\bar{\boldsymbol{h}}_{i,R}^{\uparrow}) \\
\boldsymbol{z}_{i,R} &= \sigma(\mathbf{W}^{zR}\boldsymbol{l}_i + \mathbf{U}^{zR}\bar{\boldsymbol{h}}_{i,L}^{\uparrow} + \mathbf{V}^{zR}\bar{\boldsymbol{h}}_{i,R}^{\uparrow}) \\
\boldsymbol{z}_i &= \sigma(\mathbf{W}^{z}\boldsymbol{l}_i + \mathbf{U}^{z}\bar{\boldsymbol{h}}_{i,L}^{\uparrow} + \mathbf{V}^{z}\bar{\boldsymbol{h}}_{i,R}^{\uparrow}) \\
\hat{\boldsymbol{h}}_i^{\uparrow} &= \tanh\big(\mathbf{W}\boldsymbol{l}_i + \mathbf{U}(\boldsymbol{r}_{i,L} \odot \bar{\boldsymbol{h}}_{i,L}^{\uparrow}) + \mathbf{V}(\boldsymbol{r}_{i,R} \odot \bar{\boldsymbol{h}}_{i,R}^{\uparrow})\big) \\
\boldsymbol{h}_i^{\uparrow} &= \boldsymbol{z}_{i,L} \odot \bar{\boldsymbol{h}}_{i,L}^{\uparrow} + \boldsymbol{z}_{i,R} \odot \bar{\boldsymbol{h}}_{i,R}^{\uparrow} + \boldsymbol{z}_i \odot \hat{\boldsymbol{h}}_i^{\uparrow},
\end{aligned}
\tag{4}
$$

where $\texttt{lchild/rchild}(.)$ means the set of left/right-side children, $\boldsymbol{l}_i$ is the embedding of the syntactic label between w_i and its head, and $\mathbf{W}$s, $\mathbf{U}$s and $\mathbf{V}$s are all model parameters.

Analogously, the top-down Tree-GRU computes the representation vector $\boldsymbol{h}_i^{\downarrow}$ of the word w_i based on its parent node, as marked by the blue lines in Figure 3. We omit the equations for brevity.

We use the concatenation of the two resulting hidden vectors as the the final representation of the word w_i:

$$
\boldsymbol{x}_i^{\text{syn}} = \boldsymbol{h}_i^{\uparrow} \oplus \boldsymbol{h}_i^{\downarrow} \tag{5}
$$

The SDP Approach

SDP-based features have been extensively used in traditional discrete-feature based approaches for exploiting syntactic knowledge in relation extraction. The idea is to consider the shortest path of two focused words in the dependency tree, and extract path-related features as syntactic clues.

Xu et al. (2015) first adapt SDP into the nerual network settings in the task of relation classification. They treat the SDP as two sub-paths from the two focused entity words to their lowest common ancestor, and run two LSTMs respectively along the two sub-paths to obtain extra syntax-aware representations, leading to improved performance.

Directly employing LSTMs on SDPs leads to prohibitively efficiency problem in our scenario, because we

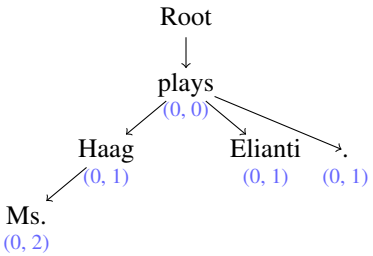

Figure 5: Example of Tree-based Position Feature. The number-tuples with brackets are relative positions of TPF.

have $O(n)$ paths given a sentence and a predicate. Therefore, we adopt the max pooling operation to obtain a representation vector over an SDP. Following Xu et al. (2015), we divide the SDP into two parts in the position of the lowest common ancestor, in order to distinguish the directions, as shown in Equation 6 and Figure 4.

$$\boldsymbol{x}_i^{\text{syn}} = \underset{j \in \texttt{path}(i,a)}{\text{MaxPool}}\left(\boldsymbol{l}_j\right) \oplus \underset{k \in \texttt{path}(p,a)}{\text{MaxPool}}\left(\boldsymbol{l}_k\right) \quad (6)$$

where $\texttt{path}(i, a)$ is the set of all the words along the path from the focused word w_i to the lowest common ancestor w_a, and $\texttt{path}(i, a)$ is for the path from the predicate w_p to w_a; $\boldsymbol{l}_j$ is the embedding of the dependency relation label between w_j and its parent.

The TPF Approach

Collobert et al. (2011) first propose position features for the task of SRL. The basic idea is to use the embedding of the distance (as discrete numbers) between the predicate word and the focused word as extra inputs.

In order to use syntactic trees, Yang et al. (2016) extend the position features and propose the tree-based position features (TPF) for the task of relation classification.

In this work, we directly adopt the TPF approach of Yang et al. (2016) for encoding syntactic knowledge.[1] Figure 5 gives an example. The number pairs in the parentheses are the TPFs of the corresponding words. The first number means the distance from the predicate in concern to the lowest common ancestor, and the second number is the distance from the focused word to the ancestor. For instance, suppose "Ms." is the focused word and "plays" is the predicate. Their lowest common ancestor is "plays", which is the predicate itself. There are 2 dependencies in the path from "Ms." to "plays". Therefore, the TPF of "Ms." is "(0, 2)".

Then, we embed the TPF of each word into a dense vector through a lookup operation, use it as the syntax-related representation.

$$\boldsymbol{x}_i^{\text{syn}} = \boldsymbol{emb}_{f_i}^{\text{TPF}} \quad (7)$$

where f_i is the TPF of w_i.

[1] Yang et al. (2016) propose two versions of TPF. We directly use the Tree-based Position Feature 2 due to its better performance.

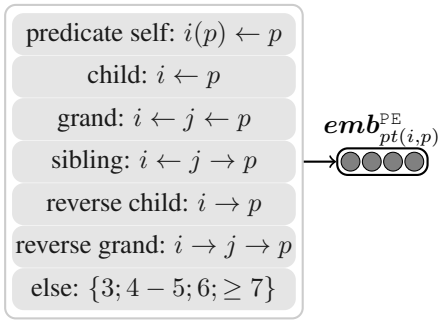

Figure 6: Patterns used in this work.

	Train	Dev	WSJ Test	Brown Test
#Sent	39,832	1,346	2,416	426
#Tok	950,028	32,853	56,684	7,159
#Pred	90,750	3,248	5,267	804
#Arg	239,858	8,346	14,077	2,177

Table 1: Data statistics of the CoNLL-2005.

The PE Approach

Jiang et al. (2018) propose the PE approach for the task of treebank conversion, which aims to convert a parse tree following the source-side annotation guideline into another tree following the target-side guideline. The basic idea is to classify the relationship between two given words in the source-side tree into several pre-defined patterns (or types), and use the embedding of the pattern and a few dependency relations as the source-side syntax representation.

In this work, we adopt their PE approach for encoding syntactic knowledge. Figure 6 lists the patterns used in this work. Given a focused word w_i and a predicate w_p, we first decide the pattern type according to the structural relationship between w_i and w_p, denoted as $pt(i, p)$. Taking $w_i =$"Ms." and $w_p =$ "plays" as an example, since "Ms." is the grandchild of "plays", then the pattern is "grandchild".

Then we embed $pt(i, p)$ into a dense vector $\boldsymbol{emb}^{\text{PE}}_{pt(i,p)}$. We also use the embeddings of three highly related syntactic labels as extra representations, i.e., $\boldsymbol{l}_i$, $\boldsymbol{l}_a$, $\boldsymbol{l}_p$, where $\boldsymbol{l}_a$ is the syntactic label between the lowest common ancestor w_a and its father. Then, the four embeddings are concatenated as $\boldsymbol{x}_i^{\text{syn}}$ to represent the structural information of w_i and w_p in a dependency tree.

$$\boldsymbol{x}_i^{\text{syn}} = \boldsymbol{emb}^{\text{PE}}_{pt(i,p)} \oplus \boldsymbol{l}_i \oplus \boldsymbol{l}_a \oplus \boldsymbol{l}_p \quad (8)$$

Experiments

Settings

Data Following previous works, we adopt the CoNLL-2005 dataset with the standard data split: sections 02-22 of the Wall Street Journal (WSJ) corpus as the training dataset, section 24 as the development dataset, section 23 as the in-domain test dataset, sections 01-03 of the Brown corpus as the out-of-domain test dataset (Carreras and Màrquez, 2005). Table 1 shows the statistics of the data.

	Methods	WSJ Test				Brown Test				Combined
		P	R	F1	Comp.	P	R	F1	Comp.	F1
Single	Baseline (He et al., 2017)	83.1	83.0	83.1	64.3	72.9	71.4	72.1	44.8	81.6
	Baseline (Our re-impl)	83.4	83.0	83.2	64.9	72.3	70.8	71.6	44.3	81.6
	Tree-GRU	83.9	83.6	83.8	65.2	72.9	71.1	72.3	44.9	82.2
	SDP	84.2	**83.9**	84.0	**65.9**	**74.0**	72.0	**73.0**	45.2	**82.6**
	TPF	**84.3**	83.8	**84.1**	**65.9**	73.7	72.0	72.9	45.5	**82.6**
	PE	83.7	83.8	83.8	65.3	73.4	**72.5**	**73.0**	**46.1**	82.3
Single+ELMo	Baseline	86.3	86.2	86.3	69.4	75.2	74.3	74.7	48.1	84.8
	Tree-GRU	86.2	86.2	86.2	68.9	77.9	75.6	76.7	50.8	84.9
	SDP	86.9	86.7	86.8	70.2	**78.0**	76.3	**77.1**	**52.4**	85.5
	TPF	**87.0**	**86.8**	**86.9**	**70.4**	77.6	75.9	76.8	51.6	**85.6**
	PE	86.5	86.3	86.4	69.6	77.4	**76.4**	76.9	51.5	85.1
Ensemble	5× Baseline (He et al., 2017)	85.0	84.3	84.6	66.5	74.9	72.4	73.6	46.5	83.2
	5× Baseline (Our re-impl)	84.6	84.0	84.3	66.4	74.9	72.1	73.5	46.1	82.9
	5× TPF	85.6	85.0	85.3	68.1	75.9	73.4	74.8	48.1	83.9
	4 Syntax-aware Methods	**85.8**	**85.5**	**85.6**	**68.7**	**76.3**	**74.5**	**75.4**	**49.3**	**84.3**
Ensemble+ELMo	5× Baseline	87.2	86.8	87.0	70.8	77.7	75.8	76.7	50.4	85.6
	5× TPF	87.5	87.0	87.3	71.1	78.6	76.5	77.5	52.5	86.0
	4 Syntax-aware Methods	**88.0**	**87.6**	**87.8**	**72.2**	**79.7**	**78.0**	**78.8**	**53.2**	**86.6**

Table 2: Comparison with baseline model and all our syntax-aware methods on the CoNLL-2005 dataset. We report the results in precision (P), recall (R), F1 and percentage of completely correct predicates (Comp.).

Dependency Parsing In recent years, neural network based dependency parsing has achieved significant progress. We adopt the state-of-the-art biaffine parser proposed by Dozat and Manning (2017) in this work. We use the original phrase-structure Penn Treebank (PTB) data to produce the dependency structures for the CoNLL-2005 SRL data. Following standard practice in the dependency parsing community, the phrase-structure trees are converted into Stanford dependencies using the Stanford Parser v3.3.0[2]. Since the biaffine parser needs part-of-speech (POS) tags as inputs, we use an in-house CRF-based POS tagger to produce automatic POS tags on all the data. After training, the biaffine parser achieves 94.3% parsing accuracy (LAS) on the WSJ test dataset. Additionally, we use the 5-way jackknifing to obtain the automatic POS tagging and dependency parsing results of the training data.

ELMo Peters et al. (2018) recently propose to produce contextualized word representations (ELMo) with an unsupervised language model learning objective, and show that simply using the learned external word representations as extra inputs can effectively boost performance of a variety of tasks, including SRL. To further investigate the effectiveness of our syntax-aware methods, we build a stronger baseline with the ELMo representations as extra input.

Evaluation We adopt the official script provided by CoNLL-2005[3] for evaluation. We conduct significance test using the Dan Bikel's randomized parsing evaluation comparer.

Implementation We implement the baseline and all the syntax-aware methods with Pytorch 0.3.0[4].

Initialization and Hyper-parameters For the parameter settings, we mostly follow the work of He et al. (2017). We adopt the Adadelta optimizer with learning rate $\rho = 0.95$ and $\epsilon = 1e-6$, use a batchsize of 80, and clip gradients with norm larger than 1.0. All the embedding dimensions (word, predicate-indicator, syntactic label, pattern, and TPF) are set to 100. All models are trained for 500 iterations on the trained data and select the best iteration that has the peak performance on the dev data.

Main Results

Table 2 shows the main results of different approaches on CoNLL-2005 dataset. The results are presented in four major rows. For the ensemble of "5 × baseline" and "5 × TPF", we randomly sample 4/5 of the training data and train one model at each time. For the ensemble of 4 syntax-aware approaches, each model is trained on the whole training data.

Results of single models are shown in the first major row. First, our re-implemented baseline of He et al. (2017) achieves nearly the same results with those reported in their paper. Second, the four syntax-aware approaches are similarly effective and can improve the performance by $0.6 \sim 1.0$ in F1 (combined). All the improvements are statistically

[2] https://nlp.stanford.edu/software/lex-parser.html

[3] http://www.cs.upc.edu/~srlconll/st05/st05.html

[4] github.com/KiroSummer/Syntax-aware-Neural-SRL

significant ($p < 0.001$). Third, we find that the Tree-GRU approach is slightly inferior to the other three. The possible reason is that Tree-GRU produces unified representations for the sentence without specific considerations of the given predicate, whereas all other three approaches derive predicate-specific representations.

Results of single models with ELMo are shown in the second major row, in which each single model is enhanced using the ELMo representations as extra inputs. We can see that ELMo representations brings substantial improvements over the corresponding baselines by $2.7 \sim 3.2$ in F1 (combined). Compared with the stronger baseline w/ ELMo, SDP, TPF, and PE w/ ELMo still achieve absolute and significant ($p < 0.001$) improvement of 0.7, 0.8, and 0.3 in F1 (combined), respectively. Tree-GRU w/ ELMo increases F1 by 2.0 on the out-of-domain Brown test data, but decreases F1 by 0.1 on the in-domain WSJ test data, leading to an overall improvement of 0.1 on combined F1 ($p > 0.05$). Similarly to the results in the first major row, this again indicates that the predicate-independent Tree-GRU approach may not be suitable for syntax encoding in our SRL task, especially when the baseline is strong.

Results of ensemble models are shown in the third major row. Compared with the baseline single model, the baseline ensemble approaches increase F1 (combined) by 1.3. The F1 score of the "5 $\times$ Baseline" of (He et al., 2017) is 83.2%, whereas our re-implemented "5 $\times$ Baseline" achieves 82.9% F1. We guess the 0.3 gap may be caused by the random factors in 5-fold data split and parameter initialization. In our preliminary experiments, we have run the single model of He et al. (2017) for several times using different random seeds for parameter initialization, and found about $0.1 \sim 0.3$ F1 variation. The ensemble of five TPF models further improves F1 (combined) over the ensemble of five baselines by 1.0 ($p < 0.001$). We choose the TPF method as a case study since it consistently achieves best performances in all scenarios. The ensemble of the four syntax-aware methods achieves the best performance in this scenario, outperforming the TPF ensemble by 0.4 F1. This indicates that the four syntax-aware approaches are discrepant and thus more complementary in the ensemble scenario than using a single method.

Results of ensemble models with ELMo are shown in the bottom major row, where each single model is enhanced with ELMo representations before the ensemble operation. Again, using ELMo representations as extra input greatly improves F1 (combined) by $2.1 \sim 2.7$ over the corresponding ensemble methods in the third major row. Similar to the above findings, the ensemble of five TPF with ELMo improve F1 (combined) by 0.4 ($p < 0.001$) over the ensemble of five baselines with ELMo, and the ensemble of the four syntax-aware methods with ELMo further increases F1 (combined) by 0.6.

Overall, we can conclude that the syntax-aware approaches can consistently improve SRL performance.

		WSJ	Brown	Combi
Single	TPF w/ ELMo	**86.9**	**76.8**	**85.6**
	TPF	84.1	72.9	82.6
	Strubell et al. (2018)	83.9	72.6	-
	He et al. (2017)	83.1	72.1	81.6
	Tan et al. (2018)	84.8	74.1	83.4
Ensemble	4 Syn-a Methods w/ ELMo	**87.8**	**78.8**	**86.6**
	4 Syntax-aware Methods	85.6	75.4	84.3
	He et al. (2017)	84.6	73.6	83.2
	Tan et al. (2018)	86.1	74.8	84.6
	FitzGerald et al. (2015)	80.3	72.2	-

Table 3: Comparison with previous results.

Methods	Devel	WSJ	Brown	Combined
Baseline	81.5	83.2	71.6	81.6
TPF	82.5	84.1	72.9	82.6
TPF-Gold	88.4	89.6	79.8	88.3
Baseline w/ ELMo	85.5	86.3	74.7	84.8
TPF w/ ELMo	85.3	86.9	76.8	85.6
TPF-Gold w/ ELMo	89.8	91.1	82.2	89.9

Table 4: Upper-bound analysis of the TPF Method.

Comparison with Previous Works

Table 3 compare our approaches with previous works. In the single-model scenario, the TPF approach outperforms both He et al. (2017) and Strubell et al. (2018), and is only inferior to Tan et al. (2018), which is based on the recently proposed deep self-attention encoder (Vaswani et al., 2017). Using ELMo representations promotes our results as the state-of-the-art. We discuss the work of Strubell et al. (2018) in the related works section, which also makes use of syntactic information.

In the ensemble scenario, the findings are similar, and the ensemble of four syntax-aware approaches with ELMo reaches new state-of-the-art performances.

Analysis

In this section, we conduct detailed analysis to better understand the improvements introduced by the syntax-aware approaches. We use the TPF approach as a case study since it consistently achieves best performances in all scenarios. We sincerely thank Luheng He for the kind sharing of her analysis scripts.

Using Gold-standard Syntax To understand the upper-bound performance of the syntax-aware approaches, we use the gold-standard dependency trees as the input and apply the TPF approach. Table 4 shows the results. The TPF method with gold-standard parse trees brings a very large improvement of 6.7 in F1 (combined) over the baseline without using ELMo, and 5.1 when using ELMo. This shows the usefulness and great potential of syntax-aware SRL approaches.

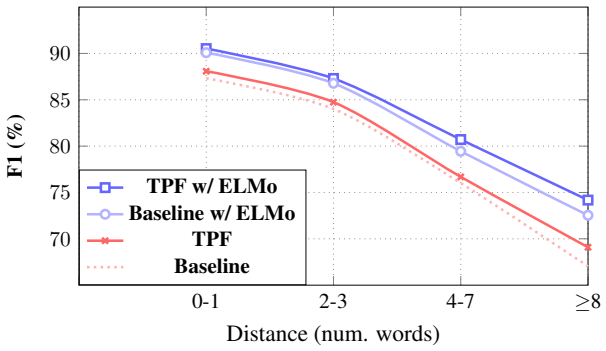

Figure 7: F1 regarding surface distance between arguments and predicates.

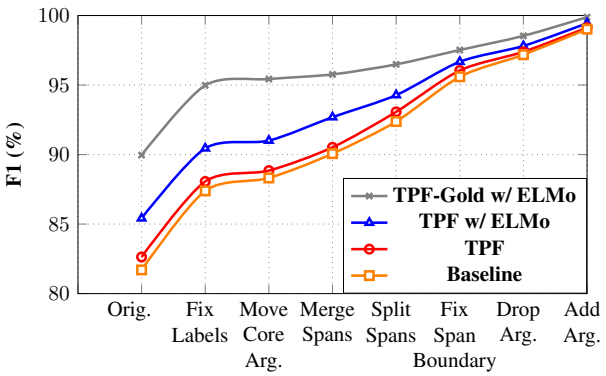

Figure 8: Performance of CoNLL-2005 models after performing oracle transformations.

Long-distance Dependencies To analyze the effect of syntactic information regarding to the distances between arguments and predicates, we compute and report F1 scores of different sets of arguments according to their distances from predicates, as shown in Figure 7. It is clear that larger improvements are obtained for arguments longer-distance arguments, in both scenarios of with or without ELMo representations. This demonstrates that *syntactic knowledge effectively captures long-distance dependencies and thus is most beneficial for arguments that are far away from predicates.*

Error Type Breakdown In order to understand error distribution of different approaches in terms of different types of mistakes, we follow the work of He et al. (2017) and employ a set of oracle transformations on the system outputs to observe the relative F1 improvements by fixing various prediction errors incrementally. "Orig" corresponds to the F1 scores of the original model outputs. First, we fix the label errors in the model outputs and the fixed results are shown by "Fix Labels". Specifically, if we find a predicted span matches a gold-standard one but has a wrong label, then we assign the correct label to the span in the model outputs. Then, based on the results of "Fix Labels", we perform "Move Core Arg.". If a span is labeled as a core argument (i.e., A0-A5), but the boundaries are wrong, then we move the span to its correct boundaries. Third, based on the results of "Move Core Arg.", we perform "Merge Spans". If two predicted spans can be merged to match a gold-standard span, then we do so and assign the correct label. Fourth, we preform "Split Spans. If a predicted span can be split into two gold-standard spans, then we do so and assign correct labels to them. Fifth, if a predicted span's label matches an overlapping gold span, we perform "Fix Span Boundary" to correct its boundary. Sixth, we perform "Drop Arg." to drop the predicted arguments that doesn't overlap with any gold spans. Finally, if a gold argument doesn't overlap with any predicted spans, we perform "Add Arg." Figure 8 shows the results, from which we can see that 1) different approaches have similar error distributions, among which labeling errors account for the largest proportion, and span boundary errors (split, merge, boundary) also have a large share; 2) using automatic parse trees leads to consistent improvements over all error types; 3) using ELMo representations also consistently improves performance by large margin; 4) using gold parse trees can effectively resolve almost all span boundary (including merge and split) errors.

Related Work

Traditional discrete-feature based SRL models make heavy use of syntactic information (Swanson and Gordon, 2006; Punyakanok, Roth, and Yih, 2008). With the rapid development of deep learning in NLP, researchers propose several simple yet effective end-to-end neural network models with little consideration of syntactic knowledge (Zhou and Xu, 2015; He et al., 2017; Tan et al., 2018).

Meanwhile, inspired by the success of syntactic features in traditional SRL approaches, researchers also try to enhance neural network based SRL approaches by syntax. He et al. (2017) show that large improvement can be achieved by using gold-standard constituent trees as rule-based constraints during viterbi decoding. Strubell et al. (2018) propose a syntactically-informed SRL approach based on the self-attention mechanism. The key idea is introduce an auxiliary training objective that encourages one attention head to attend to its syntactic head word. They also use multi-task learning on POS tagging, dependency parsing, and SRL to obtain better encoding of the input sentence. We make comparison with their results in Table 3. Swayamdipta et al. (2018) propose a multi-task learning framework to incorporate constituent parsing loss into other semantic-related tasks such as SRL and coreference resolution. They report +0.8 F1 improvement over their baseline SRL model. Different from Swayamdipta et al. (2018), our work focuses on dependency parsing and try to explicitly encode parse outputs to help SRL.

The dependency-based SRL task is started in CoNLL-2008 shared task (Surdeanu et al., 2008), which aims to jointly tackle syntactic and semantic dependencies. There are also a few recent works on exploiting dependency trees for neural dependency-based SRL. Roth and Lapata (2016) proposes an effective approach to obtain dependency path embeddings and uses them as extra features in traditional discrete-feature based SRL. Marcheggiani and Titov (2017)

propose a dependency tree encoder based on a graph convolutional network (GCN), which has a similar function as Tree-GRU, and stack the tree encoder over a sentence encoder based on multi-layer BiLSTMs. He et al. (2018) exploit dependency trees for dependency-based SRL by 1) using dependency label embeddings as extra inputs, and 2) employing a tree-based k-th order algorithm for argument pruning. Cai et al. (2018) propose a strong end-to-end neural approach for the dependency-based SRL based on deep BiLSTM encoding and Biaffine attention. They use dependency trees for argument pruning and find no improvement over the syntax-agnostic counterpart.

Conclusions

This paper makes a thorough investigation and comparison of four different approaches, i.e., Tree-GRU, SDP, TPF, and PE, for exploiting syntactic knowledge for neural SRL. The experimental results show that syntax is consistently helpful to improve SRL performance, even when the models are enhanced with external ELMo representations. By utilizing both ELMo and syntax-aware representations, our final models achieve new state-of-the-art performance in both single and ensemble scenarios on the benchmark CoNLL-2005 dataset. Detailed analyses show that syntax helps the most on arguments that are far away from predicates, due to the long-distance dependencies captured by syntactic trees. Moreover, there is still a large performance gap between using gold-standard and automatic parse trees, indicating that there is still large room for further research on syntax-aware SRL.

Acknowledgments

We thank our anonymous reviewers for their helpful comments. This work was supported by National Natural Science Foundation of China (Grant No. 61525205, 61876116, 61432013), and was partially supported by the joint research project of Alibaba and Soochow University.

References

Cai, J.; He, S.; Li, Z.; and Zhao, H. 2018. A full end-to-end semantic role labeler, syntax-agnostic over syntax-aware? In *Proceedings of COLING*, 2753–2765.

Carreras, X., and Màrquez, L. 2005. Introduction to the conll-2005 shared task: Semantic role labeling. In *Proceedings of CoNLL*, 152–164.

Chen, H.; Huang, S.; Chiang, D.; and Chen, J. 2017. Improved neural machine translation with a syntax-aware encoder and decoder. In *Proceedings of ACL*, 1936–1945.

Collobert, R.; Weston, J.; Bottou, L.; Karlen, M.; Kavukcuoglu, K.; and Kuksa, P. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research* 12:2493–2537.

Dozat, T., and Manning, C. D. 2017. Deep biaffine attention for neural dependency parsing. In *Proceedings of ICIR*.

Feng, Y.; Zhang, H.; Hao, W.; and Chen, G. 2017. Joint extraction of entities and relations using reinforcement learning and deep learning. *Computational Intelligence and Neuroscience* 2017:1–11.

FitzGerald, N.; Täckström, O.; Ganchev, K.; and Das, D. 2015. Semantic role labeling with neural network factors. In *Proceedings of EMNLP*, 960–970.

Gao, Q., and Vogel, S. 2011. Corpus expansion for statistical machine translation with semantic role label substitution rules. In *Proceedings of ACL*, 294–298.

Genest, P.-E., and Lapalme, G. 2011. Framework for abstractive summarization using text-to-text generation. In *Proceedings of MTTG*, 64–73.

He, L.; Lee, K.; Lewis, M.; and Zettlemoyer, L. 2017. Deep semantic role labeling: What works and whats next. In *Proceedings of ACL*, 473–483.

He, S.; Li, Z.; Zhao, H.; and Bai, H. 2018. Syntax for semantic role labeling, to be, or not to be. In *Proceedings of ACL*, 2061–2071.

Jiang, X.; Li, Z.; Zhang, B.; Zhang, M.; Li, S.; and Si, L. 2018. Supervised treebank conversion: Data and approaches. In *Proceedings of ACL*, 2705–2715.

Khan, A.; Salim, N.; and Jaya Kumar, Y. 2015. A framework for multi-document abstractive summarization based on semantic role labelling. *Applied Soft Computing* 30(C):737–747.

Liu, D., and Gildea, D. 2010. Semantic role features for machine translation. In *Proceedings of COLING*, 716–724.

Marcheggiani, D., and Titov, I. 2017. Encoding sentences with graph convolutional networks for semantic role labeling. In *Proceedings of ACL*, 1506–1515.

Meyers, A.; Reeves, R.; Macleod, C.; Szekely, R.; Zielinska, V.; Young, B.; and Grishman, R. 2004. The nombank project: An interim report. In *Proceedings of HLT-NAACL*.

Miwa, M., and Bansal, M. 2016. End-to-end relation extraction using lstms on sequences and tree structures. In *Proceedings of ACL*, 1105–1116.

Palmer, M.; Gildea, D.; and Kingsbury, P. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational linguistics* 31(1):71–106.

Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; and Zettlemoyer, L. 2018. Deep contextualized word representations. In *Proceedings of NAACL-HLT*, 2227–2237.

Punyakanok, V.; Roth, D.; and Yih, W.-t. 2008. The importance of syntactic parsing and inference in semantic role labeling. *Computational Linguistics* 34(2):257–287.

Roth, M., and Lapata, M. 2016. Neural semantic role labeling with dependency path embeddings. In *Proceedings of ACL*, 1192–1202.

Shen, D., and Lapata, M. 2007. Using semantic roles to improve question answering. In *Proceedings of EMNLP-CoNLL*, 12–21.

Srivastava, R. K.; Greff, K.; and Schmidhuber, J. 2015. Training very deep networks. In *Proceedings of NIPS*, 2377–2385.

Strubell, E.; Verga, P.; Andor, D.; Weiss, D.; and McCallum, A. 2018. Linguistically-informed self-attention for semantic role labeling. In *Proceedings of EMNLP*, 5027–5038.

Surdeanu, M.; Johansson, R.; Meyers, A.; Màrquez, L.; and Nivre, J. 2008. The conll-2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proceedings of CoNLL*, 159–177.

Swanson, R., and Gordon, A. S. 2006. A comparison of alternative parse tree paths for labeling semantic roles. In *Proceedings of COLING/ACL*, 811–818.

Swayamdipta, S.; Thomson, S.; Lee, K.; Zettlemoyer, L.; Dyer, C.; and Smith, N. A. 2018. Syntactic scaffolds for semantic structures. In *Proceedings of EMNLP*, 3772–3782.

Tai, K. S.; Socher, R.; and Manning, C. D. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of ACL*, 1556–1566.

Tan, Z.; Wang, M.; Xie, J.; Chen, Y.; and Shi, X. 2018. Deep semantic role labeling with self-attention. In *Proceedings of AAAI*, 4929–4936.

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 *Proceedings of NIPS*, 5998–6008.

Wang, H.; Bansal, M.; Gimpel, K.; and McAllester, D. 2015. Machine comprehension with syntax, frames, and semantics. In *Proceedings of ACL-IJCNLP*, 700–706.

Wang, Y.; Li, S.; Yang, J.; Sun, X.; and Wang, H. 2018. Tag-enhanced tree-structured neural networks for implicit discourse relation classification. In *Proceedings of IJCNLP*, 496–505.

Xu, Y.; Mou, L.; Li, G.; Chen, Y.; Peng, H.; and Jin, Z. 2015. Classifying relations via long short term memory networks along shortest dependency paths. In *Proceedings of EMNLP*, 1785–1794.

Yang, Y.; Tong, Y.; Ma, S.; and Deng, Z.-H. 2016. A position encoding convolutional neural network based on dependency tree for relation classification. In *Proceedings of EMNLP*, 65–74.

Zhang, Y.; Chen, G.; Yu, D.; Yaco, K.; Khudanpur, S.; and Glass, J. 2016. Highway long short-term memory rnns for distant speech recognition. In *Proceedings of ICASSP*, 5755–5759.

Zhou, J., and Xu, W. 2015. End-to-end learning of semantic role labeling using recurrent neural networks. In *Proceedings of ACL-IJCNLP*, 1127–1137.